# CHANGE DETECTION ON POINTS CLOUD DATA ACQUIRED WITH A GROUND LASER SCANNER

D. Girardeau-Montaut[a,b], Michel Roux[a], Raphaël Marc[b], Guillaume Thibault[b]

[a] Telecom Paris (Ecole Nationale Supérieure des Télécommunications, Paris) (daniel.girardeau-montaut, michel.roux)@enst.fr
[b] EDF R&D (Electricité de France – Direction des Etudes et Recherches, Clamart) (raphael.marc,guillaume.thibault)@edf.fr

**Commission III WG III/3**

**KEY WORDS**: Ground LiDAR, laser scanning, points cloud comparison, change detection, monitoring.

**ABSTRACT:**

Ground based laser scanning is now well settled in the fields of cultural heritage, as-built modelling and monitoring applications. We intend to use laser scanning to detect changes on building sites or inside facilities, mainly for monitoring purposes but also to be able to provide synthetic information in case of an emergency. In both cases, the main constraint is time. A minimum precision on the measure of movements is also required, depending on the type of application. Laser scanner seems to be the perfect tool for such applications as it quickly acquires a large amount of accurate 3D data. But the comparison of so huge datasets implies the use of appropriate structures and ad-hoc algorithms. A specific octree structure is described, and then several simple cloud-to-cloud comparison techniques are presented. The best one, based on the Hausdorff distance computation is improved on various points. Also, as a full automatic process seems still unachievable, a software framework has been developed. It intends to minimize human intervention and therefore prevents from wasting the LiDAR speed in time-consuming post-processing operations.

## 1. INTRODUCTION

Change detection on 3D data is a rather new technique for non-professional users, considering that it has been done exclusively by professional surveyors by now. The main reason is that before laser scanning, the techniques to acquire 3D points on real scenes were topography and photogrammetry, which are very precise but time consuming (especially if one is interested in tiny details). It can hardly be done by non-professionals or in an automatic way. Moreover it is actually very inconvenient to apply such techniques in dense industrial context, with pipes and cables everywhere. Today, aerial or ground based laser scanners acquire much more information in a global but unstructured way.

A common technique to perform change detection on a point cloud is to compute its distance to a 3D reference model. It can be done either directly or by creating an intermediary model on top of the points. The reference model can either be theoretical or also created from real data. Point-to-mesh and mesh-to-mesh distances have been very well studied and demonstrated by academic software such as Metro (P. Cignoni et al. 1998) or Mesh (N. Aspert et al. 2002). Both need time and advanced treatments.

In our process, we could use automatic 3D shape reconstruction on a point cloud. It is a challenging problem that is widely and intensively studied. However, this technique still faces big issues that prevent us from using it: important computation time, especially on true 3D clouds with millions of points, and a very bad behaviour on complex or dense scenes with incomplete coverage.

In the particular case of aerial laser scanning, the sensor displacement implies that the acquired dataset is 2D½. Therefore it can be rather easily modelled with polygons or transformed into a depth image. This has led to the development of specific comparison techniques, mainly for urban areas (Murakami et al. 1998, Vögtle et al. 2003, Vosselman et al. 2004). The main application is database updating.

Finally, we can refer to the work of Mémoli and Sapiro (Mémoli et al. 2004) who have deeply analyzed the direct point clouds comparison problem in a very formal way. However, the point clouds have to be once again uniformly sampled with a good coverage, they must represent unique objects (manifolds) and the comparison result is global.

It seems that no specific method has been developed for direct cloud-to-cloud comparison of ground-based LiDAR data. Approaching techniques in the field of as-built verification or building site monitoring consist either in binary comparison of point clouds for material appearance/disappearance mapping (Shih et al. 2004) or the comparison of a laser scanned point cloud with an existing as-built 3D model in a dense industrial environment (Gonçalves et al. 2003).

In this paper we present a method for the direct comparison of two or more point clouds acquired with a ground-based laser scanner in order to detect local changes under important time constraints. Section 2 deals with data acquisition and registration issues. In section 3 we present naive comparison methods which only need a light structure computation (octree). In section 4 we discuss their results and carry out several improvements, providing some assumptions made in section 2 are respected. Conclusions are presented in section 5.

