

A SIMPLE ORDERING SOLUTION FOR EFFICIENT GEOMETRICAL LEVEL OF DETAILS ON LIDAR POINT CLOUD

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

1. ABSTRACT

We introduce a new ordering for point cloud based on octree, which allows, after a pre-computing phase, to effortlessly get a portable representative geometric Level Of Details (LOD). 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 embed information about the sensed object geometric nature, and can also be used as a crude dimensionality descriptor, enabling fast classification and on-the-fly filtering with 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 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.

Reducing the number of point is then a vital need for practical point cloud management.

There are basically two levers to reduce the amount of data considered. The first would be **filtering** data based on 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 them well. Those objects can be more abstract (geometric primitives like plane for example), or simply some well chosen points (subset).

For instance, to visualize the parts of a massive point cloud, we might fetch only the points that are visible (filtering), and not display all the points but only the sufficient number to understand the scene (generalisation). (See Figure 2)

The system we propose un ? is compatible with both, and extensively covers the filtering and abstracting part. In this article we focus on the generalisation problem using points.

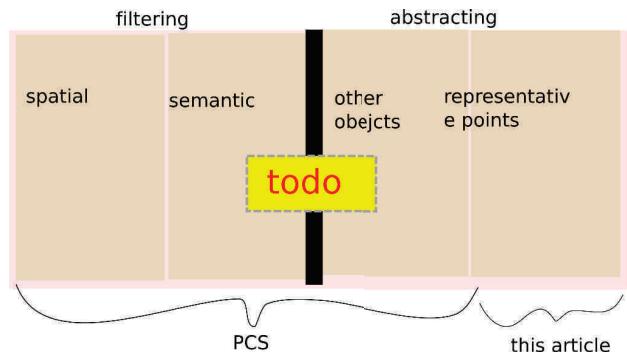


Figure 2: Two strategies to limit the amount of points to work on.

Thus we deal with a simpler version of a problem common for Geographical Information System (GIS): how to generalize information contained in huge datasets, that is reduce the details of the data while retaining its principal organisation.

This problematic is very common across research field, and could be seen as a problem of compression, clustering, dimensionality reduction, etc.

This particular problem applied to visualisation is commonly called Level Of Details (LOD, cf figure 1).

The problem is then how to reduce the amount of points while preserving the geometric characteristics of the underlying sensed object, in an efficient manner, and while being robust to the way the sensing was done? Indeed LOD approaches sacrifice a part of information in exchange of a massive reduction of data size which is necessary for efficient usage/visualisation. In this regard, a LOD method must by nature be efficient, or the information loss would be pointless.

2.2 Related Work

RC:2.2.0. ajouter citation de bereuter ? , ? ?

RC:2.2.0. parler de Latin Hypercube sampling : c est le nom savant pour faire une grille 3D et prendre un échantillon max par voxel

RC:2.2.0. parler de Rainville2012, un bon résumé de ce qui se fait

RC:2.2.0. parler de Halton sequence, low discrepancy comme autre candidats à sampler uniformément 1 espace avec un peu d'aléatoire

RC:2.2.0. parler de l'alternative : reconstruire une surface, interpoler les attributs dessus, puis choisir le point centrale ~c depend de la présence d'une surface; ou méthode d'optimisation, comment construire le point qui minimisera l erreur

Fil rouge à raconter : on veut en fait sampler les fonctions x,y,z, attributs. Ce dont on est sur c est x,y,z, donc dans l ideal on

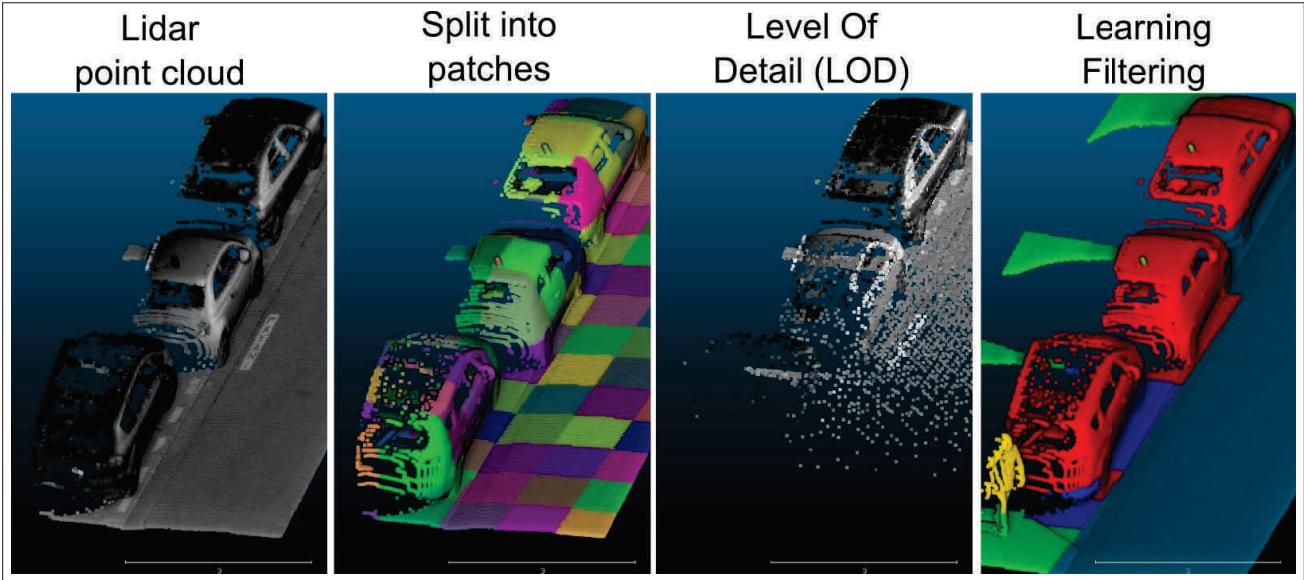


Figure 1: Graphical Abstract : a Lidar point cloud (1), is split it into patches (2) and stored in the PCS (?), patches are re-ordered to obtain free LOD (3 : a gradient of LOD here). Lastly the ordering is used as a feature for learning and efficient filtering (4)

trouverait le sampling sur la surface qui minimise la perte d erreur. Comme on est même pas sur d avoir une surface, on peut pas faire ça. On a même pas un sampling uniforme, ce qui nous interdit d utiliser les moindres carrées (pas le choix, c est la variation de densité qui le veut). A la place on prend l autre approche, on crée un sampling régulier, et comme on veut pas se dépatouiller des attributs, on prend un point déjà existant. De plus on veut pouvoir être utile à plusieurs applications, donc on cherche pas le meilleur lod pour une application, mais un bon lod pour toutes les applications.

Because points are not only 3D but may also have attributes, a perfect ordering would require extensive knowledge of those attributes and of the usage to be made with the point cloud. For instance let's consider a point cloud containing a classification, and suppose the application is to 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, as such the optimal ordering strategy would need to depend on this classification attributes. Orthogonally, even when using only the geometry to order points, the ordering may again depend on the application. For instance an application of visualisation of point cloud of man-made objects would probably benefit from ordering points based on curvature (as an approximation of visual saliency).

