The 6th International Symposium on Virtual Reality, Archaeology and Cultural Heritage
VAST (2005)
M. Mudge, N. Ryan, R. Scopigno (Editors)

Rapid Visualization of Large Point-Based Surfaces

Tamy Boubekeur[*], Florent Duguet[+] and Christophe Schlick[*]

[*]: LaBRI - INRIA - CNRS - University of Bordeaux [+]: INRIA Sophia Antipolis - ENST Paris

Abstract

Point-Based Surfaces can be directly generated by 3D scanners and avoid the generation and storage of an explicit topology for a sampled geometry, which saves time and storage space for very dense and large objects, such as scanned statues and other archaeological artefacts [DDGM]. We propose a fast processing pipeline of large point-based surfaces for real-time, appearance preserving, polygonal rendering. Our goal is to reduce the time needed between a point set made of hundred of millions samples and a high resolution visualization taking benefit of modern graphics hardware, tuned for normal mapping of polygons. Our approach starts by an out-of-core generation of a coarse local triangulation of the original model. The resulting coarse mesh is enriched by applying a set of maps which capture the high frequency features of the original data set. We choose as an example the normal component of samples for these maps, since normal maps provide efficiently an accurate local illumination. But our approach is also suitable for other point attributes such as color or position (displacement map). These maps come also from an out-of-core process, using the complete input data in a streaming process. Sampling issues of the maps are addressed using an efficient diffusion algorithm in 2D. Our main contribution is to directly handle such large unorganized point clouds through this two pass algorithm, without the time-consuming meshing or parameterization step, required by current state-of-the-art high resolution visualization methods. One of the main advantages is to express most of the fine features present in the original large point clouds as textures in the huge texture memory usually provided by graphics devices, using only a lazy local parameterization. Our technique comes as a complementary tool to high-quality, but costly, out-of-core visualization systems. Direct applications are: interactive preview at high screen resolution of very detailed scanned objects such as scanned statues, inclusion of large point clouds in usual polygonal 3D engines and 3D databases browsing.*

1. Introduction

Most of the visualization systems designed for large 3D objects (tens or hundreds of millions samples) are based on an initial mesh. Even high-quality multiresolution systems such as QSplat [RL00] or the Sequential Point Trees [DVS03], regularly mentioned for their ability to use points in order to display gigantic models, require such an initial mesh, and do not directly handle point clouds. In the case of very large scanned objects (e.g. Digital Michelangelo [LPC*00]), a non trivial surface reconstruction has thus to be performed. This process is very time consuming (from hours to days of computation on a single workstation), and is followed by other several expensive algorithms before interactive display is actually possible [CGG*04, GBBK04]. Moreover, in addition to the point data, a large memory overhead is required to store the topology, which becomes challenging when working on standard workstations.

It is clearly out of the scope of this section to recall all the approaches that have been proposed in recent years to obtain an interactive visualization of large objects. Essentially, these works can be classified according to three different principles. The first one, distributed rendering [WDS04], has shown its efficiency on models reaching one billion samples. However, this method requires expensive hardware configurations (PC clusters) and, since it is based on ray-tracing, is limited to low resolution rendering if an interactive framerate is required. The second one, out-of-core visualization [Lin03, Tol99], proposes to use some cache friendly data structures, both for storage and rendering, which offer efficient disk-to-memory updates according to the modification of viewing parameters. *Adaptive Tetra Puzzle* [CGG*04] is currently one of the most efficient systems, reaching the competitive framerate of 60 frames per second with the well-known St Matthew model [LPC*00]. Other out-of-core methods have also focused on advanced real-time rendering effect, such as shadows [GBBK04]. Finally, the last proposed principle tries to (dramatically) reduce the amount of data, while preserving almost the same appearance for the rendered object. In [COM98], the very innovative idea of mesh simplification combined with simultaneous texture generation to capture the underlying details has been proposed. Multi-resolution approaches, such as *Progressive Meshes* [Hop96], can also be enhanced by combining a normal map with simplified geometry [SSGH01]. Nowadays, such a process can be directly implemented on programmable GPU that allow the use of *normal maps* to represent fine details within a per-pixel shading [TCS03]. But it should be noticed that a complete parameterization of the model has to be performed to construct the normal map, which is a challenging task for huge models.

Multi-resolution directly on point clouds is proposed in the *Layered Point Cloud* [GM04], as well as an efficient compression scheme in the DuoDecim system [KSW05], but these methods require several hours for processing a model like the St Matthew [LPC*00]. We rather target one order of magnitude faster preprocessing, and we would like to take benefit from **polygonal** rendering. To our knowledge, the only appearance-preserving simplification that does not require a meshing or parameterization, is the *Phong Splatting* [BSK04], following [KV03], which uses *surfels* [PZvBG00] (usually a point with a normal and a color) combined with normal mapping to render point sets with fewer points but similar appearance. Unfortunately, surfel splatting [ZPvBG01] is not well adapted to high resolution display, and requires complex multi-pass rendering, and intensive use of vertex/fragment shaders [BSK05]. Moreover, the technique proposed in [BSK04] to encode the normal distribution for each rendering primitive performs a strong low-pass filtering, and the intensive use of an underlaying kD-Tree makes it difficult to extend the technique to an out-of-core implementation.