## 2. DATA ACQUISITION AND REFERENCING

The first test consists of a building site monitoring application. The dataset has been acquired on a common downtown building site, during the excavation phase (at the very beginning of the building process). We used a TRIMBLE GS200 ground-based laser scanner. Point clouds were acquired every day, from almost he same position and orientation each time (mainly for stability reasons). There are 4 scans, about 200.000 points each (See Figure 1).

The second test consists of a change detection application inside a power plant (a highly obstructed environment). After ten years of intense scanning of its facilities, EDF (Electricity of France) owns a huge database of laser scanned point clouds. Some are acquired in the same area at different epochs; others correspond to different areas but to equivalent material. Most of them contain millions of points.

For both tests, knowledge of the laser scanner position and orientation is assumed. If point clouds are composed of multiple registered scans, the scan membership information for every point is also required for the improvement step that is proposed in section 4. There is no limitation on the number of scans or the total amount of points.

The registration process is done by standard techniques (physical targets, point matching, etc.). Of course, registration error directly interferes with the change detection process and may generate false detections. We note that automatic registration procedures and especially those relying on the ICP algorithm (Besl et al. 1992) do not converge with too different point clouds: the standard deviation of the points corresponding to a change (which are equivalent to noise here) should be less than 10% of the object size. If it is not the case, one should keep only the points that have not – or nearly not – moved. As a first approximation, assuming both clouds are roughly registered, it is possible to set a threshold value and consider during the ICP process only points whose distance to their nearest neighbour in the other cloud is less than this value. If there are enough of them and if they are sufficiently well dispatched all over the scene, the iterative registration process will converge and give fairly good results.

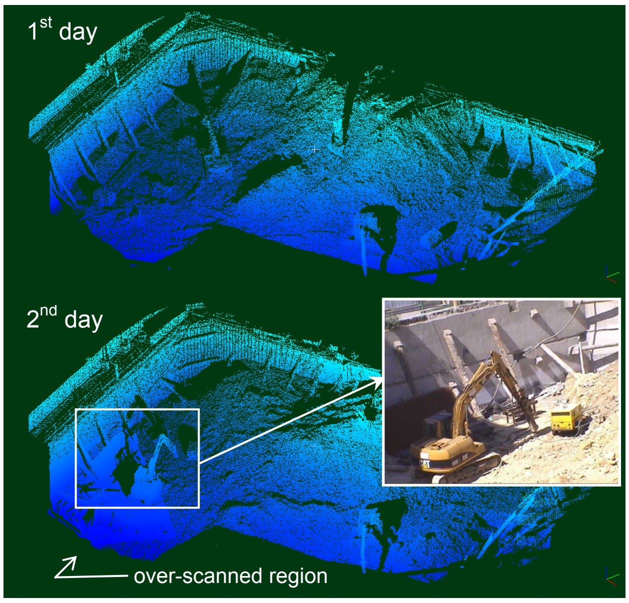

Figure 1 - Building site scans examples (two first days) coloured by height

## 3. DIRECT COMPARISON WITH OCTREE

Although we deal with large clouds (up to 30 millions of points), we wish to have an estimation of what has changed between two scans in a short or at least reasonable time. As indicated in the introduction, meshing algorithms applied to so huge and complex data are slow and hazardous. Moreover, the memory usage issue is critical, and meshes are not the best structure to face it. Thus we use a specific octree structure to ensure a good ratio between memory usage and points neighbourhood extraction speed.

### 3.1 The octree structure

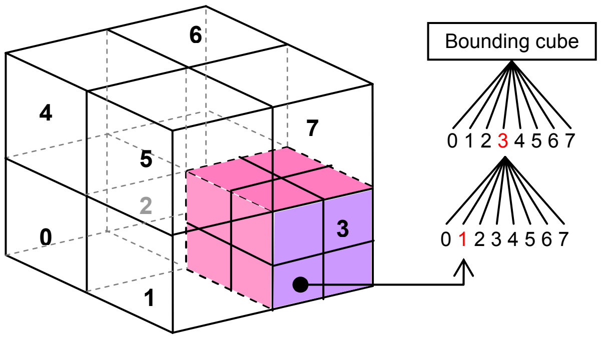

Figure 2 - Octree subdivision principle

An octree consists in a recursive and regular subdivision of the 3D space. In practical, the cubical bounding box of the set of points is divided into 8 equivalent cubes and this division process is recursively repeated for each cube. It stops when no more point lies in a cube or when a maximum preset level is reached. Such a structure gives the ability to rapidly determine which points lay in a specific cube and in its neighbouring cubes. Nearest neighbours extraction and equivalent processes become thus very fast.

