MIDOC ORDERING FOR EFFICIENT AND SIMPLE LEVEL OF DETAILS ON LIDAR POINT CLOUD

Rémi Cura, etc.

Table des matières

1. Introduction

1.1. Problem

Point cloud data is becoming more and more common. Following the same trend, the acquisition speed (points/sec) of the sensor is also increasing. Thus Lidar processing is clocking on the big data door. The precision of the sensors is also increasing, leading to more and more detailed point cloud.

Yet the usage of point cloud data is also spreading and going out of the traditional communities that used it. Lidar are commonly used on robots, intelligent cars, architecture, sometimes by non-specialized users.

For all this usage, having the raw, complete point cloud is often unnecessary, or even damageable. Thus we are facing a simpler version of a problem tat the vector community has faced for a long time : how to generalize point cloud, with data sets that are several order of magnitude bigger than usual vector data set ?

This kind of problem is very common in data processing. Having a big data set, how to reduce its size while preserving its characteristics. It is the base problem of compression for instance.

The problem is also made complex by the mix of point cloud with varying density.For instance an aerial lidar map augmented at certain places by terrestrial scanners, or vehicle-based lidar acquisition, where the density varies with speed and scene geometry.

Here we deal with a simplified version : given a point cloud, how to efficiently generate Level Of Detail of this point cloud while preserving the geometric characteristic, without duplicating data. The key to Level Of Detail approach is efficiency. We accept to loose a part of the information in exchange of a massive reduction of data size. A solution using LOD must then by nature be efficient.

1.2. Motivation

- Pointcloud : becoming common (why) Point cloud are becoming common because sensor are smaller, cheaper, easier to use. Point cloud from image (using Stereo Vision) are also easy to get with several mature structure from motion solutions. Point cloud complements very well images, LIDAR pointcloud allowing to avoid the ill-posed problem of stereovision, and providing key data to virtual reality.
- Growing data set + Multi sources At such the size of data set are growing, as well as the number of dataset and their diversity.
- Why is it important (size of the industry) The point cloud data are now well established in a number of industries, like construction, architecture, robotics, archeology, as well as all the traditionnal GIS fields (mapping, survey, cultural heritage)
- PointCLoud users = specialist in processing, not informatics/storing The LIDAR research community is very active. The focus of lidar researchers is much more on lidar processing and lidar data analysis, or the sensing device, than on methods to render the data size tractable.

1.3. state of the art

RC:1.3.0. when I have access to Zotero

State of the art should include

- what people do with poiint cloud ? (oosterom 2014)

1.4. what's missing in biblio

Point cloud generalization are far from the subtlety of vector generalization (// mettre des references). If we take the State of the art software CloudCompare (//mettre reference), we can choose between a random subsampling and a minimum distance subsampling.

More generally, proposed methods usually focus on data compression and computing acceleration.

Another common practice coming from Computer Graphics field is to compute an octree over the point cloud, then for each cell , compute a subsampled version of the pointcloud. This allows to have simple and efficient Level of Detail , at the price of data duplication, and high sensibility to density variation.

Morevover, all the method are specific and depend on a specific data structure that has to be stored extra to the point cloud data.Steaming from this the interoperability is non existant (moving from one software to another, one loose the data structure).

We have then the classicla and seemingly intractable problem : if we precompute a data structure, we have to store it. If we compute it on the fly, we may end up using lot's of computing time performing the same processing several time.

1.5. contribution

In this paper, we focus on simplicity, efficiency and re-use of existing and well established methods. All the methods are tested on Billion scale pointcloud, and are open source for reproductibility test and reuse. We propose a simple method that enable portable computation free geometrical level of detail. The first contribution is that we propose to store the LOD information directly into the ordering of points rather than compute new subsampled point cloud for each LOD. Thus, 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 so that we have a varying geometric approximation of the point cloud.

The third contribution is to use the ordering construction by-product as simple and free dimensionnality descriptors, with usuability demonstrated in a Radom forest classification experiment.

1.6. plan of the article

The rest of this article is organized as follow : In the next section we present the methods. In the section XX, we present some results and order of magnitude In the last section we discuss the results, the limitation and improvments of our solution.

