Error-aware Construction and Rendering of Multi-scan Panoramas from Massive Point Clouds

Marc Comino, Carlos Andújar, Antonio Chica, Pere Brunet
VirVIG, Computer Science Department, Universitat Politècnica de Catalunya Jordi Girona 1-3, Barcelona, Spain

Abstract

Obtaining 3D realistic models of urban scenes from accurate range data is nowadays an important research topic, with applications in a variety of fields ranging from Cultural Heritage and digital 3D archiving to monitoring of public works. Processing massive point clouds acquired from laser scanners involves a number of challenges, from data management to noise removal, model compression and interactive visualization and inspection. In this paper, we present a new methodology for the reconstruction of 3D scenes from massive point clouds coming from range lidar sensors. Our proposal includes a panorama-based compact reconstruction where colors and normals are estimated robustly through an error-aware algorithm that takes into account the variance of expected errors in depth measurements. Our representation supports efficient, GPU-based visualization with advanced lighting effects. We discuss the proposed algorithms in a practical application on urban and historical preservation, described by a massive point cloud of 3.5 billion points. We show that we can achieve compression rates higher than 97% with good visual quality during interactive inspections.

Keywords: 3D reconstruction, range data, massive point clouds, error-aware reconstruction, compression, panoramas, interactive inspection.

1. Introduction

Reconstruction from accurate range data is becoming a more and more important topic. Processing of massive point clouds from lidar sensors involves a number of challenges, from data management to noise removal, model compression and interactive visualization.

When dealing with point clouds created by registering various range scans, a single surface can be represented by points coming from both nearby and distant sensors, with highly different expected measurement noise. Moreover, by merging multiple scans, the size of the resulting cloud rapidly grows, making it harder to handle in terms of resources.

In this paper we present a new methodology for the reconstruction of 3D scenes from massive point clouds coming from range lidar sensors. Our proposal is based on an error-aware, robust normal estimation and a panorama-based compact reconstruction and visualization.

Robust normal estimation is a key issue in most point-based tasks such as visualization and surface reconstruction. Existing methods are able to deal with noisy data under the assumption that every point has a measurement error with similar statistical properties. However, this assumption does not hold for point clouds coming from lidar scanners, because the expected measurement noise depends on the point’s distance from sensor and its reflective properties. We propose and discuss a normal estimation method that is able to deal with this problem by considering, for each point, a 1D directional probability distribution with variance proportional to its associated measurement error.

Streaming and visualizing a massive point cloud is not a trivial task since it might not even fit in memory. Our solution to this problem is to build panoramas at different points of interest and enabling navigation between them. For this, we propose an efficient method to build high-quality panoramas from arbitrary view points by combining data from multiple 3D scans.

The main contributions of our approach are:

- An error-aware normal estimation algorithm for each point of the point cloud. We have developed a method which is able to deal with noisy point clouds with non-uniform and anisotropic scanning error.

- A panorama-based 3D representation for gigantic point clouds that encodes 3D data in a compact way, with an easy and inexpensive color matching algorithm.

- An efficient algorithm for interactive navigation, supporting both realistic and illustrative rendering techniques.

As a test case, we processed a collection of 3D scans of the Mercat de Sant Antoni including the archaeological remains that were found underground, resulting in a point cloud consisting of 3.5 billion points.

The rest of the paper is organized as follows. Section 2 reviews the most relevant previous work on the subject. Section 3 provides an overview of our approach, and Sections 4, 5 and 6 present the preprocessing phases of our method, including the estimation of normal vectors, generation of panoramas and estimation of consistent panorama colors. Section 7 is devoted to
visualization and interactive rendering of the panoramas. Section 8 discusses our results with the test dataset. Concluding remarks are provided in Section 9.

2. Previous work

2.1. Normal Estimation

Surface reconstruction from unstructured point clouds is an area widely studied as explained by Berger et al. [2]. In this work we focus only on the problem of normal estimation since we do not need an actual surface for rendering. Moreover, we assume that we can trivially orient these normals in a consistent way since we know the position of the scanner head from which each point was captured.

Hoppe et al. [16] presented an algorithm for surface reconstruction from clouds of points which is based on estimating the signed geometric distance to the unknown surface and then applying a variant of marching cubes in order to get the reconstructed geometry. Particularly, they estimate normals by fitting the least squares tangent plane to the local neighbourhood of each point, which can be efficiently computed by principal component analysis (PCA). This approach is robust in the presence of noise but, as noted by Mitra et al. [20], choosing the right neighbourhood size is key to obtain a smooth result whilst preserving the local curvature.

Giraudot et al. [11] tackle the problem of having different levels of noise within the same point set by estimating a noise-adaptive robust distance function which is used to reconstruct the underlying surface. However, they treat all the points within a region equally without taking into account that, in a point cloud formed by a mixture of registered scans, some points might be more reliable than others. Moreover, although normals could be computed from the resulting meshes, they do not directly do normal estimation.

In the same sense, the following methods do surface reconstruction and require normals associated to each point as input. Nevertheless they introduce the idea of using extra per-point cues related to their reliability. Curless et al. [8] propose to associate to each point a confidence value that depends on the scanning technology. In their case, they associate lower confidence values to higher scanning grazing angles. Similarly, Fuhrmann et al. [10] introduce a new method for surface reconstruction from oriented sample points with an associated scale cue and some optional confidence information. The scale of a point refers to the finite surface area the point represents. Surfaces are approximated by the zero set of an implicit function which is the result of the weighted sum of a set of basis functions parametrized by each sample point.