An octree structure is generally (and logically) implemented with a tree structure but it is also possible to use a numerical coding scheme: for every point, a *3*n* bits long code is computed, where *n* is the maximum subdivision level of the octree. For a given level, 3 bits are used to code a number from 0 to 7. This number corresponds to the sub-cube (also called a cell) where the point lies. Each bit corresponds simply to the cell position relatively to its parent cube along one dimension. These numbers are concatenated for all successive levels to build the cell code (see Table 3). This coding step is very fast as the coding process is linear (it has a complexity of *k.O(N)* where *N* is the number of points and $k \approx n$).

| *Level* | *1* | *2* | *3* | | *n* |
|---|---|---|---|---|---|
| *Bits* | $b_z b_y b_x$ | $b_z b_y b_x$ | $b_z b_y b_x$ | ... | $b_z b_y b_x$ |
| | *most significant bit* | | | *least significant bit* | |

Table 3 - numerical coding of an octree cell position

Eventually, codes are sorted to make this octree structure more efficient. It makes possible to find rather quickly a given code with a simple dichotomic search, or all the points that lay in a cell at a given level *l* by considering only the first *3*l* bits of codes. The sorting step has a complexity of *O(n.log(n))*. This octree structure is simple to implement and offers a good flexibility (it is easy to add or suppress points).

We will close this discussion about implementation by referring to more sophisticated octree structures that would perfectly match our expectations - but for a highest implementation cost - such as Botsch et al. 2002 or Duguet and Drettakis 2004.

## 3.2 Octree-based comparison process

Assuming that clouds are expressed in the same reference frame, a mere idea is to apply the same octree subdivision process to all of them. Each octree structure is computed starting from the same initial cube which is the smallest cube that bounds all clouds. By this way, similar cells in all octrees are spatially equivalent. Therefore, one can expect that the subsets of points lying in these homologous cells are indeed "comparable" or at least, are part of the same object.

Based on this principle, a lot of "homologous cells comparison strategies" can be defined. They are generally as slow as they are accurate. Below are examples of such strategies. Let *S* and *S'* be two subsets of points contained in two homologous cells at a given subdivision level *l*. Let *N* and *N'* be respectively the sizes of *S* and *S'*. By default, *S* will be the compared cloud and *S'* the reference cloud (the comparison process is not symmetrical).

**Strategy 1** – "Average distance"

All points of *S* are labelled with the same value which is the average of the distances between each point of *S* and its nearest neighbour in *S'* (it could also be the minimum of maximum distance). This is can be quickly computed if *l* is big enough or equivalently if *N* and *N'* are rather small (see Table 8). It is robust to noise and density variations. But it is very inaccurate and should only be used in a preview step (see Figure 4).

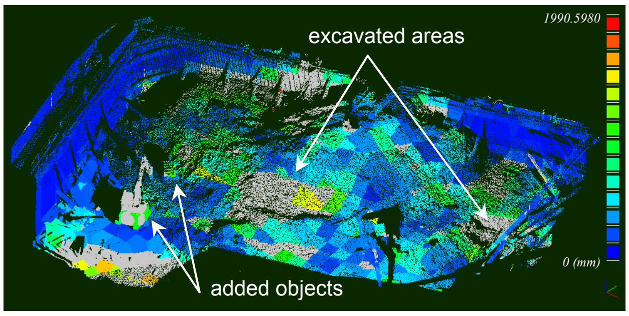

Figure 4 - "average distance" between the two first scans of the building site, computed at octree level 5 (points that have not been compared are displayed in grey).

As distance approximations are not exactly the same for different values of *l*, better results are achieved by applying the process recursively, keeping for each point the minimum value from a level to another (see Figure 5). However, some points are left not compared, as they are too far from the reference cloud (they lay in octree cells that have no homolog).