In this paper, we propose a general pipeline for converting a large point set into a coarse polygonal mesh with high resolution normal maps, which is exactly the kind of representation suitable for real-time hardware supported visualization. We use as an input an unorganized set of samples where each sample is a 3D point and its associated normal vector (we will use indifferently "point" and "surfel" for such a sample). Our major goal is to avoid time-consuming steps such as surface reconstruction or precise parameterization, while keeping nice visual results. By introducing a new rendering primitive, called *Normal Surfel Strips*, we show that in-core appearance-preserving models can be created starting from huge point sets through a very fast out-of-core process. Our technique has been tested on various very detailed scanned objects and statues, for which an interactive visualization has been obtained with a global preprocessing time that represents only a couple of minutes.

2. Our Approach

2.1. Overview

Our algorithm can be decomposed in four steps (see Figure 1):

1. we perform an out-of-core simplification of the huge model (see Section 2.2)
2. the resulting simplified point set is quickly converted into hardware friendly rendering primitives, called *Surfel Strips*, organized in a bounding box hierarchy, named the *Stripping Tree*, following the fast lower dimensional meshing of [BRS05] (see Section 2.3).
3. the Stripping Tree is used during the out-of-core normal mapping; *all* the points of the original model are *streamed* through the tree and distributed to their corresponding leaves, where the point normal is projected onto a quad texture associated to each Surfel Strip (see Section 2.4). This streaming process is the key step of our technique, as it allows us to handle large models with limited memory. At the end of this step, each leaf of the Stripping Tree contains a low definition Surfel Strips and a high definition (possibly sparse) normal map.
4. a *normal map diffusion* fills the holes of each normal map by using diffusion algorithm, to get a continuous normal field, interpolating the original normals of the huge model (see Section 2.5).

The resulting object in each leaf (a coarse piece of mesh plus a high resolution normal map), is what we call *Normal Surfel Strips*. This collection of primitives, stored on the leaf of the tree, are directly used to provide a high quality interactive rendering by using conventional per-pixel shading.

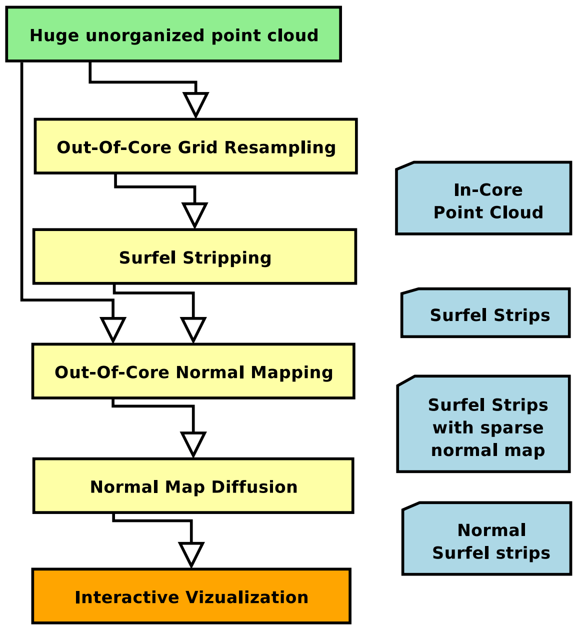

Huge unorganized point cloud
Out-Of-Core Grid Resampling
Surfel Stripping
Out-Of-Core Normal Mapping
Normal Map Diffusion
Interactive Vizualization
In-Core Point Cloud
Surfel Strips
Surfel Strips with sparse normal map
Normal Surfel strips

Figure 1: *Overview of our approach for interactive visualization of large models. The usual expensive step, the* meshing, *is only performed on a very reduced point cloud, and is no more the bottleneck. Most of the fine details are expressed through the normal maps, generated on a per-surfel strip basis with* diffusion, *a faster process than geometric reconstruction algorithms.*

2.2. Out-of-Core Simplification

Out-of-core simplifications of meshes are usually based on differential geometry properties, expressed as error metrics, which drive the polygon decimation. Such a criterion is not available on point clouds without some kind of surface reconstruction. On the other hand, state-of-the-art down-sampling methods [PGK02] for point-based surfaces are difficult to use with out-of-core models. Actually, a precise down-sampling is not mandatory in our particular case: we are not looking for the *n best* points to represent a given large model, we just need a reasonably-looking point cloud that can be quickly tessellated with the algorithm presented in Section 2.3. This algorithm can generate a fast local triangulation for point sets up to few millions of points in a reasonable time (typically about one minute for two millions of points).

But a final down-sampled point set of 2 millions points is not interesting enough to take full benefit of our fast appearance preserving approach (Section 2.4). What we need is to get very quickly a good looking coarse representation of the point cloud. Since the target objects of our application (large point clouds acquired with 3D scanners) are usually very dense (see models in Figures 5 and 10), we propose to use a simple grid to filter the point cloud. Similarly to Lindstrom's filtering for polygons [Lin00], each surfel of the model is read and tested against a 3D grid enclosing the whole model. If the intersecting grid cell is empty, then the surfel is simply stored in the cell where it becomes an accumulation surfel, and the cell cardinal is set to 1. If not, all the properties of the current surfel (position, normal, color, etc) are added to the accumulation surfel of the cell, and the cardinal is increased.