In addition to noise, range scanners might suffer from other defects such as outliers, and there exist methods designed to robust against them [19, 17, 11, 21, 5]. Nurunnabi et al. [21] use the Minimum Covariance Determinant in order to compute a robust estimation of the covariance matrix for the neighbourhood of a point, on which they apply PCA to obtain the normal. Campos et al. [5] iteratively fit d-dimensional splats (d-jets) to the local neighbourhood of each point in a RANSAC-like loop in order to minimize the impact of outliers. In fact, they do surface reconstruction although normals could be extracted from their estimated splats. However, both methods consider a general noise model which is not sensor-dependent.

Another family of methods aims at consolidating the point cloud prior to performing the surface reconstruction process, as a way to deal with noise and outliers. The process of consolidation comprises a projection operator and resamples the original point cloud producing a surface implicitly defined as the fixed point of this operator. In this direction, Alexa et al. [1] introduce a moving least squares (MLS) surfaces for point-based methods.

Using a Gaussian weighting function that gives more importance to closer samples in the cloud, points are projected onto the local tangent space defined by a locally-fitted low-degree bi-variate polynomial. Guennebaud et al. [15] extend this method by using spheres for shape approximation, achieving higher robustness. Nevertheless, these approaches are not considering specific directional assumptions on the noise and may require input normals for each point which are usually computed by using PCA.

Lipman et al. [19] propose a method for point cloud consolidation requiring no normals. Their parameterization-free Local Projection Operator (LOP) tries to project a set of points onto the local multivariate median of the original cloud by ensuring that the resulting ones are distributed as uniformly as possible. Huang et al. [17] extend this approach to deal with non-uniform sampled input clouds. Then, an initial guess for normals is computed using traditional weighted PCA and improved by using a corrector loop consisting in one consistent normal orientation step and one orientation-aware PCA step.

One shortcoming for this approach is that data redundancy is reduced by uniformly placing samples regardless of the original sampling density, however some other works, such as [10], with the help of some extra cue exploit this redundancy in order to capture finer detail.

The work of Shi et al. [27] is also related to consolidation but, instead of defining a projection operator, they integrate multiple overlapping range images by performing a normal-aware clustering of the input samples and generate a simplified point cloud composed by cluster representatives computed using the mean-shift algorithm. Nevertheless, the result is limited by the quality of the normals resulting from the choice of the algorithm used to compute them.

One common flaw to all previous approaches is the assumption that the underlying surface is smooth. Other methods specialize in preserving sharp features, such as Boulch et al. [3] who randomly draw triplets of points from a given neighbourhood, compute the plane going through them and use a Hough accumulator to estimate a robust normal. Still, this technique requires high point density near sharp features to produce reliable results and highly depends on a correct choice of parameters in order to be able to smooth out the noise. Another sharp feature-preserving algorithm is given by Castillo et al. [6] who reformulated the least squares fitting problem and use non-linear least squares solvers in order to estimate, at the same time, the denoised points and corresponding normals. For this, they introduce new weights which penalize neighbours that lie
far away from the estimated planes and favour an even distribution of the output cloud. Zhang et al. [31] estimate a confidence value for the PCA normals and those under a certain threshold are considered to potentially belong to a feature. This information is used to guide a low-rank subspace clustering that segments the neighbourhood of each feature point into planar subspaces. Finally, the normal for these points is computed using PCA on the samples belonging to the subspace with minimum fitting residual. These methods apparently achieve good results when estimating normals in sharp hard edges. Nonetheless, they do not explicitly deal with noise or take into account its directionality.

Our main contribution to this field is a robust error-aware normal estimation algorithm. We use a standard PCA which works on a sensor-dependent covariance matrix that takes into account the directional and anisotropic structure of lidar-based errors. We integrate directional errors in order to compute sensor-based, directional covariance matrices. Our approach is able to deal with non-uniform sampling, and variable directional noise.

2.2. Point-Based Representations

In this study we consider point-based representations since they are the output of range-Lidar sensors. Moreover, the advantage of point clouds is their ability to be processed without having to consider topological information.

Pauly et al. [23] presented a free-form shape modelling framework for point-based representations that used a proxy geometry mixing unstructured point clouds with the implicit surface definition of the moving least squares approximation, in order to be able to exploit the advantages of implicit and parameteric surface models. In particular, they are able to support Boolean operations and free-form deformations.

Zwicker et al. [32] introduced a system for efficient 3D appearance and shape editing of point-based models. For this, they devised a point cloud parametrization scheme and a dynamic resampling policy based on a continuous reconstruction of the model surface.

More recently, Calderon et al. [4] proposed a complete framework for the morphological analysis of point clouds. By introducing a new model for the structuring element, based on a signed scalar field representation, and substituting the Minkowski sum with a new projection procedure, they are able to simulate dilations and erosions without the need of any kind of topological information.

If the position of the scanner head from which each point was captured is known, point normals can be trivially oriented and they can be used to determine whether a point is visible. However, even if this is unknown, a point cloud inherently contains information from which it is possible to extract the visibility of the points. This is the baseline for the work of Katz et al. [18] which present their HPR operator in order to approach this problem.

Dealing with point-based representations is a must when working with data coming from range-Lidar sensors. Nonetheless, these turn out to be more powerful and less error prone than conventional mesh-based representations. In this work we propose a method to convert point-based representations into a set of panoramas, which achieves high compression rates and can be rendered and inspected interactively.