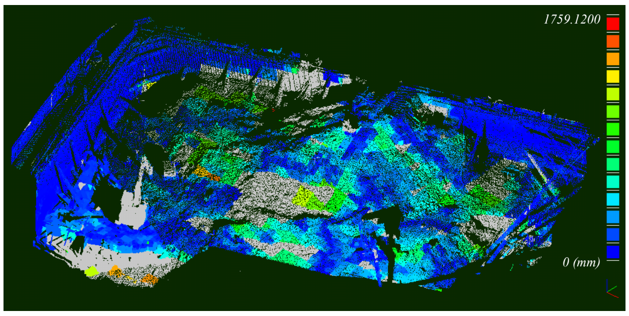

Figure 5 - "average distance" between the two first scans of the building site, recursively computed from octree level 5 to 10.

**Strategy 2** – "Best fitting plane orientation"

The least square plane is computed for both subsets. All points of *S* are labelled with the same value which is the angle between these planes. This is a simple tilt change approximation (and therefore a kind of shape modification information) but shifts are not detected. Once again, this is only a *quick and rough* evaluation of what has changed. We observe similar results compared to the first strategy (see Figure 6, to be compared with Figure 5). Once again, a recursive approach gives better results.

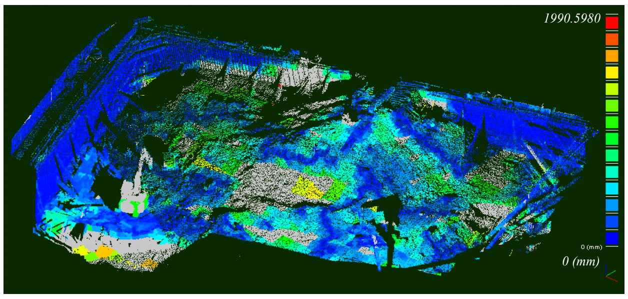

Figure 6 - "best fitting plane orientation" recursively computed from octree level 5 to 10

**Strategy 3** – "Hausdorff distance"

The Hausdorff distance between two sets of points is a common distance that consists in computing for each point *p* of a cloud *S* the distance to its nearest point in the other cloud *S'*:

$$d(p,S') = \min_{p' \in S'} \|p - p'\|_2 \qquad (1)$$

This is much more precise than the two previous methods but slightly slower. It is also very dependent on the point density variations between scans as it does not consider any implicit or explicit surface, but only points. The Hausdorff distance computation relies also on the octree structure but the nearest neighbour of a point does not lie necessarily in the homolog cell. For each point *p*, in order to assure that the truly nearest point has been founded, one must compute the distance to all points included in the neighbouring cells until the minimum distance is lower than the radius of the biggest sphere centred on *p* and fully included in this neighbourhood. There are no un-compared points by definition (see Figure 7). Note that the subdivision level at which the calculation is done does not change the result but only the computation timing (see Table ).

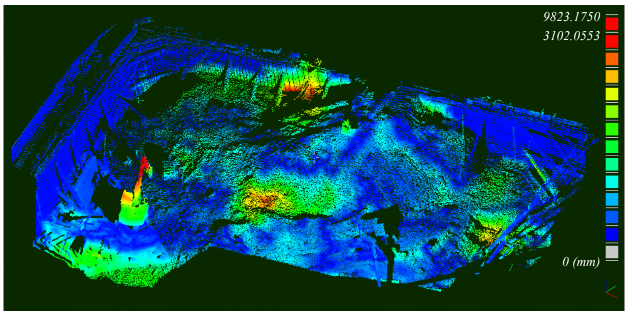

Figure 7 – "Hausdorff distance"

## 3.3 Computation costs and early results discussion

Here are presented indicative computation timings for the strategies presented above (see Table 8). These tests have been made on a notebook with a 1.7 GHz Centrino™ processor and 1 Go of memory. The first tested dataset is composed of the two first scans of the building site (170.000 and 765.000 points). The second dataset is composed of two clouds made of multiple consolidated scans and acquired inside a power plant (850.000 and 612.000 points). A third dataset, composed of two clouds of 12 million points each, is used for scalability test.