At the end, for all non empty cells, we divide all the properties of the accumulation surfel by the cardinal of the cell, and put the resulting surfel in the in-core surfel set. Note that for semi-automatic applications, the user may enter the desired grid resolution; we have observed good results when using $2^{\lfloor \log_{10}(n) \rfloor}$ where n is the total number of points. This simple grid filtering is extremely fast, as it processes more than 5 millions points per second.

2.3. Surfel Stripping

As our goal is to design a visualization system, watertight surface reconstruction is not mandatory for the coarse mesh generation. We propose to use an efficient lower dimensional triangulation, the *Surfel Striping*, that have been recently developed in [BRS05]. This technique converts the point set into a collection of overlapping triangle strips that offers a convincing smooth visualization, despite the lack of geometric continuity (see Figures 5 and 10).

First, the in-core point set is partitioned into small point sets, locally expressed as height maps, organized in an octree-like structure: the *Stripping Tree*. This is done by recursively splitting the bounding box of the decimated point

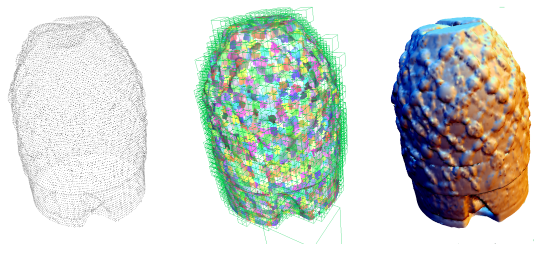

(a) (b) (c)

Figure 2: *Surfel Stripping. (a) The simplified point cloud obtained after the out-of-core simplification. (b) The local piece of meshes quickly generated thanks to the Stripping Tree partitioning (in green). Each colored patch corresponds to a surfel strip, locally generated in 2D. (c) The coarse mesh obtained, made of local triangle strips interpolating the input points.*

cloud, until having in each leaf i a set $S_i = \{s_{i0} ... s_{in}\}$ of surfels, that respects the following predicate:

$$\forall s_{ij} \in S_i \begin{cases} n_{ij}.n_i > \delta_a & \delta_a \in [0,1] \\ \frac{|(p_{ij}-c_i).n_i|}{\max_k(||p_{ik}-c_i||)} < \delta_d & \delta_d \in [0,1] \end{cases} \quad (1)$$

with p_{ij} the position of s_{ij}, n_{ij} its normal vector, n_i the average normal of S_i, which can be computed as the normalized eigen vector associated to smallest eigen value of the covariance matrix of $P_i = \{p_{i0}, ..., p_{in}\}$ (*Principal Component Analysis*) or simply by averaging the normals of S_i; c_i is the centroid of S_i. The δ_a value corresponds to the normal cone. In order to avoid distortion resulting from bent partitions, we set $\delta_a = 0.25$ in all our tests. The δ_d corresponds to the geometric displacement in the average normal direction. It can be set according to the sampling density if available, or by some heuristics. We have chosen in our experiments $\delta_d = 0.25$. To obtain regular leaves, we impose a maximum population threshold for the leaves, which may thus be subdivided, even if they are already consistent with a height map definition. The resulting partitioning is shown in Figure 2(b).

Before the generation of a local piece of surface and in order to avoid holes in-between leaves, an *inflation* pass is done on each leaf, by enlarging its associated point set with the points located in its volumetric one-neighborhood (i.e. all the points of the leaves adjacent to the current one). Actually, this is on this inflated surfel set S'_i that must be tested the predicate of Equation 1.

Then, a fast incremental 2D Delaunay triangulation of each inflated point set S'_i (leaves of the tree) is done in the average plane $\Pi'_i = \{c'_i, n'_i\}$ of each surfel set S'_i. In order to remove redundant triangles in overlapping zones introduced by the inflation process, a *decimation* pass is performed by comparing triangles of neighboring surfel strips. In practice, the decimation pass actually discards about 99% of the overlappings when using the simple rules described in [BRS05].

Finally, a fast stripping of the resulting local mesh is done using a cache-friendly adjacency graph [RBA05].

All those operations are performed at coarse level, on the decimated model and so require only a negligible processing time, even with a non optimized implementation.

When rendering the resulting collection of *Surfel Strips*, a visually continuous surface is obtained, almost equivalent to a provable watertight surface mesh in practical cases, but about one order of magnitude faster. Usually, residual overlapping does not produce visible artefacts in the shading, even under multiple light sources (see Figure 2). Each interior node of the stripping tree provides an average position, an average radius and a normal cone, in order to perform a hierarchical culling in a similar fashion to [RL00].

We have used this triangulation method because it naturally proposes a quad of projection for the normal map construction on a per surfel strip basis, simply by reusing the average plane Π'_i used for performing the Delaunay triangulation.

2.4. Streaming Normals

At this point, the Stripping Tree of the down-sampled surfel point set is available, and can be visualized as a coarse representation of the huge model (see Figure 2). In order to retrieve the original appearance of the large model, a *normal map* will be associated to each Surfel Strip. These normal maps are generated during a *normal streaming* process, where all points of the initial huge point cloud are streamed through the Stripping Tree, to quickly find the set of Surfels Strips they belong to. This second reading pass of the model requires only to deal with one of these points at the same time in the main memory. Note that one point may belong to more than one Surfel Strip, because of the *inflate-and-decimate* pass described above.