RC:2.2.0. parler de ? en précisant que visualisation et généralisation peuvent être mélangés, et que ce n'est pas toujours une question de performance, mais de choisir ce que l'utilisateur voit (similaire au flou au cinéma)

One way to tackle data size is to use a Level Of Detail strategy. Octree methods have been common in computer graphics for several decades ([Meagher, 1982](#)). They are used in many methods to speed computing, or compress data, like in the work of [Schnabel and Klein \(2006\)](#); [Huang et al. \(2006\)](#) (which have not been designed to scale).

We can see point cloud as a sampling of a surface (or volume) of the sensed object. This 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 figure 3).

[Elseberg et al. \(2013\)](#) give a good overview of octree usage for point clouds. Their method proposes to directly store the points

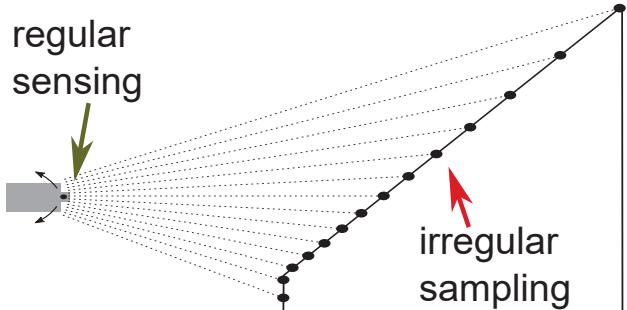


Figure 3: Regular sensing does not imply regular sampling.

and octree in a file. They explore many applications that could benefit from our method (visual LOD, registration, processing). We share many objectives. It is an extremely effective approach that also enables visual LOD, but is very specialized on geometry. In particular, this method is not integrated with other GIS data (Geographical Information System), point cloud fast querying is not possible on attributes or metadata, and point cloud format is extremely specific.

Another orthogonal way is to not use file, but store point cloud in DBMS. [van Oosterom \(2014\)](#) implements such system at very big scale and discuss how it can answer to various need. This approach has gained recent interest ([pgPointCloud \(2014\)](#)) because it is generic, it scales naturally to very large data set and is easier to implement than starting from scratch. We also observe that most of Big Data system use methods from the DBMS world. [Demantké \(2014\)](#) introduces a sophisticated per-point dimensionality descriptor, which is 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.

Random Forest method started with [Amit and Geman \(1997\)](#) and 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.

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. We propose a simple method that enables portable, computation-free, geometrical Level Of Detail. Our first contribution is to propose to store the LOD information directly into the ordering of 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 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 and free dimensionality descriptor. We demonstrate the interest of this descriptor 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 (?), 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 ??.

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 re-ordering 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 classification at the patch level (3.5). This classification can be directly transferred to points, or indirectly exploited in a pre-filtering step.

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).

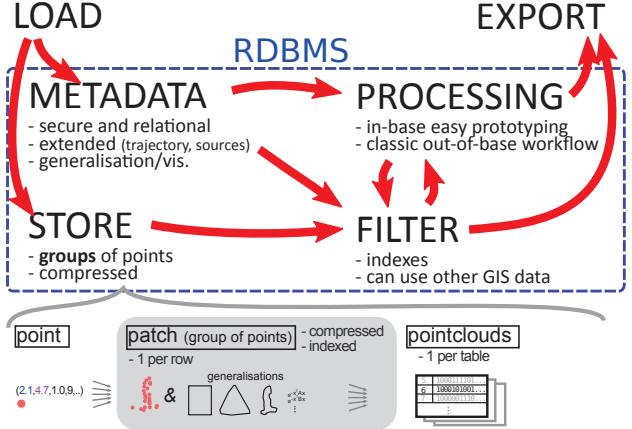


Figure 4: Overall and storage organisations of the Point Cloud Server.

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 1 m(Paris) or 50 m(Vosges) wide.

All the methods described in this article are applied on patches. So what 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, that when reading the points from the beginning to the end, we get gradually a more accurate geometrical approximation of the point cloud (see figure 5).

For instance, for a given list L of points ordered from 1 to N . Reading the points 1 to 5 is going to give a rough approximation of the point cloud, and reading another 16 points (points 1 to 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.

Leveraging the point ordering as LOD gives 3 main advantages. 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.

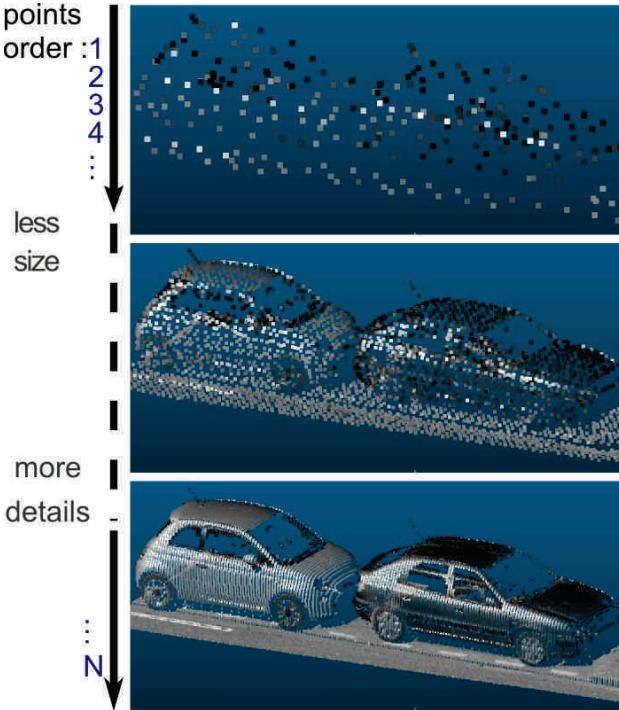


Figure 5: 3 geometrical Level Of Detail (LOD) for the same point cloud. Starting from very low detail by reading the first point of the patch, reading further points increases the details.

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 introduce any other points or data, it avoids all precision-related issues that may come from aggregating.

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 4.4.1). Even if a tool were to change the order, it is easy to add the ordering number as an attribute, which increases slightly the storage requirement, but is totally portable and can be used with all existing tools. This simplicity also implies that adapting tools to use 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 details we have on the point cloud. Obviously an ordering method that class the points from first LOD to last LOD is needed. So this ordering is in fact a 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 with different point cloud 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 of the point cloud, 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), not be sensitive to density variations, be regular, be fast to compute and be deterministic.

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 we can not rely on the variation of 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 squared distance to the points, and as such is the optimal point if the point probability are normal distributions.

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

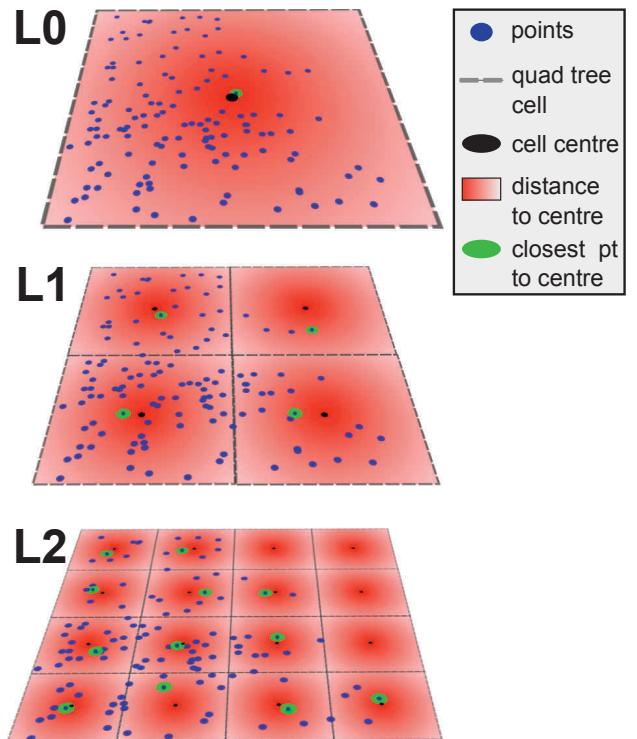


Figure 6: MidOc explained in 2D. Given a point cloud (Blue) and quad tree cells (dashed grey), the chosen point (green ellipse) is the one closest to the centre (black point) of the cell.