We observe that strategies relying only on homologous cells subsets comparison are of course faster than the true Hausdorff distance. However, their output is so approximate that they should only be used as preview processes. The Hausdorff distance is surely the best solution to our problem. It has however some important issues:

- As it has already been said, it is very sensitive to point sampling variations between scans.
- Its computation time depends greatly on the octree level (see Table 9) and the points repartition in space.
- A point will always have a nearest point in the other scan but it does not necessarily mean that their distance corresponds to any real change.

| Process | dataset 1* | dataset 2* |
|---|---|---|
| octrees computation | 01.25 s | 01.76 s |
| strategy 1 *average* | 02.21 s | 14.97 s (6) |
| strategy 2 *one pass* | 00.11 s (5) | 00.19 s (6) |
| strategy 2 *recursive* | 00.47 s (5-10) | 01.33 s (6-10) |
| strategy 3 *Hausdorff* | 07.07 s (7) | 21.38 s (8) |

Table 8 - computation costs comparison

*(*) numbers in braces are the octree levels used for computation that approximately correspond to the same number of points per cell*

| octree level | dataset 1 | dataset 2 | dataset 3 |
|---|---|---|---|
| 5 | 36.11 s | n.a. | n.a. |
| 6 | 12.88 s | 129.47 s | n.a. |
| 7 | 07.07 s | 042.78 s | n.a. |
| 8 | 24.03 s | 021.38 s | 563.59 s |
| 9 | n.a. | 116.32 s | 657.64 s |

Table 9 - octree level influence on Hausdorff distance computation time (best in yellow)

To address the first issue, we suggest to model locally the surface to avoid density variation issues. For each point of the first cloud, once the nearest point in the second cloud has been determined, a set of its nearest neighbours is extracted. With this set, a local surface model is computed (the least square plane, a Delaunay triangulation, a spline surface or a quadratic height function for example). Finally, the distance from the point of the first cloud to this surface patch is computed. By this way, any point sampling influence is avoided (see Figure 10). It can still remain error, depending on the chosen model, especially on highly curved or angular surfaces but it is much smaller. It should also be an optional treatment as it is not always useful while rather time consuming (with an appropriate algorithm and Delaunay triangulation, a multiplication of the process time by 2 to 4 has been observed).

To fix the second issue, we compute the average number of points per cell for every level (for the cost of one table scan – linear process – thanks to the octree structure described in section 3.1). The smallest level of subdivision that gives an average number of points per cell below a certain limit (which depends greatly on the computer capacities but which should not be greater than a few hundreds of points) is then chosen for computation.

Finally, the last issue is surely the most challenging. It does not result from the Hausdorff distance itself but more likely because of the laser scanning principle. Imagine a laser sensor at a fixed position, scanning a changing scene at different epochs. Because of the displacements of occluding materials, the laser beam will not go through the same portions of the 3D space (even if the field of view remains the same for each scan).

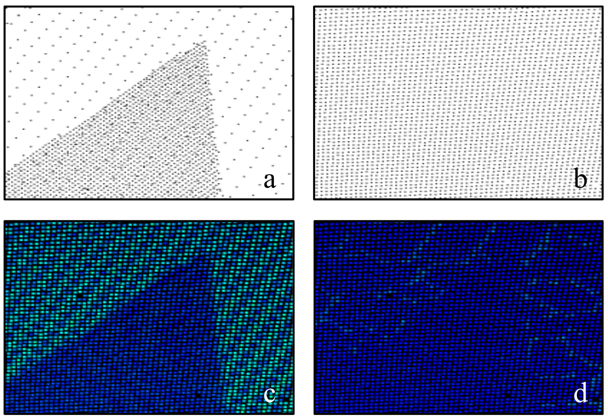

Figure 10 - Comparison of a set of non-uniformly sampled points (a) with uniformly sampled points (b). Points are sampled on the same plane. Results of the Hausdorff distance computed without (c) and with (d) local modelling (Delaunay triangulation).

Consider Figure 11: in case of disappearing material (top figure), the laser beam at epoch 2 goes through a portion of space that has never been scanned during the first epoch (the visible pink zone below the orange area). The fact is that it is impossible to have a clue about what may have changed in this shadow area. It is only possible to conclude that something has disappeared or moved in the first scan. However, a human - who have some *a priori* information on the scanned scene - would say that the two subsets of points on the left are indeed “comparable” while the points at the right have no homologous points in the first scan (bottom figure). A solution to this problem is proposed in the next section.