2. Method

2.1. introduction

In this section we introduce a simple and new method which proposes free geometrical LOD features, at the price of few preprocessing time. This method has been designed to work with Lidar data set, and should be used on noisy SfM point clouds after filtering. Similarly, this method has been used inside our new point cloud data management system which is centred on a Point Cloud Server (the article is being written). At such, the example showcase a real urban scene where the point cloud has been cut into 1m3 points groups called patch. The method is then used on each patch. Lastly computing this LOD gives interesting by-product that can be used as crude classifiers, and allow to perform extremely fast rough-filtering of massive point cloud.

2.2. exploiting the order of points

2.2.1. principle

We propose a very simple yet efficient method. We start from a point cloud and we want to generate geometrical levels of details on it.

Point clouds are physically stored as a list of points. Now, the information contained by a list of point is not the sum of information for each point.A big difference is that the list is ordered.

This fact is common for every point cloud format. At one point, points must be written to disk, and they become ordered. In fact it would be quite hard to store the point in a totally unordered fashion (true random is expansive). The first contribution of this article is to propose to exploit this ordering to store information. The key idea is : we order the points of the point cloud so that when reading the points from beginning to end, we get gradually a better and better geometrical approximation of the point cloud. We call this ordering "MidOc" ordering, but the idea to use order to store LOD works with any ordering.

2.2.2. conceptual example

So for instance if we have a 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. Reading another 16 (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.

2.2.3. advantages

no on-line processing time. The advantages of this method is that except a pre-processing step to write the point cloud following the MidOc ordering, each time we want 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 time at several level and by several users simultaneously, no processing makes a big difference. (See result section for an example with LOD visualisation.)

no data duplication. Another big advantage of using the order of points to store the LOD information rather than using external structure data is that it is free storage wise. This is an advantage on low level. First we need less space on disk. Second we are sure that all the information is at the same place, which avoids to perform OS level commands like going trough directories, opening the structure file, etc. Having no data duplication is also a security in concurrent use. If somebody changes the point cloud, another user could use the old structure with the new points in the time where the new structure is computed. On the opposite, when the ordering gives the LOD information, a user either get the old points with the old ordering, or the new points with the new ordering, which guarantee that data is in a coherent state. Lastly by ordering rather than creating a sub sampled version, we avoid all precision-related issues, because the order doesn't change existing points and their attributes, but only there order in the point cloud.

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 !), this way to create LOD is portable. Most software don't change the points order inside a cloud. If a tool changes the order, it is easy to add the ordering number as an attribute, which increase a little bit storage, 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.

2.3. MidOc : an ordering for gradual geometrical approximation

2.3.1. principle

The key of LOD is efficiency, because using LOD is already a trade-off between data information loss and data size reduction. Thus we need an efficient method that order the points so that reading from the start toward the end is going to give a gradually better geometrical approximation of the point cloud.

Precisely because we do a trade-off, we must exploit the intrinsic structure of the point cloud. For this, we make some assumption that are mostly verified on Lidar point cloud. - Point cloud represents mostly 3D surfaces sensed by a fix or mobile sensor (with the exception of multi-echo, which is correctly dealt with anyway). - geometrical noise (outliers) is low. - density may vary, but does not gives information (that is, depending on the sensing situation, we may have parts of the cloud that are more or less dense, but this has nothing to do with the nature of the object sensed, thus must not be preserved). - density of the LOD should be has constant has possible.

For simplicity and speed we propose MidOc ordering, which is a re-use of well known existing methods. We name this ordering for clarity of this article, nonetheless we don't think we are the first to use it. Again the principle is very simple, and is to take for each level the points closest to centre of its octree voxel, if any. Given a point cloud of N points and a tree depth of T, We compute an octree of depth T over the bounding box of the point cloud.

For each cell, we take the point closest to the centre of the cell if any.

We re-unite the points chosen, avoiding duplicates by giving priorities to lowest (= bigger cell) level. Then we put together all the points, putting the chosen points first, ordered by lowest level, and an inverse Z curve or random order. The inverse Z curve is a Morton curve read backward bit wise and can be used if determinism is wanted, or more simply random order if determinism is not a constraint. We consider T has the maximum depth, thus we modify the algorithm to stop when a given level is not sufficiently filled to save computing.

This method can have the same complexity as octree construction, because the only supplementary cost are to find the point closest to the middle , which can be done will constructing for free, and uniting the point at the end, which doesn't really need a sorting because we can use walk trough the octree width-first for this.