2.3. Rendering of Point Clouds and Panoramas

Surface splats were first proposed for rendering purposes by Zwicker et al. [33]. Mutual overlap of splats in object-space guarantees a hole-free rendering in image-space however a naive approach might cause shading discontinuities. Zwicker et al. [33] proposed a high quality anisotropic anti-aliasing method based on the Elliptical Weighted Average (EWA) filter. Each splat is assigned a radially symmetric Gaussian filter kernel, such that a continuous surface signal in object-space is reconstructed by a respectively weighted averaging of splat data. Nevertheless this implementation was completely CPU-based which limited the real-time interactivity.

Preiner et al. [24] estimate local surface tangents and point densities of the splats interactively on the GPU. The algorithm relies on a screen-space algorithm to quickly find the k nearest neighbors of each point. The method supports visualization of dynamic scenes for reasonably sized point clouds (about 10M points).

Rusinkiewicz and Levoy [25] proposed a hierarchical structure, based on a pre-computed bounding sphere tree, that can be used for visibility culling, level-of-detail control, and rendering of point-based models.

More recent out-of-core rendering approaches benefit from GPU acceleration by organizing the hierarchical decomposition into blocks maintained out-of-core [13, 29, 14]. All these approaches build the hierarchy bottom-up through a simplification process. Hierarchical LOD approaches can render arbitrary point clouds at interactive rates, but do not benefit from the implicit 2.5D structure of point scans acquired from static Lidar equipment.

For data acquired from a discrete set of locations, image-based rendering approaches are a valuable tool to reduce the scene complexity while maintaining a fairly good rendering quality. Image downsampling is significantly simpler and faster than point cloud simplification. Furthermore, once we fix a parameterization of the unit sphere, pixels of the panoramic images can be associated with color, normal and depth data at independent resolutions. Our approach benefits from these advantages by converting Lidar data into polar-based panoramas.

Panoramic images (in particular 360-degree cylindrical images [7]) have been used to model scenes for decades, allowing the user to look around in arbitrary view directions. The panoramas can be warped on-the-fly to simulate camera panning and zooming. Walkthrough navigation can be achieved e.g. by hopping to different panoramic points, by precomputing video sequences between view-points [9], or by interpolating new views from panoramas resampled at grid points [22]. Our contribution to this field is the ability to combine robustly multiple Lidar scans into (depth and normal-enriched) panoramas resampled at arbitrary locations. These panoramas can be rendered efficiently by exploiting current GPU shaders (Section 7).
3. Outline of the approach

Our main goal is to represent gigantic datasets in a compact way suitable for efficient rendering. We assume that independent point clouds \( C_1, \ldots, C_n \) have been acquired from \( n \) independent scanner positions, and that a 3D registration of these point clouds has already been performed by some automatic or semi-automatic algorithm (see for instance the Iterative Closest Points method, [10]). Registration results in a set of \( n \) geometric transformation matrices \( T_1, \ldots T_n \) that align the individual point clouds to a common coordinate system. We will name \( S_k \) the 3D location of the scanner used to acquire \( C_k \). Individual points are represented by their 3D position \( (x, y, z) \), their \( RGB \) color and captured intensity \( i \) and the label \( k \) of their cloud \( C_k \). We use the obvious fact that during the registration of \( C_k \) and for any direction from \( S_k \), the scanner gets the closer data point and ignores all possible occluded points.

In what follows, a panorama \( P \) will be 3D polar representation of the scene. A panorama \( P \) stores its center of projection \( o_P \) and three images: a color map, a depth map and a normal map. Given any 3D point \( p \), the vector \( p - o_P \) is transformed to spherical coordinates \( (\cos \phi \cos \theta, \sin \phi, \cos \phi \sin \theta) \) and \( p \) is represented by encoding its distance \( d \) in the pixel \( q \) associated to \( (\theta, \phi) \) in the depth map of \( P \) (see Figure 2). In a similar way, the color attribute is represented in the corresponding pixel of the color map. The range of the rotation \( \theta \) around the vertical axis is \([0, 2\pi]\) whereas the range of the elevation angle \( \phi \) is \([-\pi/2, \pi/2]\). In our implementation, we have used image resolutions of 16384x8192 with \( RGB \) color maps, and 16-bit depth maps, providing a 2 mm precision in depth values at the order of 128 meters. We also include an additional normal map with the estimated normal vectors at the corresponding data points. All these maps define an equirectangular projection of a sphere.

Please note that, although a finite set of panoramas cannot completely model complex and general 3D shapes with strong occlusions, a panorama \( P \) with \( o_P = S_k \) is always a valid discrete representation of the point cloud \( C_k \), since occlusions in \( P \) are identical to occlusions in \( C_k \). Of course, a panorama \( P \) is still an approximate discrete representation of \( C_k \) due to the finite resolution of the images in \( P \). We use this property in our normal estimation algorithm, see next Section.

In a first step, we estimate normals at each point \( p \) of the point cloud and generate a compact, panorama-based 3D representation of the scene, by creating a set of panoramas \( P_1, P_2, \ldots P_m \) with \( m \) equal or greater than \( n \). This set of panoramas includes \( n \) panoramas centered at \( S_k \), plus \( m - n \) extra panoramas located at relevant positions other than \( S_k \). The location of the center-points of these extra panoramas could be computed as the result of a previous optimization, but this is outside the scope of this paper. In what follows, we will consider that the set of center-points of extra panoramas (if any) is an extra input of the algorithm.

Panoramas are computed from the set of registered point clouds \( (C_k, T_k) \) for \( k = 1, \ldots n \). As shown in Figure 1, registered points from all point clouds are first inserted into a voxelization of the bounding box of the whole scene for efficiency purposes. Then, normal vectors are estimated for each data point.