Then, for each intersected leaf, the local parameterization of the point relative to its Surfel Strip is computed by projecting the point on the average plane Π'_i of the strip. Actually, we parameterize the projected point according to a bounding quad, including P'_i, and aligned to the two eigen vectors associated to the two highest eigen values of the covariance matrix of P'_i, previously computed and stored as leaf data with n'_i and c'_i.

This parameterization is used to fill the relative pixel value of the associated normal map with the normal vector of the streamed point. We use floating point textures, so if more than one normal is projected onto the same pixel, we just add the normal vector value to the existing pixel, and normalize all the normal maps after having processed all the points of the original model. This also prevents from aliasing artefacts that may occur.

The resolution of each normal map is proportional to the number of points of its corresponding Surfel Strip, but can also be specified by the user as a "global quality" parameter, or restricted to the amount of available GPU texture memory for very large models or less competitive graphics cards. Similarly, its aspect ratio is equivalent to the bounding rectangle of its Surfel Strip. Using a flat parameterization of a non-flat Surfel Strip may generate some distortions, especially in areas of high curvature that would result in a global loss of details. But in practice, the constraints imposed during the construction of the stripping tree lead to close to planar Surfel Strips, which limit distortion. No artefacts were visible in our experiments. Since the normal maps are generated on a quad basis (aligned to the surfel strip), they are easy to pack in few large textures, an optimized way to store textures on the GPU memory.

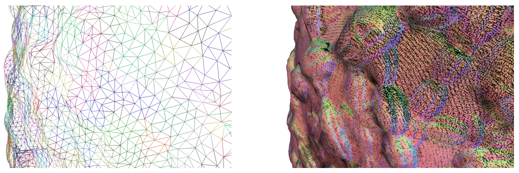

(a) Surfel strips (b) Sparse normal map

Figure 3: *After the normal streaming step, a sparse normal map is attached to each Surfel Strip. (a) Coarse topology computed from the sub-sampled point cloud. (b) Color visualization of the spare normal map: pixels color is set with the XYZ coordinates of the normals. Black points corresponds to pixels of the normal map where no surfel as been projected.*

2.5. Normal Map Diffusion

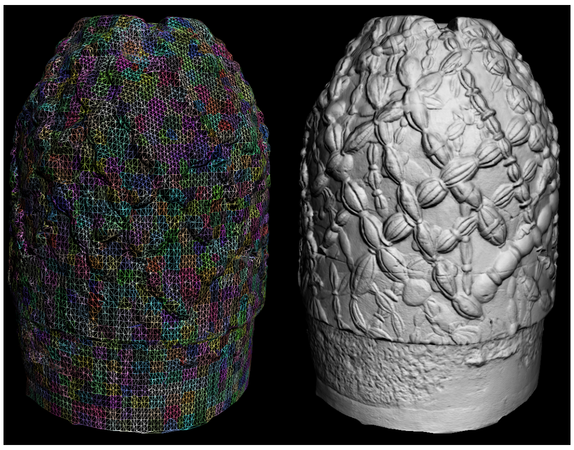
After the normal streaming process, each surfel strip is enriched with a sparse normal map since several pixels may not have been filled by projected normals (as shown in Figure 3). For using this map as a texture for our coarse surfel strips, holes need to be filled (black pixels in Figure 3). Many approaches have been developed over the years to fill holes in an image, which is a basic operation for image repairing. Exploration-based approaches such as [BWG03] directly compute an illumination value for a pixel given by exploring its neighborhood. On the other hand, iterative PDE-based approaches such as [PGB03], spread existing color in the image using PDEs such as Poisson equation, or diffusion equation. We use the PDE-based diffusion technique presented in [XP98], for its guarantees of continuity and smoothness. The implementation is based on a multigrid resolution scheme that first solves the problem at a coarser resolution, and then uses this coarse result to initialize the algorithm at finer resolution:

```
**Solve (h,** $Ax_h = b$**)**
1. Pre-smoothing steps: $Ax = b$
2. Downsample: $x_{h-1} = Dx_h$
3. **Solve (h-1,** $Ax_{h-1} = b$**)**
4. Upsample: $x_h = Ux_{h-1}$
5. Post-smoothing steps: $Ax = b$
```

where $Ax = b$ corresponds to the matrix formulation of discrete diffusion equation with finite differences, and h corresponds to the quadtree level associated with the resolution of the processed image (see also the Push-Pull algorithm in [GGSC]).

The approach of [PG01] corresponds to a multigrid iteration with no pre-smoothing step, a single post-smoothing step and a specific down-sampling algorithm that only takes into account existing samples. We inspired from [PG01] by skipping the pre-smoothing, and only using existing samples for down-sampling, but ran the post-smoothing iterations until the convergence criterion is met. Indeed, without these extra iterations, some blocky interpolations are present in the texture we obtained, especially around holes.