Even a very crude approach, where we construct the octree, then use it to find the points closest to middle of cells, will be faster than O(n*ln(n)).

2.3.2. efficiency and performance

In fact we don't really need to construct the octree because octree can be obtained by reading coordinates bitwise in a correctly centred/scaled point cloud. For a given point cloud, if we centre it so that the lowest point is all dimension is (0,0,0), and scale it so the biggest dimension is in [0,1]. Then we quantize this point cloud into [0,2**L-1]. Coordinates are now integer, and for each points, reading the coordinates bitwise left to right gives the position of the point in octree for level 1-L. This means performing this centring/scaling/quantization directly gives the octree, and that further operation can be performed using bit arithmetic, thus be extremely fast.

We also note that it is possible to propose a streamed implementation, thus enabling on the fly computing of it. We propose a python recursive implementation as a proof of concept.

example. For instance for a 2D point cloud with L = 3, and two points P1(4,3) and P2(5,2) already quantized in [0,7]. Their binary form is P1(100,011) , P2(101,010). Reading the first bit (left-right order) of the points gives the coordinate of the cell in the level 1 of the octree. for P1 and P2 it is (1,0), which means the bottom right cell of the quad-tree The second bit gives the position inside the second level of the octree. It is (0,1), which means the upper left sub-cell of the previous pixel. The last bit is (0,1) for P1 and (1,0) for P2, which allows to tell they won't be in the same cell. @

2.3.3. conceptual example

give example with very high density in the middle (vehicle stopped), plus object, plus tree.

2.3.4. advantages

Well known, Simple and efficient. This method is extremely classical and based on Octree This make it simple to implement, and possibly extremely memory and CPU efficient.

The complexity is O(n*L) or less . Octree construction can even be performed on GPU for extreme speed up. It is a matter of days to implement it.

fixed density. This method guarantee a constant density for given levels, even when the acquisition process produced varying-density point cloud. Thus using the output of this method is a safeguard for most complex point cloud processing method that may be badly affected by singularity in density.

permissive. This method is permissive because we make very few hypothesis 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, we can leverage the information given by this ordering as a good geometrical descriptor of the point cloud. We can keep the number of points per level for each level, which can be a good descriptor of the geometrical nature of the object contained in the point cloud.

2.4. using the ordering by-product as a crude dimensionality descriptor

2.4.1. principle

During the ordering of a point cloud, we can keep the number of chosen point for each level. Now 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 extremely simple descriptor with a classical Random Forest classifier on a real world dataset publicly available. @

RC:2.4.1. cite DemantkÃ© thesis because he creates sophisticated dimensionality indicators

We don't try to provide a per point classification, but a classification per group of points (i.e patch) . The idea is that for many points we may only use this simple classifier. So we could speed up a very complex classifier by first using the simple one, then use the complex one on parts that the simple classifier is not confident with. Another application is filtering. If the classification process is very fast, one can easily eliminate large parts of the point cloud when looking for a given type of points (for instance, looking for ground points, or tree points).

2.4.2. conceptual example

@For instance, we take 3 typical street extracts : a pole, a piece of ground, and a piece of tree. Due to geometry of acquisition and sampling, the pole is almost a 3D line, thus the points will be concentrated in very few octree cells. A typical number of points chosen per level would be (1,2,4,8), thus average a 2**L function. A piece of ground is often extremely flat and very similar to a plan (in fact, more like a very large radius cylinder part). Thus the typical points chosen per level would 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 fact that leaf are about the same size as point spacing and are partly transparent for laser (leading to several echo). So a foliage patch would typically be (1,8,64,512) (if enough points), so a 8**L function.

2.4.3. method

Again we don't work at the point level, but at the patch level (max 1m3). We order all patches following MidOc ordering, and for each patch ordered, we associate the number of points per level that where chosen. We train a random forest classifier using the number of chosen points per level (Vmidoc=[N1,n2,n3,n4...]). We use the number of points per level 1 to 4 included. For each level, we normalize number of points by the maximum number of points possible (8**i), so every feature is in [0,1].

We use the IQmulus benchmark, where each point has a class. Our first goal is to determinate which classes are susceptible to be successfully classified using only this descriptor. Our first strategy is then to learn/predict on all our classes using 10-fold strategy, then construct an affinity matrix out of random forest score. This allow us to cluster the classes into groups than can or cannot be effectively separated using only this descriptor.