We use a tailored, anisotropic error-aware robust estimation of the normal vectors, as discussed in next Section. These normal vectors will be used for illustrative rendering, shading, and to perform back-face culling. Then, voxels are processed in voxel layers from the panorama center and outwards, see Figure 1. Points inside each voxel of the active layer are mapped to the unit sphere. Panorama pixels are then traversed and, for each void pixel, the best candidate point is selected through adaptive sampling; non-void pixels that have already been computed in previous steps are skipped. This sampling takes into account the projected distance of each sample to its corresponding line, its relative depth and its noise confidence to compose a weight with which to combine them. In this way, depth, color and normal maps are updated by visiting all voxels in the active layer and by traversing all layers from the panorama center to the boundary of the voxelization.

One of the key ingredients of our normal estimation algorithm and panorama generation is the use of two pieces of information that are usually not taken into account: sensor locations and directional scanning errors. By simply adding an integer index \( k \) to each data point \( p \), data points become sensor-aware and can be associated to their scanning line \((S_k, p)\). Directional, sensor-dependent scanning errors along scanning lines can then be statistically modelled. In the case of laser scan sensors, considering anisotropic, directional scanning errors results on an improved normal estimation.

Color requires a special treatment. Time-of-flight (TOF) scanners capture a reliable intensity value \( i \) of the laser echo simultaneously with point coordinates \((x, y, z)\), but color of the samples is usually obtained from photos after the geometry scanning process has finished. This may cause artifacts due to dynamic objects in the scene and different illumination/exposure conditions. We propose a simple color correction strategy that has proved to be very effective, see Section 6.

The set of panoramas is a compact, redundant 3D representation of the initial dataset. It supports interactive navigation and inspection of the reconstructed digital model, as discussed in Section 7. Moreover, this panorama-based representation supports advanced lighting effects, expressive rendering and distance queries, Section 7.

Note that we are not reconstructing any geometric surface.

We focus on creating independent panoramas. First of all, as any 3D point \( p \) has a unique sensor position, we ensure that its normal vector \( n_p \) always faces its 3D sensor origin \( s_k \). Even when two sensors record the same surface from different angles, each one of them captures its own set of disjoint points.

Each point has a unique sensor position and its normal vector will always face the sensor position by construction. Then, for any panorama, our algorithm performs backface culling by using the fact that all normals of visible points must face the panorama centerpoint.
A robust method for estimating the normal at each point in a point cloud is a key issue for many applications such as visualization (shading, coherent back-face culling), splatting and surface reconstruction. We show that a crucial issue for this problem is taking into account the directional structure of the noise present in the data. Hoppe et al. [16] compute the normal at each point as the normal to the fitting plane obtained by applying the total least square method to the k-nearest neighbourhood of the point. This method is robust to the presence of noise due to the inherent low pass filtering. Furthermore, Mitra et al. [20] analyze the noise in the data in order to estimate the optimal neighbourhood size. Both of these methods perform a least square fitting and, therefore, assume that the points are normally distributed around the fitted plane. However, when we have clouds coming from multiple registered scans, the noise associated to points that are relatively close can greatly differ (Figure 3b). In the next paragraph we propose a new, error-aware normal computation algorithm which is based on considering small Gaussian intervals instead of points and

4. Robust normal estimation

Layer-wise front-to-back traversal

Voxelization

Normal estimation

Panorama center

Adaptive sampling

Adaptive sampling

Adaptive sampling

Figure 1: Overview of the panorama generation pipeline. We start with a number of point clouds acquired from the environment. Registered points from all point clouds are first inserted into a voxelization of the bounding box of the whole scene for efficiency purposes. The voxelization involves no data loss since all points will still have a chance to contribute to the final panorama. Then, normal vectors are estimated for each data point through an anisotropic, error-aware robust estimation algorithm. Normal vectors will be used for back-face culling, illustrative rendering and shading. Next, voxels are processed in voxel layers from the panorama center and outwards. Panorama pixels are then computed and filled by selecting, for each void pixel, the best candidate point through adaptive sampling.

Figure 2: Two point clouds $C_k$ and $C_{k+1}$ with their corresponding sensor origins $s_k$ and $s_{k+1}$. The point $p$ in the cloud $C_{k+1}$ projects to the pixel $q$ in the additional, virtual viewpoint panorama $P$ located at a relevant position $o_p$, different from all $s_k$.

400 non-uniform scanning errors. It is based on the spatial distribution of the scanner locations $s_k$.

We start by computing a normal vector for each data point. As any 3D point $p$ has a unique sensor position, we ensure that its normal vector $n_p$ always faces its 3D sensor origin $s_k$.

Our normal vector estimation at a data point $p$ is inspired by Hoppe et al. [16]'s method (see Figure 4). When the scan captures a point $p$ there is always an error in depth which depends on the distance to the point and its reported intensity. This is shown in Table 1, which presents error specifications for a high-end 3D scanner. We therefore deal with 1D directional error intervals with probability distributions which are Gaussians centered at each point $p$ and with a variance proportional to the error associated with the distance between $p$ and its scanning centerpoint $s_k$. We neglect the errors in directions orthogonal to the line $p - s_k$ as scanners have much higher resolution in $\theta$ and $\varphi$ directions than in depth, and we use the variance $(\text{distance, intensity})$ function supplied by the scanner specifications. We compute the centroid and covariance matrix of the set of points $N(p)$ in the neighbourhood of $p$ by assigning a weight to each point/segment in $N(p)$ that represents its reliability, see Figure 4, and by integrating the contribution in each segment of the points in $N(p)$. We use segment weights of the form $w_p = e^{-k_2 \sigma_p^2}$ ($\sigma_p$ being the error associated with the segment for point $p$ and $k_2$ being a constant used to convert errors to mm) and a quadratic Gaussian approximation for the contribution of the points in each segment during integration. We use [16] to compute the estimated normal vector from the covariance matrix of sets of weighted segments: we perform Principal Components Analysis on it and estimate the normal as the eigenvector corresponding to the smallest eigenvalue. Finally, normal orientations are automatically derived from the fact that $n_p$ must always face its 3D sensor origin $s_k$.