The multigrid resolution algorithm proved to be very efficient in practice (see Table 6), only a few iterations (e.g. 5) were needed for convergence with 10^{-3} error bounds in most cases. Note that the same approach can be used to create maps for other per-surfel attributes (e.g. color, geometric displacement, etc). The Figure 4 shows the resulting set of high resolution normal maps attached to coarse surfel strips. The hole-filling process provides normal maps that express the essential part of the original large model appearance. These maps are stored as texture in the GPU memory, and benefit from the **automatic filtering** provided by the hardware mip-mapping. This property is quite interesting, since it can be interpreted as both an anti-aliasing process and an hardware supported multiresolution rendering, thanks to the different levels of the mip-mapping.


(a)

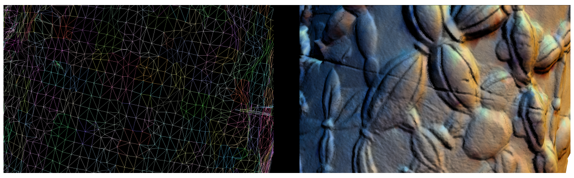
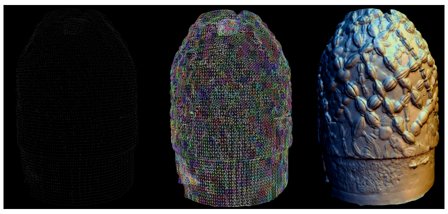

(b)

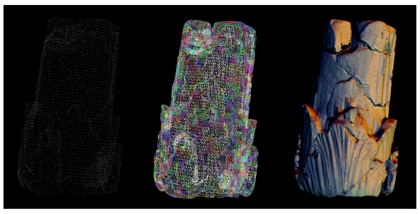
Figure 4: *Normal diffusion. (a) The Omphalos model (11 664 466 points). (b) Close-up. Left: coarse surfel strips quickly generated after the out-of-core decimation of the large point cloud (random per surfel strip color). Right: real-time rendering, with per-pixel illumination using the high-resolution normal maps. High quality rendering at high resolution, with minimal pre-process. Note the nice **automatic filtering** of the model, thanks to the intrinsic hardware mipmapping of the normal maps.*

3. Results

We have implemented our system under Linux on an Intel PIV 3.2 GHz, 1GB RAM, 160GB UDMA HD, NVidia Quadro FX 4400. We use C++ and the OpenGL Shading Language (for normal mapping). We consider input binary files where points are encoded as an unorganized list of chunks of 6 floats (3 for the position and 3 for the normal). Table 6 summarizes the preprocessing times of our system. Figure 5 shows the real-time rendering obtained on various point clouds with our approach. It appears that the normal map initialization is the main bottle-neck. Obviously, tree-traversal and local projections involved in this out-of-core streaming remain costly since they are performed on the whole model. Nevertheless, all the different stages involved in our approach are highly parallelizable (each point is treated separately), and can take benefit from recent dual-core CPUs (an improvement factor of 1.5 can reasonably be considered for dual-core CPUs, more for multi-CPU workstations). Note also that we use a pointer-based implementation of the Stripping Tree, which could be enhanced. Our resolution criteria for the normal maps works quiet well in most of the cases. Actually, even when a high density variation occurs inside a leaf of the tree, aliasing is prevented in the normal map thanks to the iterative diffusion step (see Figure 7). Note that the use of compression for textures could reduce the GPU memory footprint. The hard-drive latency strongly influences the performances of *simplification* and *normal mapping* passes. Better performances can be reached by using high-speed hard-drives (U-SCSI) and a dedicated workstation, where useless processes are stopped (usually between 20 and 30 on our Linux system). The excellent framerates given in Table 6 are reached thanks to the highly optimized polygonal hardware graphics pipeline, particularly adapted to display at high resolution polygonal models with high definition textures.

Comparison The critical point in our work was to reduce as much as possible the pre-process time needed for obtaining a convincing visualization of large point clouds. Compared to *QSplat* [RL00], our preprocessing is faster (on order of magnitude in the worst experimental case) and it does not require a previous surface reconstruction (huge additionnal processing time). Compared to the *Layered Point Clouds* [GM04], our preprocessing is about ten times faster. Of course, these multiresolution methods are conservative, and do not perform a low pass filtering on the geometry


(a) Omphalos (11 664 466 points)


(b) Drum 2 (22 877 845 points)

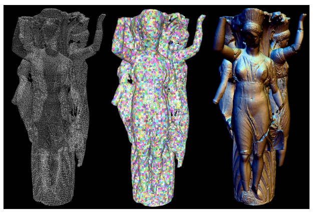

(c) Dancers (31 620 449 points)

Figure 5: *Visual quality for various large models. Antialiased rendering with 3 color light sources on a 1600x1200 screen resolution. From left to right in each image: the sub-sampled point cloud decimated at the first out-of-core reading pass, the coarse piece of mesh locally generated, colored with random per-surfel strip colors and the interactive rendering of this collection of piece of meshes, enhanced with normal map expressing the fine details, generated during the second reading pass of the point cloud.*

Models	Omphalos	Drum 2	Dancers	St Matthew
Number of points	11 664 466	22 877 845	31 620 449	186 810 938
TIMINGS				
Simplification time	5 s	10 s	14 s	61 s
Surfel Stripping time	2 s	4 s	5 s	7 s
Normal mapping time	45 s	151 s	213 s	667 s
Diffusion time	35 s	35 s	34 s	152 s
Total time	100 s	201 s	274 s	887 s
RENDERING				
Number of surfel Strips	1602	1721	2013	2457
Number of triangles	45 504	51 012	66780	79030
Textures memory used in MB	68	71	94	185
FPS (frames per second)	> 200	> 200	198	165