3.3.2 Introducing the MidOc ordering We propose to a re-use 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 by aggregating. 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. 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 point possible to pick. The process can be reached 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 3.3.3).

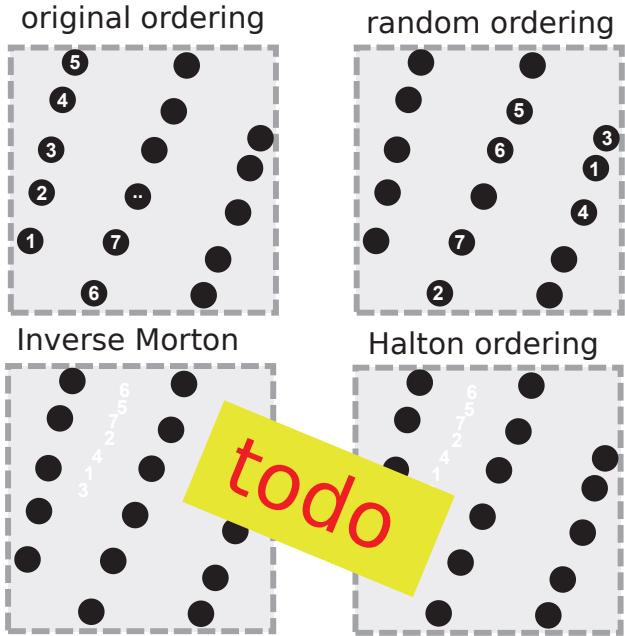


Figure 7: Several possible ordering with varying coverage.

3.3.3 Intra-level ordering 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. 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 the points of a designated level L_i , and compute online the best plan. See Figure 7 If the point The points are ordered either randomly or deterministically (for instance, with an inverted Z Morton curve (read backward),).

RC:3.3.3. ajouter illustrations et explication de l'importance du choix de l'ordonnancement inter level penser que Linfiny est aussi un niveau

3.3.4 Applications We see 3 type of application for this ordering :

- graphical LOD : service for point cloud visualisation.
- density correction : service for complex processing that may fail to deal with important density variation
- point cloud generalisation, service for processing that may use only a fraction of points.

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



Figure 8: All successive levels for common objects (car, window, 2 wheelers, light, tree, people, pole, ground), color is intensity for other points.

3.3.5 Implementation This method has the same complexity as an octree construction. Similar strategies can be followed,

pending on available resources.

The most straightforward implementation is streamed : The closest point to the center is stored in each octree cell. At the beginning the octree is empty. When receiving a point, traverse the octree update/creating cells and updating the closest point for each cell traversed. When the tree is computed, traverse it breadth-first to collect chosen points. The worst complexity is $O(N^*T)$, the entire octree must be stored in memory.

The simplest implementation use a recursive strategy. it only necessitate a method that given a cell and a list of points chose the point closest to the center 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.

We propose this kind of implementation in Python with extensive use of bit-level operations (cf 3.3.6) as proof of concept. The octree is not stored in memory, but it may prove difficult to parallelize (which is done at the data level anyway for our experiments).

3.3.6 Efficiency and performance 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 that this centring/scaling/quantization directly gives the octree. Moreover, further operations can be performed using bit arithmetic, which is extremely fast.

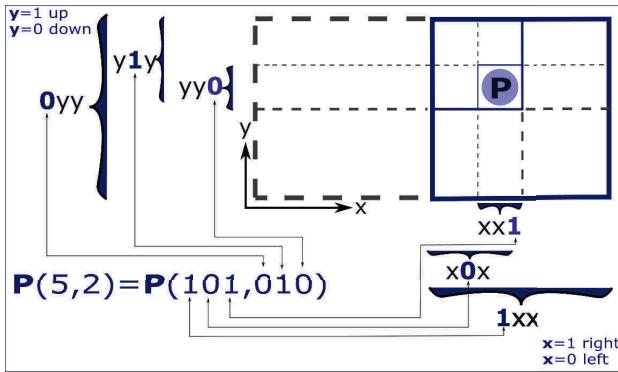


Figure 9: Principle of binary coordinates for a centered, scaled and quantized point cloud.

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

3.3.7 Advantages

Common, Simple and efficient This method feels classical and is based on Octree. This makes it simple to implement, and possibly extremely memory and CPU efficient.

The complexity is $O(n_{points} * Levels)$ or less. It is very simple to implement. Octree construction has been commonly done on GPU for more than a decade.

Fixed density This method can be used to guarantee an almost constant density for a given levels, even when the acquisition process produced varying-density point cloud. Thus using the output of this method is a safeguard for most of the complex point cloud processing methods that may be badly affected by extrem variation in density (real world case illustrated in [this illustration](#)).

Generic Few hypothesis are made on the points properties. In particular, this method works well with 2.5D point cloud (aerial single echo Lidar) and full 3D point cloud (urban 3D cloud with multi echo).

Construction Descriptors Lastly, the information given by this ordering can be used as a geometrical descriptor of the point cloud. The number of points per level for each level gives a crude dimensionality descriptor of the geometrical nature (at the patch scale) for the object contained by the point cloud.

3.4 Using the ordering by-product as a crude dimensionality descriptor

3.4.1 Principle During the ordering process the number of chosen point per each level can be stored. This number of chosen points per level gives an indication on the geometric nature of the object in the point cloud. We demonstrate the use of this crude descriptor along with other simple feature with a Random Forest classifier on a real world dataset publicly available. This patch classification can be used in other ways than pure classification.

3.4.2 Conceptual example In the [figure 8](#), we manually segmented typical parts of a street in the Paris dataset: 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 the public light 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 plan, 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 $(1, 8, 64, 512)$ (if enough points), so a 8^L function.

3.4.3 Patch classifier applications The base method is to use Random Forest to classify patches (and not points directly).

Because patches may contains points belonging to several classes, transferring the patch classes to points naturally increases the errors.

We can foresee three type of application for patch classification.

Speeding a complex point classifier The first application could be to speed up and/or improve a complex per point classifier. For a speed up, the patch classifier could perform a first basic classification extremely fast, thus eliminating a large number of points, before the complex classifier is used. If necessary, it is possible to artificially increase recall at the cost of a diminution of precision, like in [19](#).

Improving a complex point classifier Patch classifier can also be used to improve result of a complex classifier by performing a first rough analysis which may determinate which complex classifier to use amongst several, like Cascaded classifier. 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

RC:3.3.7. si barycentre proche du centre du l'octree (ce qui arrive en moyenne), le point binaire est calculé.

especially precious for classes that are statistically very minoritary. If necessary it is possible to artificially increase precision at the cost of recall.