Note that, by integrating along the directional error intervals, we obtain sensor-dependent and directional covariance matrices which result in better and sensor-adapted normal estimations. To integrate contributions along segments, we use closed formulas based on approximating Gaussians for efficiency purposes. We have compared a uniform weights approximation (no
Figure 3: Partial point cloud with 9.5 million points coming from 5 different sensors. The need for a robust normal estimation is clear when realizing how the expected error changes according to sensor locations: (a) shows the color-coded sensor ID contributing to each point, and (b) shows the expected error according to the scanner specifications (brighter values represent larger errors). Normal estimation results with different methods: (c) Hoppe et al. [16]; (d) Our approach with uniform weights approximation; (e) our approach with quadratic weights approximation; (f) our approach with sinusoidal weights approximation. The noise in the normals (which is specially significant on the left side on the cloud) practically disappears with our method. A zoom of this region can be seen in Figure 5.

In Figure 5 we qualitatively compare different normal estimation approaches. Figure 5c does not handle well mixtures of points with different error distributions. Finally, Castillo et al. [6]’s approach (Figure 5d) is designed to filter noise and recover sharp features but, in this case, the method fails to estimate proper normals for very noisy points as they are being treated as features.

5. Panorama generation

This Section first gives an overview of the method and then gives a more detailed explanation of each step in the subsections.

To build the set of panoramas, registered points from all point clouds are first inserted into a voxelization of the bounding box of the whole scene. This is done for scalability, so that fractions of the point cloud can fit in main memory, and for efficiency purposes. Also, normal vectors are estimated for each data point as described above. To avoid voxel aliasing effects, the normal estimation at point \( p \) takes into account 27 scene voxels, including the 3 x 3 neighborhood of the \( p \) voxel. Then, for each panorama, voxels are processed in closed layers from the panorama center, Figure 1.

For the sake of conciseness, we will note \( \text{op} \) the center point of the current panorama that we are generating. Let \( C \) be the cloud formed by all points in the current layer of the voxelization. For each point \( p \in C \) we know its normal \( n_p \) and we can therefore perform backface culling by discarding it if \((p - \text{op}) \cdot n_p > 0\). Observe that normal vectors are per-point whereas backface culling is panorama-dependent. Then, we build an auxiliary kd-tree with all points in \( C \) to speed up neighbourhood searches.

The panorama pixels are traversed and, given a panorama pixel \( q \), the kd-tree is used to search and compute the subset of points in \( C \) that are at a certain search distance of the straight line \( l_q \) defined by \( q \) and \( \text{op} \). Points in \( N_q \) are considered as surfels \( s_p \) with normal vector \( n_p \) (see Section 5.2). Then, \( N_q \) is pruned by removing all points \( p \) such that \( s_p \) does not intersect \( l_q \). After pruning, a reduced neighbourhood \( N_{q}' \) is obtained with the \( C \) points that will be relevant to \( q \). In a final step, \( C \) points \( p \) in \( N_{q}' \) are averaged with a Bilateral Filter driven by the distances between the projections (onto

Table 1: Range noise (RMS) as a function of depth and material reflectivity for a high-end pulsed laser scanner. Some of these values even exceed the point spacing provided by the scanner (at maximum resolution, the point spacing at 10 m is 0.8 mm) and thus prevent non-error-aware techniques from computing robust normals. Source: Leica ScanStation P20 Product Specifications.
5.1. Cone-shaped search

For each panorama pixel \( q \) we would like to compute the set of points \( N_q \) that will be relevant for computation. For this, we would like to find all the points contained in a cone-shaped volume with apex at \( \text{op}_q \) and an aperture angle that is a function of the panorama resolution. To speed-up this process, we construct, for each voxelization layer, a single kd-tree with the points of the \( C \) layer mapped onto a unit sphere centered at the unit sphere centered at \( \text{op}_q \) and \( l_q \) and by the differences between normal directions.

5.2. Splatting

In what follows, we will consider input sample points \( p \) to represent surfels \( s_p \) contained in the plane \( \pi_p : (p, n_p) \) with an associated radius parameter that is proportional to the distance \( d(p) \) between \( p \) and its source sensor origin. Particularly, we compute this radius \( r_p \) as:

\[
r_p = \frac{d(p) k_s}{2}
\]

Where \( k_s \) is a sensor-dependent parameter related the spacing between points used to capture the scene. In our case, we configured the Leica ScanStation P20 to use a spacing of \( 3 \times 10^{-4} \text{m} \) at depths of 1 m.

5.3. Refining the Neighbourhood

Because in \( N_q \) we can have surfels of multiple sizes and orientations, we must find a way to determine the contribution of each of them to the panorama pixel \( q \). One option could be determining the intersection \( t_{lp} \) of \( \pi_p \) and \( l_q \) and discarding those surfels for which the distance between \( d(p, \mu_p) = |\mu_p - p| \) is bigger than \( r_p \). We propose an adaptive approach to this problem which consists on determining the contribution \( w^p_{\mu} \) for surfel \( s_p \) by modelling it as a normalized gaussian distribution:

\[
w^p_{\mu}(x) = \frac{1}{r_p \sqrt{2\pi}} e^{-x^2/(2r_p^2)}
\]

The contribution of \( p \) for a given panorama pixel \( q \) is computed as \( w^p_{\mu}(d(p, \mu_p)) \).