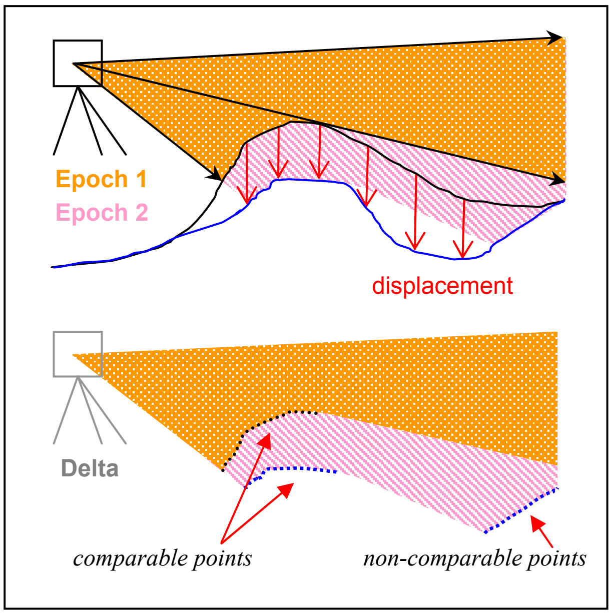

Figure 11 – scanner *shadow* areas

## 4. IMPROVEMENT WITH DEPTH MAPS

In the previous section, we have pointed out a consequence of the laser scanning principle that prevents us from using the Hausdorff distance as a "change detector" in some specific areas. To solve this problem, points that lie in those areas are filtered out in a first step and then specific treatments (chosen by a human operator, thanks to *a priori* knowledge of the scanned scene) are applied to these points.

### 4.1.1 Visibility maps

The filtering phase consists in sorting the points into two categories: each point of the compared scan lies either in an area of space that has been effectively observed by the sensor during the scanning process of the reference scene (i.e. the laser beam went through this area at both epochs), or in a shadow area of the first sensor.

The process relies on a Z-buffer (Watkins 1970) which is a common structure in computer graphics applications. As most of ground laser scanners use rotating mirrors and/or axes that move with constant angle steps, their 3D measures can actually be expressed in 2D½: 3D points are projected into a 2D grid in the scanner angular coordinate system (see Figure 12). This grid is called a "depth image" or "depth map" (see Figure 13) and is equivalent to a Z-buffer. Every pixel in this image corresponds to a given direction and its associated value is the distance from the scanner to the nearest obstacle in this direction.

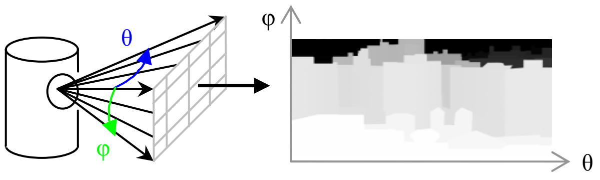

Figure 12 – scanning principle and depth image projection

Some holes may exist in the depth image as some laser pulses have not been reflected back. To fix this, a simple "hole-filling" process is applied to the image (holes are replaced by the mean value of its neighbours). Finally, if one knows also the maximum sensor range, it can be used to initialize the buffer (otherwise we initialise it with zero values).

The depth image is associated with the sensor position and orientation. With such a structure, the visibility of any 3D point can be determined: a queried 3D point is projected into the 2D image space; if the projected point is outside of the image, it means that the 3D point is outside the scanner field of view. Otherwise, the distance between the point and the scanner centre is compared to the buffer value at the same position. If the point is closer, it is labelled as *visible* (e.g. the corresponding space area has been actually visited by the laser beam) else it is *invisible*. This visibility test is made during the comparison process, before each nearest neighbour request. If it is positive, the Hausdorff distance is calculated, otherwise the invisibility type is stored (*out of field of view*, *out of range* or *hidden*). Invisible points will be processed in a specific way (see next section).

The visibility map is only computed for the reference cloud. If this cloud is composed of multiple scans, membership to the different point of views (*p.o.v.*) should be known for every point (as requested in Section 2). A visibility map is then computed for each *p.o.v.*, and the visibility query is made for every *p.o.v.* until one test is positive (e.g. the point is *visible* from at least one *p.o.v.*). Otherwise the point is *invisible*.