Filtering Another application is filtering for applications that only require one class. When the learning is done, classifying is extremely fast.

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 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 facade 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.

3.4.4 Descriptors

Crude dimensionality descriptor Again we work at the patch level (1^3 or 50^3 m^3). We order all patches following the MidOc ordering. For each ordered patch, we associate the number of points per level that where chosen. A Random Forest classifier is trained using the number of chosen points per level. We use the number of points for the level [1..4] included. For each level, the number of points is normalized by the maximum number of points possible (8^i), so every feature is in [0, 1].

Other simple features We also use other very simple features that require almost no computing. Feature usage is then analysed afterwards. For the sake of simplicity and efficiency, all the feature use basic statistics per patch that need to be precomputed anyway by the storage compression mechanism. This free statistics are min, max, and average of any attribute.

Contextual features are avoided. They are more in the spirit of complex classification, would require computing, and could introduce a bias (in our favor) in the result. However using the context after the classification is a lever to significantly improve recall.

3.4.5 Analyzing data set classes

Analysing class hierarchy The Paris data set classes are organized in a hierarchy (100 classes). Because of the hierarchy and the unbalancing of classes, we first determinate or similar the classes are for the simple descriptors. This information is extracted from the confusion matrix, which is used as an affinity matrix. The matrix is clustered by spectral clustering and the result are interpreted as a graph of classes. From this we choose the classes to use.

Balancing the data set We tried two classical strategies to balance the data set regarding the number of observation per class. The first is undersampling : we randomly undersample 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 prevalance. This weight is then used in the learning process.

3.4.6 Patch classifier To ensure significant results we follow a K-fold cross-validation method. We randomly split the observations into K parts, then for each part, we use the K-1 others to learn and predict on the part. All the evaluation are then performed on the total of predicted observations.

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. Similarly, 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.

3.4.7 Advantages

Dimensionality descriptor

- simple : Dimensionality feature for point clouds are already well researched, and can be more precisely computed ([Demantké \(2014\)](#)), with less sensibility to outliers (but more to density variation). However This kind of feature is generally designed at the point level, and is more complex. Using the result of the MidOc ordering has the advantage of being free and extremely simple.
- Efficient : Moreover, because $x_0 \rightarrow (2^0)^x$, $x_1 \rightarrow (2^1)^x$, $x_2 \rightarrow (2^2)^x$ diverge very fast, we only need to use few levels to have a quite good descriptor. For instance, using $L = 2$, we have $D = 4, 16$ or 64 , which are very distinguishable values, and don't require a density above 70 points /patch.
- Density and scale independent : As long as the patch contains a minimal number of points, the descriptors is density and scale invariant.
- Mixed result : 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).

Patch classification

- simple and fast When the Random Forest classifier is trained, prediction is extremely fast.
- good result Even if the classifier works on patch that may contain points from several classes, the global results for well represented classes are not far from state of the art.
- many applications teh classification is not necessarily interesting per see, but also for fast filtering or other applications.

3.5 Classification with the PCS

4. RESULT

4.1 Introduction to all experiments

We design and execute several experiments in order to validate all points that have been introduced in the "method" part. 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. We analyse the potential of this free descriptors, and what it brings when used in conjunction to other simple descriptors.

Software stack 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 \(2014\)](#).

4.2 Point Cloud server introduction

4.2.1 Principle

Server 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 groups of points along with some basic statistics about points in the group. Patch are compressed using various strategies. This organisation is based on the observation that in typical point cloud processing workflow, a point is never needed alone, but almost always with its surrounding points.

Fast filtering 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 we can find all points in a data set of 2 Billion points in a matter of milliseconds - between -1 and 3 meters high in reference to vehicle wheels - in a given 2D area defined by any polygon - close to a street called Rue Madame (according to IGN BDTopo) - between 3 and 5 meters to the sensor position - not in buildings according to Open Data Paris building layer - acquired in the second passage of the vehicle at this place - acquired between 8h and 8h10

Parallelism friendly Cutting a point cloud into patches provides also a very easy parallel processing possibility by data partition, which we extensively use in our experiments.

Point cloud splitting For our experiments we cut terrestrial Lidar point cloud into 1 m^3 cubes oriented on (North ,Est,Z) axis. We cut aerial lidar point cloud into 50^3 m^3 cubes. The choice of size is a compromise between speed, index size, patch size, typical feature size, etc. In fact the patch can be cut arbitrary, we chose this splitting for simplicity.

4.3 Data set used

We use two data sets. There were chosen as different as possible to further evaluate how proposed methods can generalise on different data.

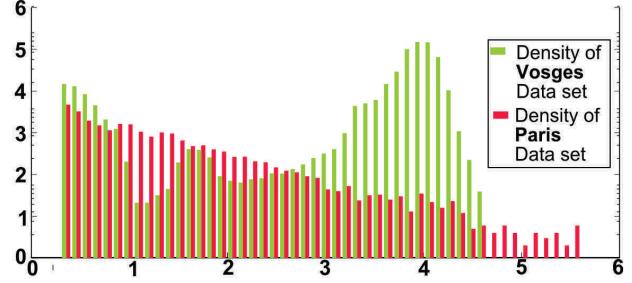


Figure 10: Histogram of number of points per patch, with a logarithmic scale for X and Y axis

4.3.1 IQmulus data set First [IQmulus \(2014\)](#), an open source urban data set with varying density, singularities, and very challenging point cloud geometry. Data set is about 600 Millions points , over 12 kms of road. Points are typically spaced by 5cm to 0.2 cm. It is a multi echo laser (Riegl). We have access to a training set where every point is labeled in a hierarchy of 100 classes. The training set is only 12 millions points. Only 22 classes are represented. We will refer to this data set as Paris data set.

4.3.2 Vosges data set We also use the Vosges data set, which is a very wide spread aerial data set of 6 billions points.

Density is much more constant to 10k pts/patch . We have access to a vector ground truth about surface occupation nature (type of forest), produced by the French Forest Agency. We will refer to this data set as the Vosges data set.

4.4 Exploiting the order of points

4.4.1 Experiment summary 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. Then for each processing software, we read the reference file and convert it into another format, then check that the conversion didn't change the order. The tree common open source software tested are CloudCompare, LasTools and Meshlab. This test point cloud is available in supplementary materials.

4.4.2 Results All software passes test. We stress that even if software change order, it is still very easy to add the order as an attribute, thus making it fully portable.

4.5 MidOc: an ordering for gradual geometrical approximation

4.5.1 Experiment summary In this experiment we first test ordering on typical street objects,then on terrestrial dataset to visually appreciate the fitness to use it for geometrical LOD. Then we compute MidOc for both our dataset and evaluate the trade-off between point cloud size and point cloud LOD. We briefly consider computing bottleneck.

We demonstrate an immediate application of LOD for fast abnormal density detection and correction. Lastly as a proof of concept we stream 3D point cloud with various LOD to a browser.

4.5.2 Visual evaluation

Visual evaluation on typical objects (ground, facade, car, pole, vegetation) The figure 8 illustrates LOD on common street objects, of various dimensionality.