Intuitively, surfels with large \( r_p \) are blurred and surfels with small \( r_p \) are added as high frequencies. Also, smaller \( d(p, \mu_p) \) will yield bigger contributions, and \( d(p, \mu_p) \) decreases the closer \( p \) is to \( l_q \) and the more parallel \( n_p \) is to \( l_q \), as expected.

5.4. Bilateral Filtering

We compute color and depth using bilateral filtering by averaging neighbor points with a weighting term \( w^d_p \) that enhances values near edges. Let \( p^* \in N_q \) be the point whose intersection \( \mu_p \) is closest to \( \text{op}_q \). Then, for a point \( p \in N_q \):

\[
w^d_p = e^{(-|\mu_p - p|^2)}
\]
We compute the color $c_q$ associated to the panorama pixel $q$ by a weighted average of the colors $c_p$ of the samples in $N_l_q$:

$$c_q = \frac{\sum_{p \in N_l_q} w_p^e w_p^s w_p^d c_p}{\sum_{p \in N_l_q} w_p^e w_p^s w_p^d}$$ (5)

Where $w_p^e$ is the contribution associated to point $p$ and $w_p^e$ is a weighting term related to the error $\sigma_p$ associated to $p$, of the form:

$$w_p^e = e^2_p$$ (6)

Then, the depth $d_q$ for the panorama pixel $q$ is computed as:

$$d_q = \frac{\sum_{p \in N_l_q} w_p^e w_p^s w_p^d \|a_p - op\|}{\sum_{p \in N_l_q} w_p^e w_p^s w_p^d}$$ (7)

Finally, the normal for point $q$ is computed using the approach explained in Section 4 on $N_l_q$.

### 6. Color Estimation and Correction

In Section 5.4 we have explained how to compute the color $c_p$ for panorama pixel $q$ as a weighted average of the colors $c_p$ for points $p \in N_l_q$. TOF scanners capture a reliable intensity value $I$ of the laser echo simultaneously with point coordinates, but colors $c_p$ of samples are obtained from photos after the scanning process has finished. This may cause artifacts due to dynamic objects present in the scene. Also, the scan takes multiple pictures and combine them to generate the color of the samples, but these pictures are potentially taken with different illumination conditions since photo acquisition can take several minutes or even hours. Color cameras usually try to adjust the settings by controlling exposure (lens aperture, shutter speed) to the individual lighting conditions of each shot. This means that automatic exposure causes image colors to vary across different shots. Because of this, discontinuities that are not present in the intensity values can appear in the generated panoramas (see Figures 6 and 7). Moreover, color is not consistent between different scans, which is not desirable. Consequently, we propose a simple color correction strategy, in order to improve the consistency between colors $c_p$ before they are averaged, that has proved to be very effective. Our approach consists in computing the intensity $i_{rgb} = (r + g + b)/3$ of the color as the mean of the three channels and, then, substituting this by the scan reported intensity $i$, e.g. the output red channel is computed as $r = i / i_{rgb}$. The results are shown in Figure 8.

### 7. Rendering and interactive navigation

Algorithms for rendering single panoramas and multi-perspective panoramas have been studied for years, both for RGB and RGBZ panoramas (see [28, 12] for a review). Reconstructed panoramas can be explored interactively in multiple ways. The simplest modality uses a single panorama. In this case, to avoid image gaps corresponding to non-sampled parts, the camera is usually constrained to remain at the panorama’s center of projection, and only zoom and rotation are supported. Figure 9a shows the render of a single panorama. In this modality, unlit rendering requires just the color map, whereas the normal and depth maps allow for real-time lighting and illumination effects such as ambient occlusion [26] and shadow mapping [30], see Figures 16 and 17.

A more rich modality for exploring panoramas is to allow free camera movement. In this case multiple panoramas from different perspectives are combined to provide a better coverage of the scene (Figures 9, 10 and accompanying video).

We tested two different algorithms for panorama rendering (Figure 11). The first one uses point splats, whereas the second one uses a polygonal mesh. In both cases, our OpenGL-based implementation feeds the graphics hardware with $V = u \times v$ vertices from a highly tessellated unit sphere, where $u$ is the number of vertical segments (meridians) and $v$ is the number of horizontal rings (parallels) on the sphere.

Each vertex requires both $(x,y,z)$ coordinates as well as $(s = \theta/(2 \pi), t = (\phi + \pi/2)/\pi)$ texture coordinates (for the sake of simplicity, here we assume the panorama provides a complete 360° view of the scene and thus $\theta \in [0, 2\pi]$ and $\phi \in [-\pi/2, \pi/2]$).

We use a Vertex Buffer Object (VBO) to store sphere vertices. Inserting all vertices in the VBO would take $5 \times 4 \times u \times v$ bytes. For a 16384×8192 panorama, the size of the VBO would
be 2.5 GB. A more space-efficient solution is to insert in the VBO the vertices of a single ring, with attributes \((s, \sin \theta, \cos \theta)\). The resulting VBO takes \(3 \times 4 \times \mu\) bytes (192 KB for the 16384×8192 panorama). In this case, the application draws the VBO \(v\) times per panorama, one for each ring. We send (as uniforms) the three values \((t, \sin \theta, \cos \theta)\) corresponding to the ring being drawn. This allows the vertex shader to recover the original unit-sphere vertices \((\cos \theta \cos \phi, \sin \theta, \sin \theta \cos \phi)\) with no trigonometric function calls. Besides the space-saving advantage, this approach is much more flexible in terms of adjusting dynamically the level of detail (i.e. the number of vertices \(V\) used to sample the panoramas), to get a suitable tradeoff between rendering speed and sampling rate.