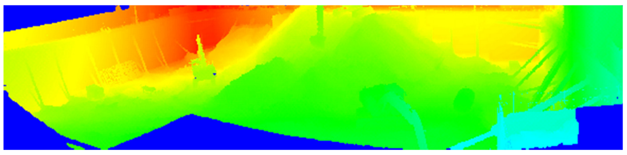

Figure 13 - depth image of the first scan of the building site (equivalent to the 1st day in Figure 1)

Note: as for graphical Z-buffers, an uncertainty value is used for the distance test. It is a positive epsilon variable added to the buffer range value during the test. It is a simple way to take errors into account (mainly sensor and registration errors), but also to arbitrarily consider small distances as differences and not as *hidden* configurations (depending on the objects sizes). For outdoor scenes with high ranges, we set epsilon between 10 cm and 1 meter. For indoor scanning, with lower ranges and a much higher precision, we use values between 3 cm and 10 cm.

### 4.1.2 *Invisible* points handling

It appears that *invisible* points can be classified into three groups:

- Points that are *out of field of view* or *out of range*. We can forget them, as there is no way to compare them.
- Points that are labelled as *hidden* because there has been a shape modification or a shift of the object surface, farer from the scanner (a true change).
- Points that are labelled as *hidden* because something has disappeared or shifted, revealing what was behind it, in its shadow (see Figure 14).

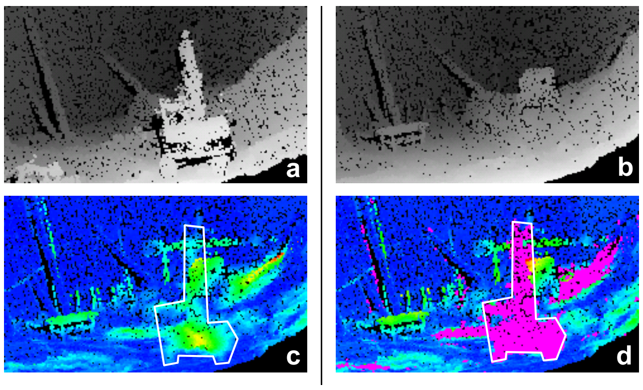

Figure 14 - Comparison of day 3 (b) with day 1 (a). With a simple Hausdorff distance (c), points in the shovel shadow are labelled as *different* (non-blue points inside the white polygon). After a visibility check (d), those points are labelled as *hidden* (in pink).

In order to make the distinction between the two *hidden* cases, a solution is to use a third scan. Given three scans acquired at three different epochs (*t-1*, *t* and *t+1*), lets consider a set of points that are labelled as *hidden* at epoch *t*. If those points have not moved between epoch *t+1* and *t-1*, then there is a great probability that they have not moved at epoch *t*. If they have moved at epoch *t+1*, the outside border of the shadow zone at epoch *t* can give a hint on what happened inside.

If there is no third scan or some ambiguities remain, different strategies can be proposed as a choice to the user:

1. To compare the remaining points in a 2D½ way (the user must specify a direction). It is most indicated for outdoor datasets or more generally for nearly 2D½ datasets. For example, the user may guess that gravity is the main reason for the displacement of some hidden points, and will therefore want to compare them to points vertically above or below them.
2. To compare the *hidden* points with their occluding points in the reference cloud (which can be identified via the Z-Buffer). If the two sets have at least partially the same shape, there is a good probability that they correspond to the same shifted surface.
3. To manually select and tag subsets of points with specific values.

Finally, whatever processes the user has chosen, he obtains one set of points labelled with distance values, and one set of points that could not be compared but which are labelled with visibility tags. This allows to produce accurate maps of 3D changes without false detection (see Figure 15-a).

To extract higher level information, a 3D connected components extraction process can be applied on this dataset. The user sets (graphically or numerically) a distance threshold to filter out points considered as unmodified (see Figure 15-b). Doing this, only points corresponding to "changes" remain under the form of isolated subsets of points (as they were connected to unchanged surfaces that have been filtered out). Then, the octree structure – considered as successive 2D slices of cells – is used to perform on each slice a simple binary connected components extraction process (cells can have two values: *empty* or *occupied*). The results are merged in 3D by regrouping components in the orthogonal direction (see Figure 15-c).