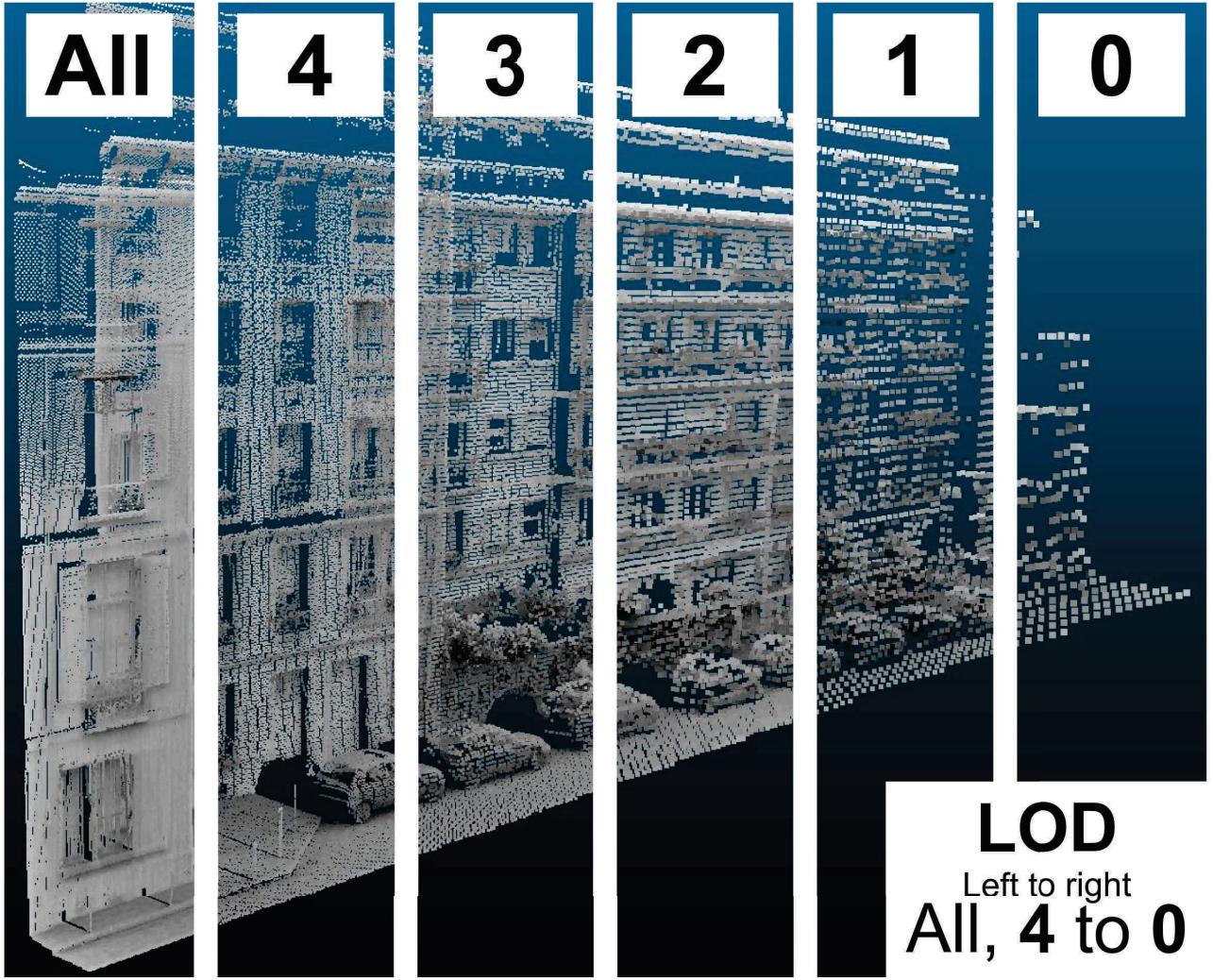


Figure 11: Schematic illustration of different LOD. Left to right, all points, then LOD 4 to 0. Visualized in cloud compare with ambient occlusion. Point size varies for better visual result.

4.5.3 Size versus LOD trade-off We compute the size and canonical transfer time associated for a representative street point cloud. For this order of magnitude, the size is estimated for 5*4 Byte (5 floats) per point, and the internet transfer rate is estimated at 1 Mbyte/s.

Table 1: Number of points per LOD, plus estimated transfer time with modern internet connection.

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

4.5.4 Large scale computing

MidOc implementation We use 3 implementations of MidOc, two being pure plpgsql (postgreSQL script langage), and one Python.

Computing on very large dataset 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, which we consider at least 10 times too slow. We briefly analyse performances, and conclude that data IO limits the number of efficient workers to 10, and that most of the time is actually not spent on computing, but on getting the points and writing them back.

4.5.5 Fast abnormal density detection and correction

Density abnormality (peak) Important variation of density can be a serious issue for some processing methods, or simply performances (especially in parallel environment). The figure 12 shows a place where the density is 5 times over the normal value for 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.

Fast detection We analyse the data set to find places of abnormally high density. It can be performed extremely fast at the patch level. (0.1 s for Vosges dataset, same for a 2 Billion points urban dataset). The Top left image shows the number of points per m^3 directly read from database, increasing from blue to yellow to red) 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 illus-

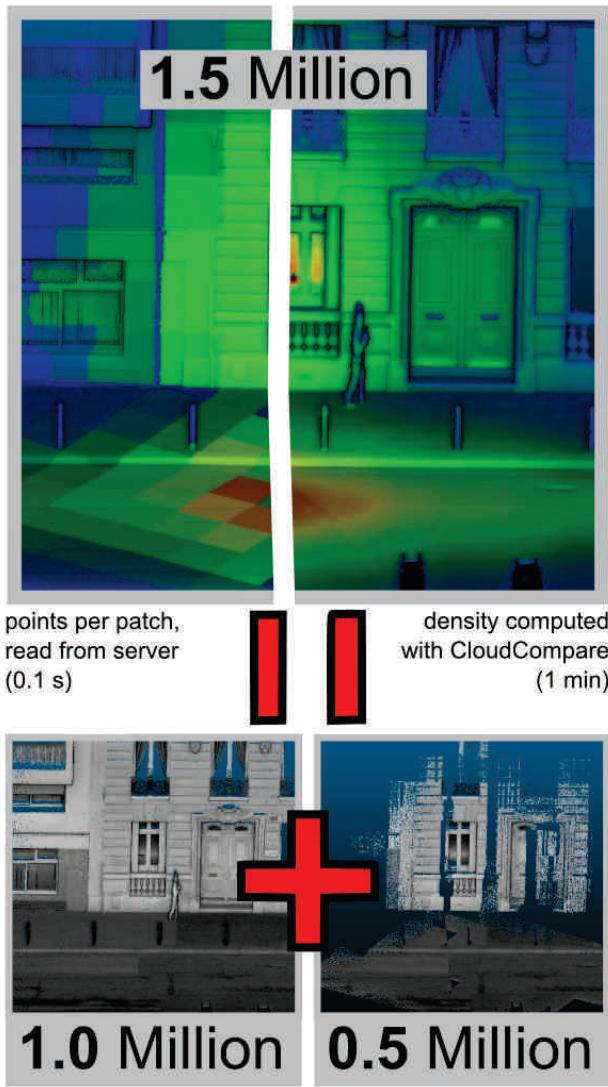


Figure 12: Abnormal density detection and correction. Top points per patch (left) or density (right), green-yellow-red. Bottom reflectance in grey.

tration), and after the density is computed for each points, all the point cloud still need to be filtered to find abnormal density spot.