In the case of splat rendering (Figure 11-top), the vertex shader unprojects the point according to its depth (sampled from the panorama’s 16-bit depth map) and sets the splat size accordingly. We used OpenGL (square) points, although other splat shapes are also possible [25].

In the case of mesh rendering (Figure 11-bottom), we use a similar approach but drawing triangle strips instead of OpenGL points. The VBO is constructed as above but repeating each vertex twice (each strip takes vertices from two consecutive rings). For arbitrary view points, a geometry shader is required to prevent faces with sharp depth changes to appear extruded along a panorama’s radial direction (Figure 12). Let \(m\), \(max\) be the min/max depth values sampled from the depth map at the triangle vertices. When \(\max\) – \(\min\) is above some threshold \(\epsilon_d\), the depth of the three triangle vertices is set to \(\min\) (see Figure 12 for \(\epsilon_d\) values). This prevents large extruded surfaces while avoiding unnecessary gaps when rendering the panorama from the sensor location.

In both cases the fragment shader simply samples the panorama’s color and normal maps to perform lighting computations.

The accompanying video shows walkthroughs and flythroughs using multiple panoramas. As discussed in next Section, we reach point throughputs of 1–4 billion points per second.

Although the mesh-based approach is slower than the point-based approach (Section 8), the mesh-based method provides higher quality images at extreme zoom levels (Figure 13) and is less sensitive to reducing the number of vertices (depth map samples), provided that colors and normals are still sampled from high-resolution maps (Figure 14).

Panorama visualization also supports distance queries at any moment during the scene inspection. Upon user selection of two image pixels, the algorithm simply has to compute the two corresponding pixels \(q_1\) and \(q_2\) and their panoramas \(P_{k1}\) and \(P_{k2}\). The required distance is the Euclidean distance between the 3D points associated with \(q_1\) and \(q_2\).

8. Results and discussion

We have tested the proposed algorithms on a huge data set (including 31 different 3D scans) of the building known as Merce cat de Sant Antoni. This building has an extent of 5,214 m² on a city block of about 15,876 m². The building is composed of four big arms that converge in a big octagonal dome 28 meters high.

This dome is supported by eight big steel columns. The whole building is undergoing a reshuffling which uncovered some archaeological remains and part of the medieval city walls. A Leica ScanStation P20 was used to digitize key parts of the building, the remains, and the digging progress. The complete dataset included 31 ASCII files containing information on 3,487,095,733 points and requiring a total of 157.3 GB.

The storage requirements of the point clouds of the initial set of 31 scans, in binary format, was 45.8 GB. In contrast, the set of panoramas representing the same scene (building plus diggings) required 1.08 GBytes. We therefore achieved a compression ratio of 1:42, i.e. a space savings of 97.64%. The generation of the panoramas took about 15 hours (30 min per panorama) on a PC with an Intel Core i7 at 4GHz and 32 GB of RAM. Table 2 shows a breakdown of the time needed to build all panoramas using only the points above the floor (a cloud of 372 million points), while comparing the effect of voxel size. As expected, applying a higher resolution results in increased processing times, but the quality of generated panoramas also improves. Figure 15 shows a close-up of a panorama built at different resolutions. For our model, 1 meter resolution already results in good quality panoramas.

<table>
<thead>
<tr>
<th>Voxelization</th>
<th>0.5 cm</th>
<th>1 m</th>
<th>10 m</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal estimation</td>
<td>9° 37’’</td>
<td>7° 5’’</td>
<td>3° 47’’</td>
</tr>
<tr>
<td>Panorama sampling</td>
<td>1h 10’ 43’’</td>
<td>24° 20’’</td>
<td>34° 25’’</td>
</tr>
</tbody>
</table>

Table 2: Running times of the three steps of the building algorithm (rows) for different voxelization sizes (columns).

Figure 18 shows several results of our algorithm with points of view far away from sensor locations, therefore requiring the integration of data from many scans. The figure shows the estimated normals, the color-coded sensor ID contributing to each pixel and the color map of the panorama.

Tables 3 and 4 show the performance of the free-camera rendering algorithms (point-based and mesh-based) for varying sampling rates on NVidia GTX 770 hardware at Full HD resolution. The number of points refers to the number of vertices of the unit sphere used to render the panoramas, which also determines the number of samples taken from the depth maps.
Figure 9: A single 8192 × 4096 panorama rendered (mesh-based algorithm) from its center point (a) and from a distant point (b). Adding additional panoramas (c) improves coverage and allows for free camera navigation. The aerial view shown in (c) is far apart from the original sensor locations, shown as color spheres.

Figure 10: Combining multiple 8192 × 4096 panoramas to improve coverage. From top to bottom: 1, 2 and 5 panoramas (mesh-based rendering).

Table 3: Performance of the point-based rendering algorithm when rendering N panoramas at increasing sampling rates. The table shows both throughput (millions of points per second) and frame rate (fps, within parenthesis).