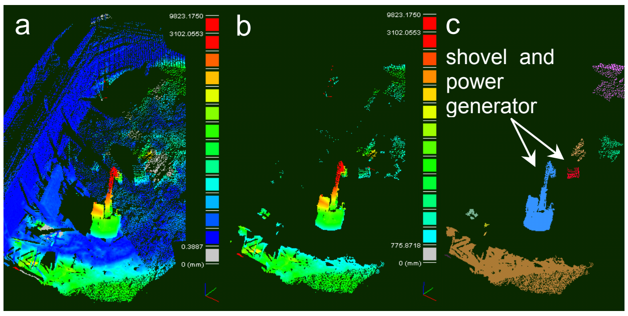

Figure 15 – Objects extraction principle: distance calculation (a), distance thresholding (b) and 3D connected components extraction (c).

## 5. CONCLUSIONS

We have presented a direct point-to-point comparison framework. Given two datasets acquired on the same area, one can use either preview processes (for on-site verification) or a more precise comparison process relying on Hausdorff distance. This last process can be refined if the sensor pose and scan membership information for every points are available. Speed constraint is assured thanks to the use of a light octree structure such as the one presented in section 3.1. Results can be used for basic change mapping or human driven verification, but also for higher level treatments, such as 3D connected components extraction and objects recognition. No time consuming process is needed, allowing the whole distance calculation and change detection process to be completed in a short time.

In case of an emergency or more generally if the user is not familiar with 3D point clouds, knowledge of the sensor parameters or manually setting threshold values can become critical issues. Therefore, we are now working on automatic distance thresholding to achieve unsupervised changing objects extraction (allowing automatic recognition of 3D objects for vehicle tracking or displacements monitoring). We are also working on automatic sensor pose estimation (directly from the 3D points repartition in space, without any information other than the sensor type). Another field of research is the classification and recognition of the different types of change that can occur on geometry, in order to produce cleaner and more informative results.

## REFERENCES

N. Aspert, D. Santa-Cruz and T. Ebrahimi, 2002. MESH: Measuring Error between Surfaces using the Hausdorff distance. In: *proceedings of the IEEE International Conference on Multimedia and Expo*, vol. I, pp. 705-708.

P. Besl and N. McKay, 1992. A method for registration of 3-D shapes. *IEEE Trans. Pattern Anal. Mach. Intell,* 14 (2), pp. 239-256.

M. Botsch, A. Wiratanaya, and L. Kobbelt, 2002. Efficient high quality rendering of point sampled geometry. In: *proceedings of the Eurographics Workshop on Rendering 02*.

P. Cignoni, C. Rocchini, and R. Scopigno, 1998. Metro: measuring error on simplified surfaces. *Computer Graphics Forum*, vol. 17(2), pp. 167–174.

F. Duguet and G. Drettakis, 2004. Flexible Point-Based Rendering on Mobile Devices. *IEEE Computer Graphics and Applications,* n°4 vol. 24.

J. G.M. Gonçalves, V. Sequeira, B. Chesnay, C. Creusot, S. Johnson and J. Whichello, 2003. 3D Laser Range Scanner for Design Verification. In: *proceedings of the 44th Meeting of the Institute for Nuclear Materials Management, Phoenix, Arizona*.

F. Mémoli and G. Sapiro, 2004. Comparing Point Clouds. *Eurographics Symposium on Geometry Processing*.

H. Murakami, K. Nakagawa, H. Hasegawa, T. Shibata and E. Iwanami, 1999. *Change detection of buildings using an airborne laser scanner*, ISPRS Journal of Photogrammetry & Remote Sensing, Vol. 54, pp. 148-152.

N-J. Shih, M-C. Wu and J. Kunz, 2004. The Inspections of As-built Construction Records by 3D Point Clouds. *CIFE Working Paper #090*, Stanford University.

T. Vögtle and E. Steinle, 2004, Detection And Recognition of Changes in Building Geometry Derived From Multitemporal Laserscanning Data, In: *proceedings of the XXth ISPRS Congress (Istanbul)*, Vol. 35, Part B2, p. 428-433.

G. Vosselman, B.G.H. Gorte and G. Sithol, 2004, Change Detection for Updating Medium Scale Maps Using Laser Altimetry, In: *proceedings of the XXth ISPRS Congress (Istanbul)*, Vol. 35, Part B3.

G. S. Watkins, 1970. A Real-time Visible Surface Algorithm. *Technical Report UTEC-CSc-70-101*, Dep. Comptr. Sci., U. of Utah, Salt Lake City.