Simple correction 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 / m^3 , bottom right, 5k to 24 k pts / m^3). It considerably reduces the number of points (-33%).

4.5.6 LOD stream As a proof of concept we stream points from the data base to a browserIGN (2014). For this experiment we only stream a given number of points per patch, which allows to accelerate greatly data loading.

4.6 Dimensionality descriptor and patch classification

4.6.1 Experiment summary For each patch, we store the associated number of points chosen per level (*ppl*) while computing MidOc ordering.

This 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). An analysis of confusion matrix shows which meta classes are separable or not.

Extra descriptors are then needed : (P : for Paris , V : for Vosges: - *points_per_level* (*ppl*), level 1 to 4 (P+V) - average of intensity (P+V) - average of *number_of_echo* (P+V) - average of height regarding laser origin(P) - average Z (V) - patch height (P) - area of *patch_bounding_box* (P) :

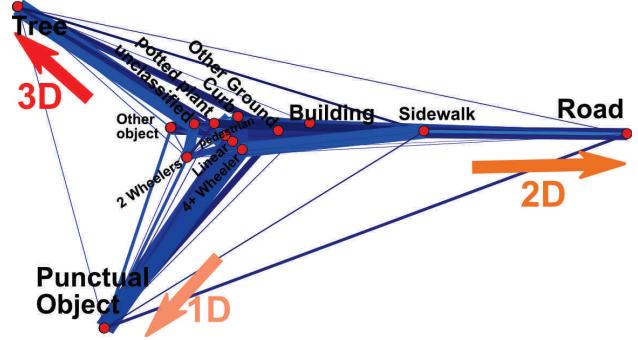


Figure 13: Spectral clustering of confusion matrix of Paris data set classification using only *ppl* descriptor. Edge width and colour are proportional to affinity. Node position is determined fully automatically. Red-ish arrows are manually placed to help understand

4.6.2 Separator power of *ppl* descriptor for Paris data set
Using only the *ppl* descriptor, a classification is performed on paris data set, then a confusion matrix is computed. A spectral clustering of this matrix interpreted as a distance matrix between classes is performed. This clustering is used to place classes on the illustration. We manually add 1D, 2D and 3D arrows.

4.6.3 Analyzing data set classes

Balancing the data set 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 lessens variability of results when classes are very heterogeneous.

Analysing class hierarchy Choosing which level of the class hierarchy uses 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 sets 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 analyse the confusion matrix by spectral clustering.

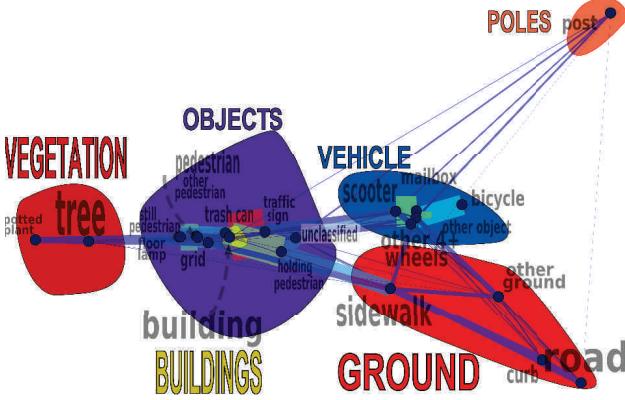


Figure 14: Result of automatic spectral clustering over confusion matrix for patch classification of Paris data set with all simple features. Edges width and colour are proportional to confusion. We manually draw clusters for easier understanding.

This analysis is useful to show 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).

4.6.4 Patch classifier

Vosges data set We perform a analysis of error on Vosges dataset and we remark that the error seems to be significantly correlated to distance of borders.

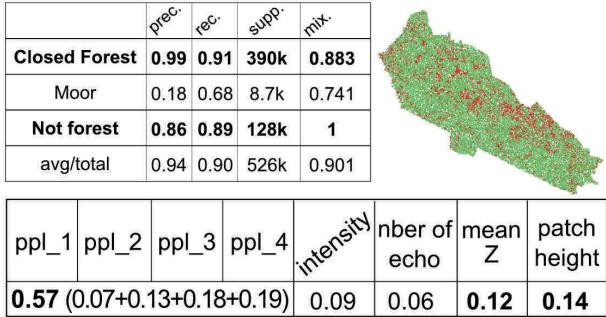


Figure 15: Feature usage (first table). Precision(prec.), recall (rec.), support (supp.), and average percent of points of the class in the patches, for comparison with point based method (mix.).

The learning time is few minutes (monoprocess, python), the predicting time is few seconds (same).

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

4.6.5 Patch classifier applications

Artificial increase of precision We demonstrate the use of artificial increase of precision. Initial results (blue) are mostly correct, but by only keeping patches with high confidence, it is possible to increase precision to 100%. Above 100%, we reduce the variability of the found building patches.

In this example the precision was already very good (most of the blue patches are in building), but increasing precision to reduce class heterogeneity provides a much better base for building reconstruction.

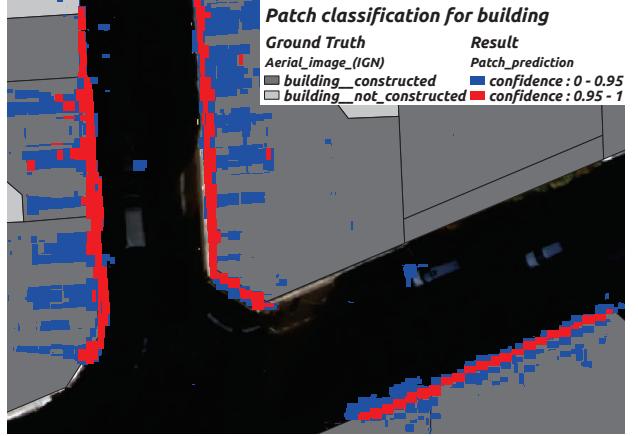


Figure 17: Plotting of patches classified as building, using confidence to increase precision. Ground truth from IGN and Open Data Paris

It is possible to use this method only when precision is a rising function of confidence given by random forest. This is the case for 4+wheeler class.

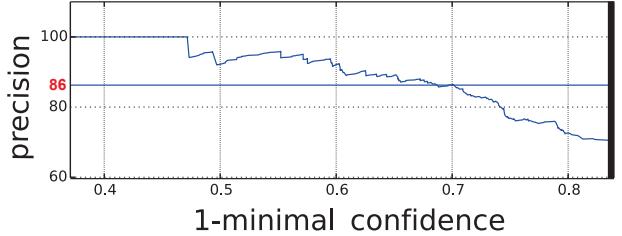


Figure 18: Precision of 4+wheeler class = $f(1\text{-random forest confidence score})$. The horizontal line is the average confidence.

Filtering: artificial increase of recall The patch classifier is used on paris data set. The goal is to find all ground patches very fast. We focus on an 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 ($X, Y : 2 \text{ m}$, $Z : 0.5 \text{ m}$) 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

In this section we discuss the results following the result section, and give limitations of the methods.