<table>
<thead>
<tr>
<th>N</th>
<th>2048×1024</th>
<th>4096×2048</th>
<th>8192×4096</th>
<th>16384×8192</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3,380 (1612)</td>
<td>4,404 (525)</td>
<td>4,630 (138)</td>
<td>4,563 (34)</td>
</tr>
<tr>
<td>2</td>
<td>1,140 (272)</td>
<td>1,207 (72)</td>
<td>1,207 (18)</td>
<td>1,342 (5)</td>
</tr>
<tr>
<td>3</td>
<td>918 (146)</td>
<td>956 (38)</td>
<td>1,006 (10)</td>
<td>1,207 (3)</td>
</tr>
<tr>
<td>4</td>
<td>838 (100)</td>
<td>1,174 (35)</td>
<td>1,207 (9)</td>
<td>1,073 (2)</td>
</tr>
<tr>
<td>5</td>
<td>775 (74)</td>
<td>880 (21)</td>
<td>1,006 (6)</td>
<td>1,342 (2)</td>
</tr>
</tbody>
</table>

Table 4: Performance of the mesh-based rendering algorithm when rendering N panoramas at increasing sampling rates. The table shows both throughput (millions of points per second) and frame rate (fps, within parenthesis).

<table>
<thead>
<tr>
<th>N</th>
<th>2048×1024</th>
<th>4096×2048</th>
<th>8192×4096</th>
<th>16384×8192</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>480 (229)</td>
<td>495 (59)</td>
<td>503 (15)</td>
<td>537 (4)</td>
</tr>
<tr>
<td>2</td>
<td>486 (116)</td>
<td>503 (30)</td>
<td>536 (8)</td>
<td>537 (2)</td>
</tr>
<tr>
<td>3</td>
<td>491 (78)</td>
<td>503 (20)</td>
<td>503 (5)</td>
<td>402 (1)</td>
</tr>
<tr>
<td>4</td>
<td>495 (59)</td>
<td>503 (15)</td>
<td>536 (4)</td>
<td>536 (1)</td>
</tr>
</tbody>
</table>

Table 11: OpenGL pipelines to render the panoramas: point-based (top) and mesh-based (bottom).

The point-based pipeline was 2×-9× faster than the mesh-based approach. With a single panorama, we were able to draw up to 4.5 billion points per second. When multiple panoramas were combined, we still got a throughput around a billion points per second. For Full HD resolution, a sampling rate of 4096×2048 per panorama provided excellent quality results. At this resolution, we could render five panoramas simultaneously (resulting in about 41M points) at 21 fps.

The mesh-based rendering algorithm was consistently slower, with a roughly constant throughput of about 500 M points per second. We could render 4 panoramas at 2048×4096 resolution at about 60 fps. Although slower, the mesh-based approach generated higher quality images for views far part from the sensor locations, and was less sensitive to the sampling rate.

The accompanying video (see supplemental material link below) shows interactive inspections of the test dataset. The video was recorded at Full HD resolution, then reduced to 960×720 to keep the movie size below 60 MB.

Limitations. The quality of our panorama-based representation depends on the number and location of the individual scan acquisitions and also on the distribution of the centers of the reconstructed panoramas. This is however an inherent limitation of any scan-based reconstruction, because of the lack of data.
in occluded regions (see artifacts in Figure 10 due to missing parts). We have partly addressed this issue by inducing virtual visits that can be guided by landmarks at scanner locations (see accompanying video, where these landmarks are shown as colored spheres in the reconstructed scene).

One shortcoming of our normal estimation algorithm, which is inherent in most PCA-based methods, is the assumption that the underlying surface is smooth. We do not deal explicitly with sharp edges at this step, nonetheless we are doing fine when recovering sharp edges and sharp discontinuities in depth thanks to the bilateral weights.

9. Conclusions and future work

We have proposed an algorithm for creating arbitrary-viewpoint, multi-scan panoramas from massive point clouds. We take a set of registered point clouds and voxelize it while computing normals for their points. For normal estimation we employ a new, error-aware algorithm. The voxelization is traversed by layers in a front-to-back order. At each step we consider the points in a single layer and map them into an image using their spherical coordinates. An elegant splatting strategy has been proposed in order to fill the gaps caused by regions with a low point density, and suitable color correction approximates the albedo of the scene, guaranteeing more uniform colors within and between panoramas.

We have proposed also a visualization tool that enables the interactive inspection of scenes represented by a set of panoramas. Lighting effects such as direct lighting, ambient occlusion and shadow mapping are well supported.

In the future we plan to explore the application of the neighborhood segmentation ideas from Zhang et al [31] to our error-aware normal estimation approach, to further improve normals around features. We intend to include an error-aware resampling of the scanned points in the space voxelization in order to improve the accuracy during the generation of panoramas.

We want also to investigate the use of panoramas with depth intervals, by encoding confidence intervals together with depth values. Finally, we would like to investigate the use of an optimization preprocess to estimate the optimal location for the panorama centers.

Acknowledgements

This work has been partially funded by the Spanish Ministry of Economy and Competitiveness and FEDER under grant...
Figure 14: Effect of reducing the number of vertices of the unit sphere used to render the panorama, for the point-based (top) and mesh-based (bottom) pipelines. From left to right: 1024×512, 512×256, and 256×128. In all cases the color map was 4096×8192.

Figure 15: Closeup of a generated panorama using different voxelization resolutions (left: 0.5 meters, center: 1 meter, right: 10 meters). The difference between using 0.5 or 1 meter is barely perceptible, while errors are apparent with 10 meters. Surfaces that should be hidden become visible, thus interfering with color estimation.

References

Figure 18: Results of our panorama generation algorithm with points of view far away from sensor locations and thus requiring the integration of data from many scans. From top to bottom: estimated normals, color-coded sensor ID contributing to each pixel, and final color panorama. The middle column corresponds to a zenithal point of view.


URL http://doi.acm.org/10.1145/2700428


URL http://doi.acm.org/10.1145/1230100.1230113


URL http://dx.doi.org/10.1117/12.386541