Figure 6: *Preprocessing time and rendering framerate for various large models. The total timing represents all the steps needed for the preprocessing, starting from an unorganized point cloud on disk up to a ready-to-render data structure in memory. The framerates are given for 1600x1200 screen resolution.*

such as ours, but from the visualization point of view, we keep the essential appearance thanks to high resolution normal maps reconstructed in 2D (see Figure 10). Note also that our system provides a **polygonal** rendering, highly optimized on today's GPU. This allows us to reach high framerates at high resolution. Actually, our approach provides results that confirm [PGK02]: performing decimation on the point cloud and then applying reconstruction methods is definitely more efficient than meshing and optimizing the full resolution point cloud, at least for our visualization purpose. Finally, the diffusion process of normal maps can be seen as a kind of surface reconstruction, where not the geometry, but the normal field is reconstructed from points, in the lower dimension (the average plane of the leaf node). Figure 8 shows our normal mapping reconstructed directly from original samples: the same order of visual quality is reached when comparing to prior art methods where a full resolution surface reconstruction and parameterization were necessary before performing the appearance preserving simplification.

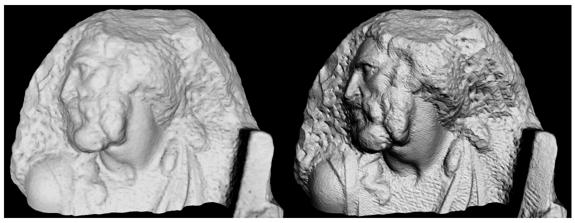


Figure 7: *Upper part of the St Matthew model with our method rendered at 165 FPS, without (left) and with (right) the normal maps. The maps are recreated* ***directly*** *from the point cloud, providing a convincing appearance, while using less than 80k triangles (left image). Most of the "appearance" information carried by the original point cloud is directly stored through these normal textures on the GPU memory (185 MB) and used for the per-pixel lighting.*

Limitations: We have made the choice to use a very simple Sub-sampling scheme at the beginning of our algorithm (see Section 2.2). The choice has been made after various experiments with real data set, which show essentially that

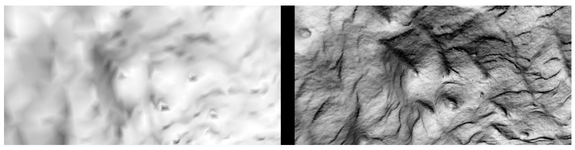


Figure 8: *Left: coarse surfel strips rendering. Right:* normal *surfel strips rendering. High frequency details of large scans models are preserved, using detailed normal textures instead of huge polygons sets. Globally, our approach provides similar results to usual appearance preserving methods that require full resolution tessellation, parameterization and simplification, while we deal only with the original samples.*

most of the time, large scans are dense enough to support this decimation, and so allow for this fast pre-process. Nevertheless, one could imagine situations where small topological features are lost during this first out-of-core sub-sampling. In this case, more adaptative decimation scheme must be used. Unfortunately, this usually means a much longer preprocess. The reader must also note that our approach is still a "simplification" one, which exhibits drawbacks and advantages. On one hand, even if most of the fine visual details are kept thanks to the high resolution normal maps (see Figure 9), a slight shrink effect can appear in silhouettes because of the coarse mesh definition. This is the price for reducing the time preprocessing and improving the rendering framerate compared to "multiresolution" approaches [RL00]. On the other hand, this low-pass filtering has frequently removed the *registration noise* present in our examples. Figure 9 shows the rendering of the St Matthew model with the publicly available QSplat software. We can observe that our method provides a globally equivalent appearance, with a much higher framerate, even under a strong close-up. Note that we have compared with QSplat because it is the only software publicly available for large dataset. Note also that QSplat is not tuned for recent graphics hardware, which explains the poor framerate obtained. One of our future work will be to compare our results with a combination of [DVS03] and [BSK05], which should be the stat-of-the-art point-based rendering system. In order to avoid the slight shrink effects, we have also plane to compute displacement maps on top of normal maps, using the dynamic GPU tessellation method proposed by Boubekeur and Schlick [BS05].

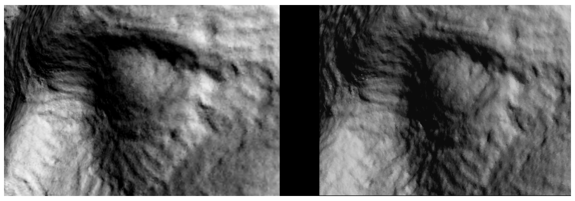


Figure 9: *Eye of the St Matthew. Visual quality comparison between our approach at 165 FPS (left) and the QSplat rendering (right), obtained at 0.3 FPS. Even under a strong close-up, our method keeps the fine details as well as the QSplat system, but with a much higher framerate.*

4. Conclusion

We have proposed an efficient pre-process to obtain an interactive visualization of large 3D objects represented by point clouds with appearance preserving. The main advantages of our technique are:

- Our system directly handles the unorganized point clouds, avoiding any kind of surface reconstruction of the large model.
- No complex data structure or complex processing is needed on the large model.
- Our local quad-based approach for appearance preserving does not require a globally consistent parameterization of the model.
- The pre-process is very fast as it basically only requires two out-of-core passes, which makes it usable in various applications such as large model data base browsing.
- The final in-core model is entirely stored on the GPU memory, large enough on today graphics devices to handle efficiently appearance attributes of hundred millions of samples, through, for instance, normal textures.
- Since all details are stored as normal maps, the rendering takes automatically benefit of the hardware mip-mapping for antialiasing details at a given resolution.