5.1 Point cloud server

The focus of this article is not the Point cloud server, thus we discuss only briefly it. The point cloud server has been proven to work easily with different type of point cloud, for many traditional application of point cloud. It is to the best of our knowledge the fastest and easiest way to filter very big point cloud using complex spatial and temporal criteria. It is also the only solution that is natively integrated with other GIS data (raster, vector).

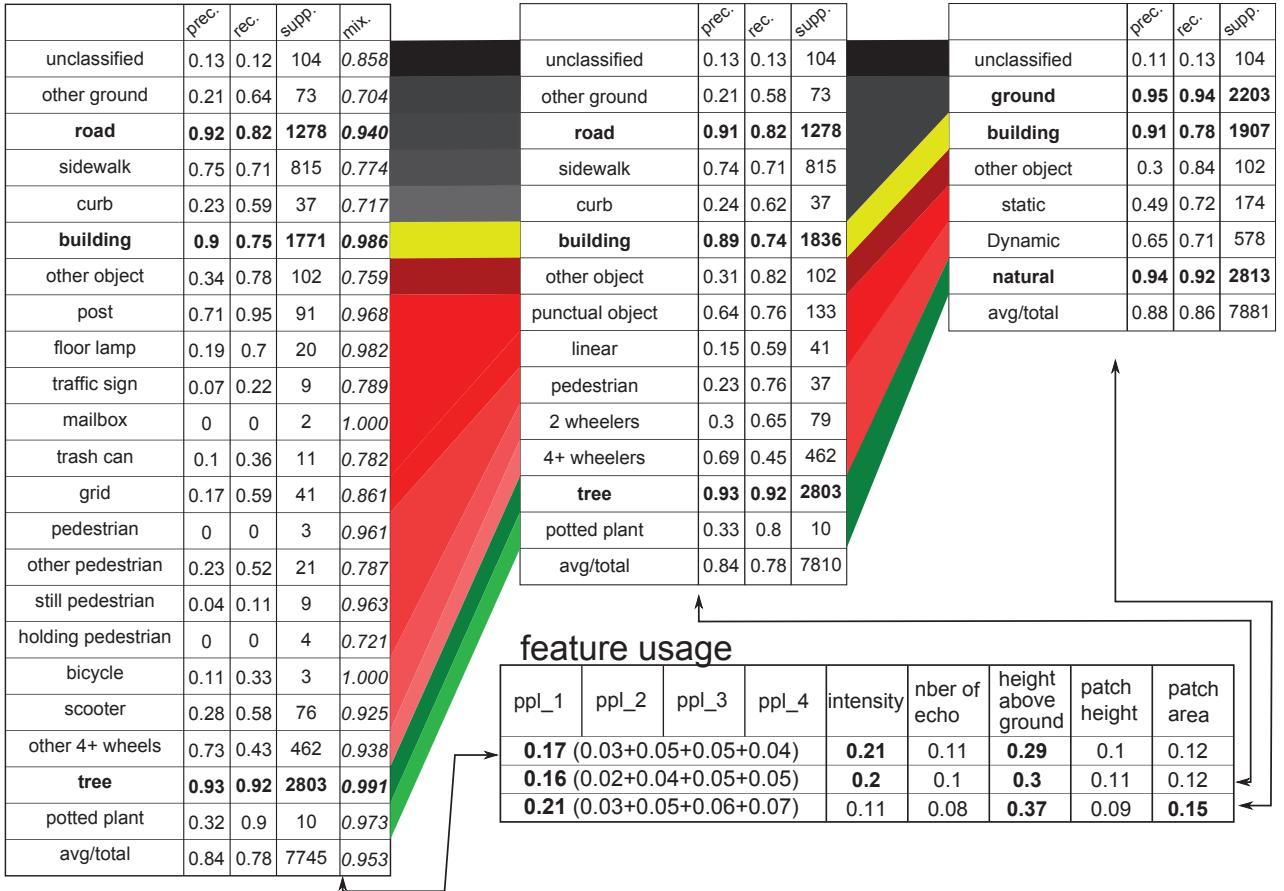


Figure 16: Results for Paris data set: at various level of class hierarchy. Precision(prec.), recall (rec.), support (sup.) and average percent of points of the class in the patches of the class, for comparison with point based method (mix.). Classes of the same type are in the same continuous tone. Feature usage is evaluated for each level in the class hierarchy.

Point cloud server limitation The limitations are that the multi-acquisition process is not defined, the lack of facilities to easily add attributes to an existing point cloud, and more importantly, the fact that accessing only few points in a patch still requires to read the whole patch (all points and all dimensions of this patch). We note that these limitations are not fundamental but simply require more work.

5.2 Data set

Vosges data set The figure 10 exposes important density variation, even for aerial lidar point cloud (green) that are supposed to be more homogeneous. The variation of density may be due to variation of plan speed, and more importantly, variation of distance between plan and ground (due to height of plan or ground). Some of this variation is also artificial and due to the way the acquisition is loaded into the point cloud server : the acquisition we use consist of thousand of las files.

Those files are separated into 50^3 m^3 patches individually, which means that some patches on the border may be less filled.

Paris data set Surprisingly the Paris data set density distribution seems to match a linear distribution (in the log /log space of the figure 10). For a short period of time, we assume building and ground form a U shape, the laser device being in the middle. The laser rotates on himself around the back-front of the vehicle axis, and acquire points at fixed angles (about 1/10 of degree). It creates points more and more spaced along the building height.

Limitation of the data set The data set used are representative of urban and aerial Lidar point cloud. However the methods are not tested with indoor point cloud, or Structure from Motion point cloud.

5.3 Exploiting the order of points

Limitation Schematically, the method proposes to exploit order to store LOD information. There is a strong and evident theoretical limitation : the order may already contain a useful information ! For instance with a static Lidar device the neighbourhood information can be reconstructed from the order. We note that for the Paris data set the point are initially ordered by time of acquisition. Because the time of acquisition is also an attribute, this ordering can be reconstructed if necessary.

5.4 MidOc : an ordering for gradual geometrical approximation

5.4.1 Visual evaluation 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 8 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.

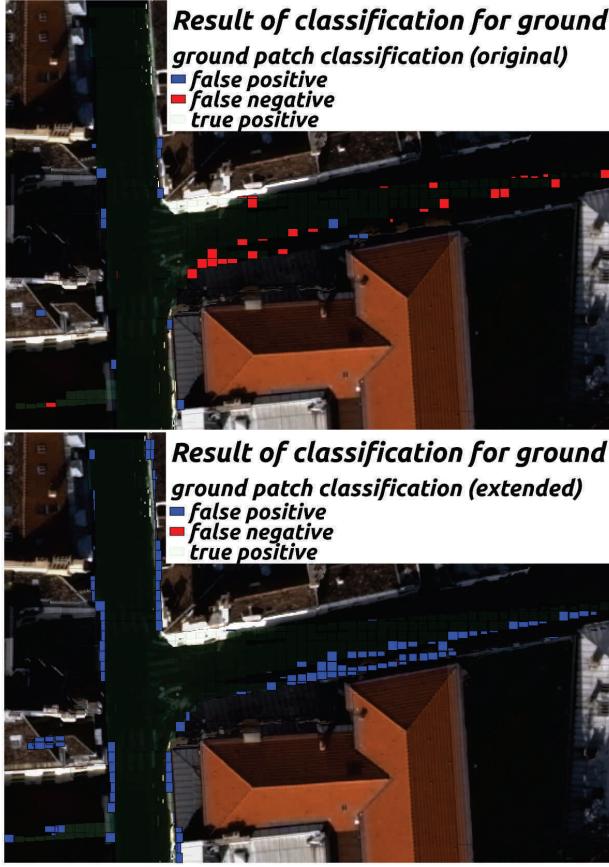


Figure 19: Map of patch clustering result for ground. The classical result finds few extra patches that are not ground (blue), and misses some ground patches (red). Recall is increased by adding to the ground class all patches that are less than 2 meters in X,Y and 0.5 meter in Z around the found patches. Extra patches are much more numerous, but all the ground patches are found.