We randomly chose patches on which to work, correcting the uniform distribution to have almost equal number of patches reprensenting each class.

We then train / predict with the classifier using a 10 fold strategy to have significant result.

2.4.4. advantages

simple. Dimensionality feature for point clouds are already well researched, and can be more precisely computed (**?**), with less sensibility to outliers. 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, x**2, x**3 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**n 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).

3. Result

3.1. 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 out of the box most used open sources software Second we perform MidOc ordering on very large point cloud and analyse the efficiency and quality of the results. Third we use the number of points chosen in the MidOc ordering has a descriptors for a random forest classifier on two large data set. We analyse the potential of this free descriptors, and what it brings when used in conjunction to other simple descriptors.

3.2. Point Cloud server introduction

All the experiments are performed using a Point Cloud Server (article is being written, but a very detailled presentation is accessible

RC:3.2.0. put reference to postgres presentation

). 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. This organisation is based on the observation that in typical point cloud processing workflow, we never need a point alone, but more often a points and most likely its surrounding points.

Having such a meta-type allows use to find points based on various criteria extremely fast. (order of magnitude : ms). Cutting a point cloud into patches provides also a very easy parallel processing possibility, which we use in our experiments.

For our experiments we cut terrestrial Lidar point cloud into 1m3 cubes oriented on (North ,Est,Z) axis. We cut aerial lidar point cloud into

RC:3.2.0. put the cell size for Vosges dataset

m3. 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.

3.3. Data Set used

We use two data sets. First

RC:3.3.0. cite IGQmulus data set

, an open source urban data set with varying density, singularities, and very challenging point cloud geometry. Data set is about 600Millions points, over 12 kms of road. Points are typically spaced by 5cm to 0.2 cm. It is a mutli echo laser (Riegl). We have access to a training set where every point is labeled. Then

RC:3.3.0. cite IGQmulus data set

, which is a very wide spread data set of 6 billions points. density is almost constant to . We have access to a rough ground truth about surface occupation nature (type of forest).

insert avg density for Vosges

3.4. Exploiting the order of points

3.4.1. 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, to whoch we add MidOc ordering as rgb. We export it as a ".txt" text file, this is 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.

3.4.2. results

All softwares passes test. Any software changing order of point-cloud can still be used if the order is exported as an attribute.

3.5. MidOc : an ordering for gradual geometrical approximation

3.5.1. experiment summary

In this experiment we first test ordering on typical street objects,then on small subset of terrestrial and aerial Lidar to appreciate the fitness to use it for geometrical LOD. Then we compute MidOc for two big datasets and try to qualify result visually as well as identify computing bottleneck . Lastly we briefly test exploiting LOD to stream 3D point cloud with various resolution to a browser.

3.5.2. visual evaluation

Visual evaluation on typical objects (ground, faÃ§ade, car, pole, vegetation). See fig XX

RC:3.5.2. put correct renv

RC:3.5.2. Visual results on aerial (Vosges).

visual evaluation on a portion of street. See fig XX

RC:3.5.2. put correct renv

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 Internet(s)	estimated time LAN(s)
All	0.2 to 5	1600	100	60	3
0	100	3	0.2	0.1	0.005
1	50	11.6	0.7	0.4	0.02
2	25	41	2.6	1.5	0.075
3	12	134	8.4	5	0.25
4	6	372	23	14	0.7

3.5.3. large scale computing

We use 3 implementations of MidOc, two being pure plpgsql (postgres script langage), and one python. Computing MidOc on very large point-cloud (parallel)

RC:3.5.3. compute on Vosges, 12 x parallel processing

Sucess : even on "singularity" : big accumulation of points Analysis -> Bottleneck is not computing , but read/update with new order.

3.5.4. LOD stream

? ?(parler du stream de patch dans itowns ?) ? ? limitation = have to put the whole patch in memory, even when getting only few points. Conclusion : as is : interesting when bandwidth limited.

3.6. using the ordering by-product as a crude dimensionality descriptor

3.6.1. experiment summary

When computing MidOc ordering, we can easily store the number of points chosen per level. This experiment tries to evaluate the utility of such crude dimensionality descriptor using random forest as classifier. This dimensionality descriptor 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 dimensionnaly very similar, yet semantically very different). We evaluate first what kind of reliable prediction can be made with the dimensionality feature, then we add several very simple feature to provide a result to compare to.