One weakness of the presented approach may be in the initial out-of-core grid down-sampling step, which clearly trades speed for quality. It only uses the volume of the object and not its surface, thus in very complex areas, some undersampling may appear which may generate visible artefacts. However, these artefacts occur very rarely, and do not degrade the overall appearance of the object. Of course some more accurate (but more expensive) down-sampling may always be employed as an alternative.

The whole pipeline is very easy to implement, and has provided very convincing results when applied on a large variety of acquired point clouds. We hope that it can become a good complement to existing high quality but slower visualization methods of large models, and that it will help, following [DDGM*], to use point-based surfaces for storing and transmitting huge sampled models such as scanned archaeological artefacts.

Acknowledgement : We are grateful to Electricit'e de France (EDF) and the Digital Michelangelo Project (Stanford) for the 3D data sets. This work as been partially done at UBC (Vancouver) and supported by LIGHT (INRIA associated team program).

References

[BRS05] Boubekeur T., Reuter P., Schlick C.: *Surfel Stripping*. Tech. rep., LaBRI - RR-1352-05 - (to appear in Proceedings of ACM Graphite 2005), 2005. 2, 3

[BS05] Boubekeur T., Schlick C.: Generic mesh refinement on gpu. In *ACM SIGGRAPH/Eurographics Graphics Hardware 2005* (2005). 7

[BSK04] Botsch M., Spernat M., Kobbelt L.: Phong splatting. *Symposium on Point Based Graphics 2004* (2004), 25–32. 2

[BSK05] Botsch M., Spernat M., Kobbelt L.: High quality splatting on today's gpu. *Symp. on Point Based Graphics* (2005), 25–32. 2, 7

[BWG03] Bala K., Walter B., Greenberg D. P.: Combining edges and points for interactive high-quality rendering. *ACM Trans. Graph. 22*, 3 (2003), 631–640. 4

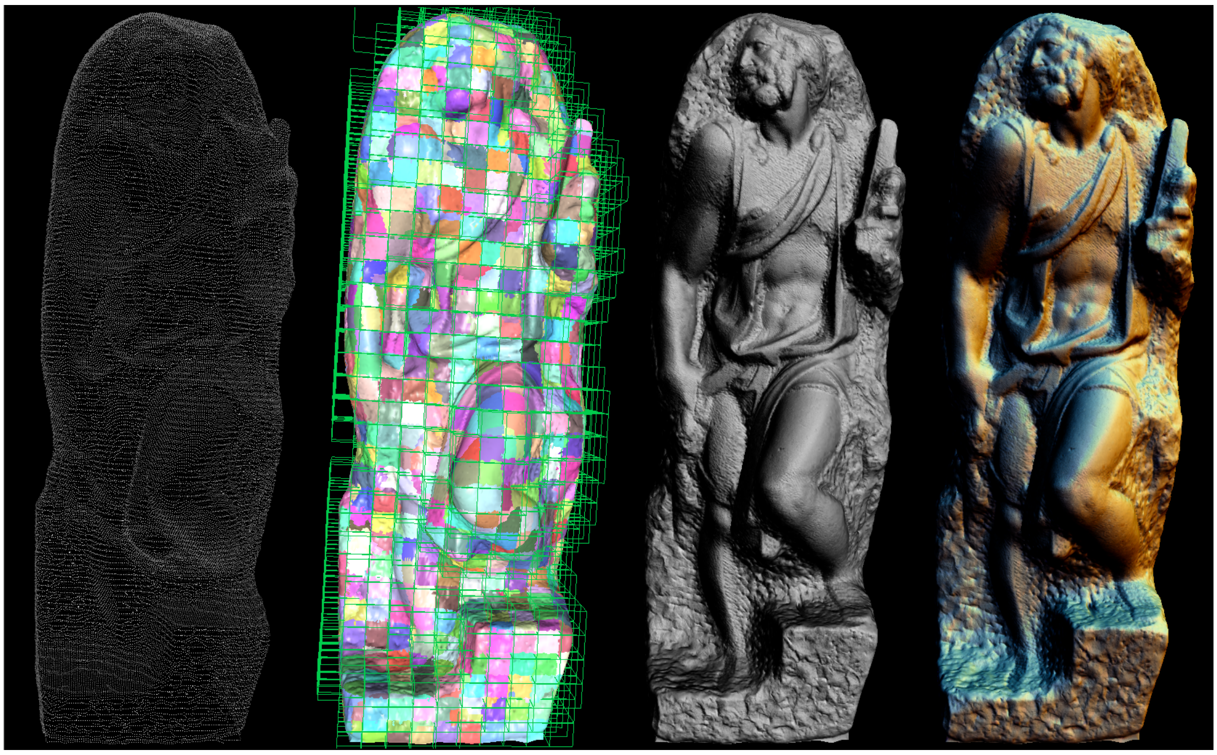


Figure 10: *Real-time visualization of the St Matthew model (186 810 938 points). From left to right: the sub-sampled model, the coarse mesh locally generated in the leaves of the stripping tree (in green), the coarse mesh with the high resolution normal mapping with one white light source and 3 colored light sources.*