5.4.2 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 throughput, but not fast enough for a LAN 10 Mbyte /s. This relatively slow throughput is due to current point cloud server limitation (cf 5.1).

5.4.3 Large scale computing The relatively slow computing (100 Millions points /s) is a very strong limitation. About 2/3 of the computing time is spend on data transformation because algorithm works and output a list of points, which must then be disaggregated/aggregated from/to a patch. This could be avoided by a C implementation which can access raw patch, and would also be faster for ordering points.

5.4.4 Fast abnormal density detection and correction

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 easy.

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. It also introduces the classical sampling limits : only density abnormalities spatially bigger than 2 patches (at least 2 m wide for Paris) can be detected.

Simple correction The correction of density peak we propose has the advantage of being instantaneous and not induce 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 /m³ 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. For instance if an application requires 1000 points /m³ 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't be separated by more than 30 seconds, and all points in a patch must come from the same acquisition file).

5.4.5 LOD stream Streaming low level of detail patches greatly accelerate visualisation, which is very useful when the full point cloud is not needed. However all patches should not have the same LOD, but the required LOD should vary for each patch based on distance to camera.

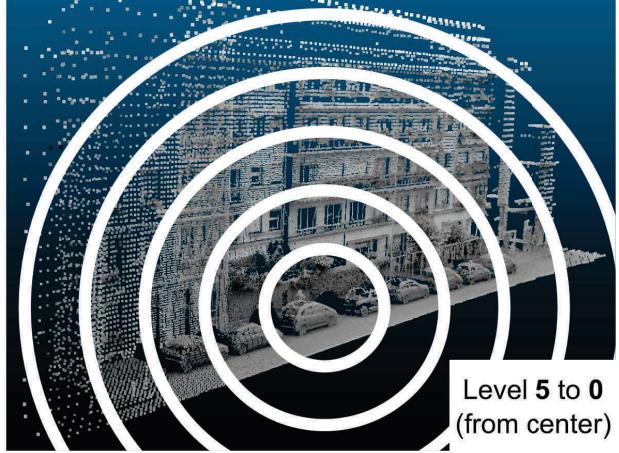


Figure 20: Schematic example of LOD depending on distance to camera

As seen before (5.4.2), 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.

5.5 Dimensionality descriptor and patch classification

5.5.1 Ppl as a crude dimensionality descriptor Figure 13 is obtained fully automatically (excepting the red arrows), using only the *ppl* descriptor. Yet objects of pure dimensions 1D,2D and 3D are neatly separated, with mixed dimensionality classes being in the center, which support the role of *ppl* as a crude dimensionality descriptor. Further evidence can be observed on the 2D axes. Sidewalk are not pure 2D because patches may contains parts of other objects. Building is also mostly 2D but balcony, building decoration and floors introduce lot's of object-like patches, which explain that building is closer to the object cluster.

Limitations As expected, *ppl* descriptor are not sufficient to correctly separate complex objects, which is the main limitation. There is a more fundamental limitation of *ppl* as crude dimensionality descriptor. Because *ppl* are per patch and not per point, dimensionality can only be estimated for objects that are bigger than a patch (1 m³ for paris dataset). And for this estimation to be possible, a patch must also contains a sufficient number of

points, which is not completely the case for distant tree patches for example.

5.5.2 Analyzing data set classes

Balancing the data sets Strategies to balance data sets have a big impact on result. We observe that some classes have simply not enough support and too much diversity to be correctly predicted. This is the limit of our approach where all classes are learned and predicted at once. For classes with very small support the one against all strategy should be tested.

Analysing class hierarchy The figure 14 shows automatic class clustering using the confusion matrix between classes with all simple features. Interestingly, the layout preserve the separation between pure 1D, 2D, 3D, and mixed dimensions. The distinct meta classes that will be possible to separate correctly are vegetation (3D), ground (2D), poles (1D), objects, vehicle and buildings. This illustration also 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.

5.5.3 Patch classifier

Vosges data set 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. 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 from 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 be more performant or faster.

Paris data set The figure 16 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 are 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).

5.5.4 Patch classifier Applications Because the propose methods are preprocess of filtering step, it can be advantageous to increase precision or recall.

Artificial increase of precision In the figure 17 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 18. 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 strategy is not possible for incoherent class, like unclassified class.

Filtering: artificial increase of recall 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 this 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

We propose to store LOD in the order of points. For visualization application, the best ordering may not be geometrical (like MidOC), but may depends on visual saliency of the points. In this case the LOD of each patch should be adapted to the distance to camera.

Theoretically *ppl* descriptor could be decomposed on a pure dimensionality base. For all applications, it could be interesting to find methods to group patches (and merge *ppl*) to enable dimensionality analysis at several scales.

Because some patches contain mixed classes, a bias is introduced which mechanically lower the average per points results. The random forest classifier should be in fact a random forest regressor, predicting for each patch not a class, but classes distribution.

References

- Amit, Y. and Geman, D., 1997. Shape quantization and recognition with randomized trees. *Neural Comput.* 9, pp. 1545–1588. [2](#)
- Breiman, L., 2001. Random Forests. In: *Machine Learning*, pp. 5–32. [2](#)
- Cura, R., 2014. A PostgreSQL Server for Point Cloud Storage and Processing. [8](#)

- Demantké, J., 2014. Reconstruction of photorealistic 3D models of facades from terrestrial images and laser data. PhD thesis, Paris Est, Paris. [2](#) [7](#)
- Elseberg, J., Borrman, 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., Xu, H. and Yan, S., 2013. Online robust pca via stochastic optimization. In: Advances in Neural Information Processing Systems, pp. 404–412. [5](#)
- Girardeau-Montaut, D., 2014. CloudCompare. [4](#)
- 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. [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](#)
- IGN, 2014. iTowns. [10](#)
- IQmulus, 2014. IQmulus & TerraMobilita Contest. [8](#)
- Meagher, D., 1982. Geometric modeling using octree encoding. Comput. Graph. Image Process. 19(2), pp. 129–147. [2](#)
- Networkx, d. t., 2014. Networkx. [8](#)
- pgPointCloud, R., 2014. pgPointCloud. [2](#) [8](#)
- PostGIS, d. t., 2014. PostGIS. [8](#)
- PostgreSQL, d. t., 2014. PostgreSQL. [8](#)
- Schnabel, R. and Klein, R., 2006. Octree-based Point-Cloud Compression. In: SPBG, pp. 111–120. [2](#)
- 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](#)
- 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. [3](#)
- van Oosterom, P., 2014. Point cloud data management. [2](#)