3.6.2.

4. Illustrations

4.1. MidOc ordering

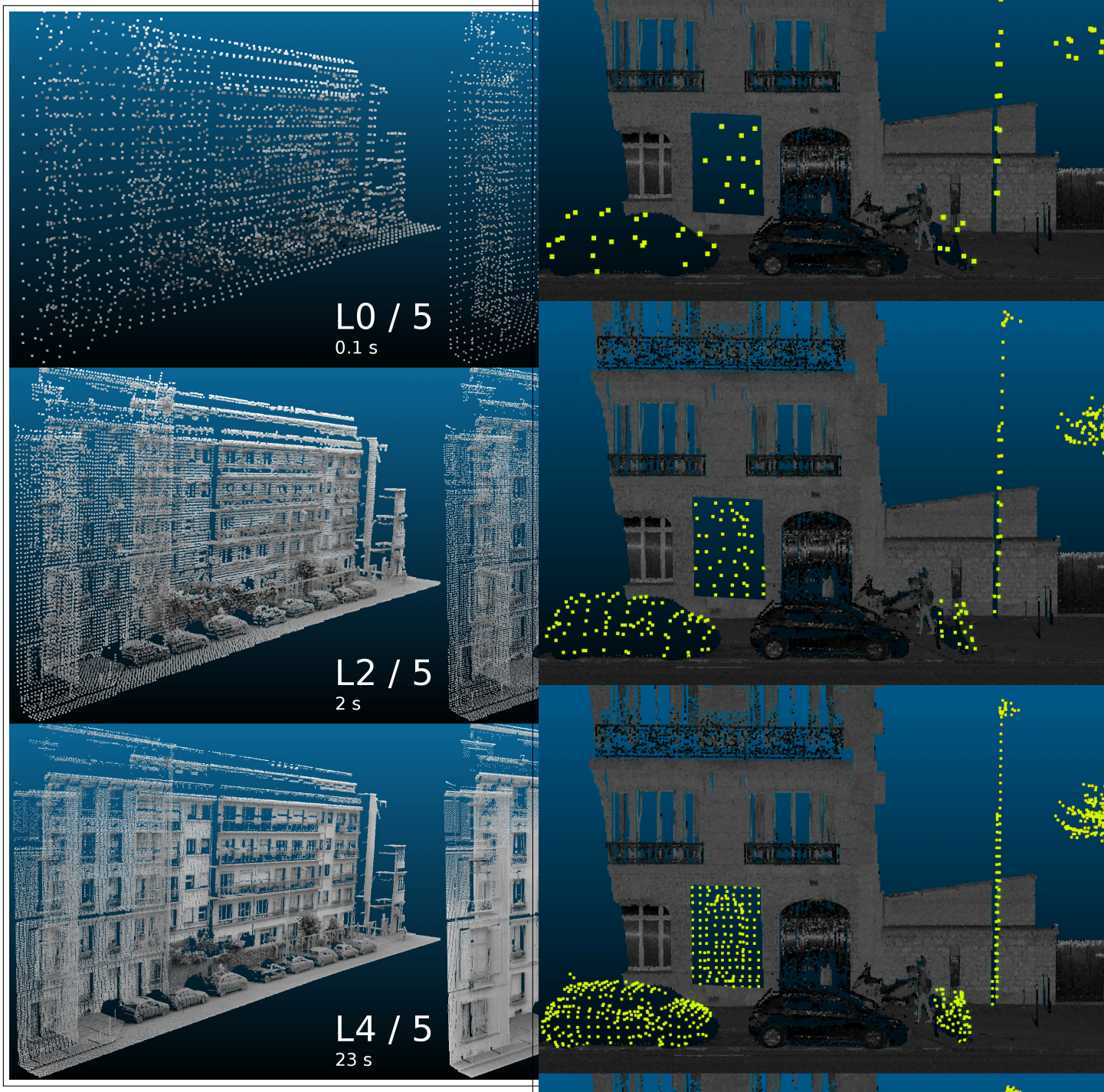

FIGURE 1: All successive levels, lighting by Ambiant Occlusion for visual comfort

4.2. Crude dimensionality descriptor