[CGG*04] Cignoni P., Ganovelli F., Gobbetti E., Marton F., Ponchio F., Scopigno R.: Adaptive TetraPuzzles – efficient out-of-core construction of gigantic polygonal models. *ACM Trans. Graphics (SIGGRAPH 2004)* (2004). 1

[COM98] Cohen J., Olano M., Manocha D.: Appearance-preserving simplfication. *Proceedings of SIGGRAPH 98* (1998). 1

[DDGM*] Duguet F., Drettakis G., Girardeau-Montaut D., Martinez J.-L., Schmitt F.: A point-based approach for capture, display and illustration of very complex archeological artefacts. In *VAST 2004*. 1, 7

[DVS03] Dachsbacher C., Vogelgsang C., Stamminger M.: Sequential point trees. *ACM Trans. Graphics (SIGGRAPH 2003)* (2003), 657 – 662. 1, 7

[GBBK04] Guthe M., Borodin P., Balázs A., Klein R.: Real-time appearance preserving out-of-core rendering w/shadows. In *EGSR 2004*. 2004, pp. 69–79 + 409. 1

[GGSC] Gortler S., Grzeszczuk R., Szeliski R., Cohen M.: The lumigraph. In *Proc. of ACM SIGGRAPH 1996*, pp. 43–54. 5

[GM04] Gobbetti E., Marton F.: Layered point clouds. In *Eurographics Symposium on Point Based Graphics* (2004), Alexa M., Gross M., Pfister H.,, Rusinkiewicz S., (Eds.), pp. 113–120, 227. 2, 5

[Hop96] Hoppe H.: Progressive meshes. *Computer Graphics 30*, Annual Conference Series (1996), 99–108. 1

[KSW05] Krüger J., Schneider J., Westermann R.: Duodecim - a structure for point scan compression and rendering. In *Symp. on Point-Based Graphics* (2005). 2

[KV03] Kalaiah A., Varshney A.: Modeling and rendering of points with local geometry. *IEEE Trans. Visualization v9*, n1 (2003), 30–42. 2

[Lin00] Lindstrom P.: Out-of-core simplification of large polygonal models. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (2000), pp. 259–262. 3

[Lin03] Lindstrom P.: Out-of-core construction and visualization of multiresolution surfaces. In *Symposium on Interactive 3D graphics* (2003), pp. 93–102. 1

[LPC*00] Levoy M., Pulli K., Curless B., Rusinkiewicz S., Koller D., Pereira L., Ginzton M., Anderson S., Davis J., Ginsberg J., Shade J., Fulk D.: The digital michelangelo project : 3d scanning of large statues. In *Proc. SIGGRAPH 2000* (2000), ACM. 1, 2

[PG01] Pauly M., Gross M.: Spectral processing of point-sampled geometry. *ACM Trans. Graphics (SIGGRAPH 2001)* (2001), 379–386. 5

[PGB03] Perez P., Gangnet M., Blake A.: Poisson image editing. *ACM Trans. Graph. 22*, 3 (2003), 313–318. 4

[PGK02] Pauly M., Gross M., Kobbelt L. P.: Efficient simplification of point-sampled surfaces. In *Proc. of IEEE Visualization '02* (2002), pp. 163–170. 3, 6

[PZvBG00] Pfister H., Zwicker M., van Baar J., Gross M.: Surfels: Surface elements as rendering primitives. *ACM Trans. Graphics (SIGGRAPH 2000)* (2000), 335–342. 2

[RBA05] Reuter P., Behr J., Alexa M.: An improved adjacency data structure for fast triangle stripping. *Journal of Graphics Tools (JGT, to appear)* (2005). 4

[RL00] Rusinkiewicz S., Levoy M.: Qsplat: a multiresolution point rendering system for large meshes. *ACM Trans. Graphics (SIGGRAPH 2000)* (2000), 343–352. 1, 4, 5, 7

[SSGH01] Sander P. V., Snyder J., Gortler S. J., Hoppe H.: Texture mapping progressive meshes. In *Proceedings of SIGGRAPH '01* (2001), pp. 409–416. 2

[TCS03] Tarini M., Cignoni P., Scopigno R.: Visibility based methods and assessment for detail-recovery. In *Proc. of Visualization 2003* (2003). 2

[Tol99] Toledo S.: A survey of out-of-core algorithms in numerical linear algebra. 161–179. 1

[WDS04] Wald I., Dietrich A., Slusallek P.: An Interactive Out-of-Core Rendering Framework for Visualizing Massively Complex Models. In *Proceedings of the Eurographics Symposium on Rendering* (2004). 1

[XP98] Xu C., Prince J. L.: Snakes, shapes, and gradient vector flow. *IEEE Transactions on Image Processing 7*, 3 (March 1998). 4

[ZPvBG01] Zwicker M., Pfister H., van Baar J., Gross M.: Surface splatting. *ACM Trans. Graphics (SIGGRAPH 2001)* (2001), 371–378. 2