Visualization of Point-Based Surfaces with Locally Reconstructed Subdivision Surfaces

Tamy Boubekeur, Patrick Reuter, Christophe Schlick

▶ **To cite this version:**

Tamy Boubekeur, Patrick Reuter, Christophe Schlick. Visualization of Point-Based Surfaces with Locally Reconstructed Subdivision Surfaces. IEEE Shape Modeling International, Jun 2005, Boston, United States. inria-00260839

HAL Id: inria-00260839
https://inria.hal.science/inria-00260839

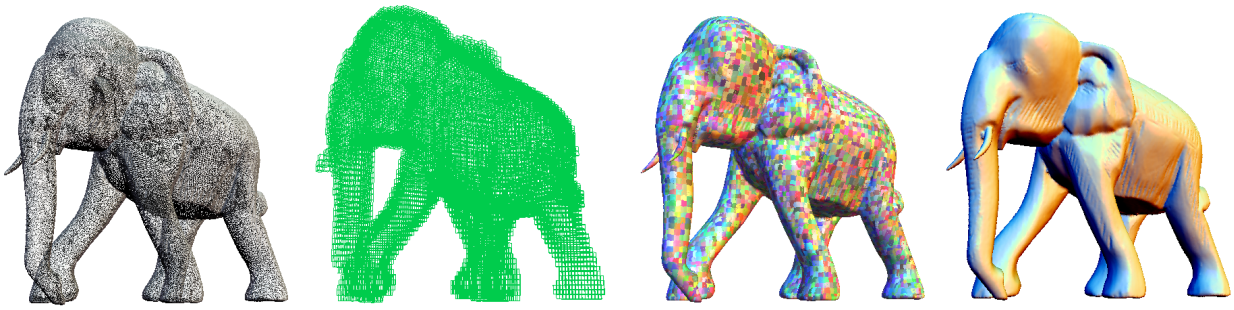
Submitted on 5 Mar 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Visualization of Point-Based Surfaces with Locally Reconstructed Subdivision Surfaces

Tamy Boubekeur[+] Patrick Reuter[*] Christophe Schlick[+]
[+]: LaBRI - INRIA - CNRS - University of Bordeaux I
[*]: LIPSI - ESTIA - CNRS
[Tamy.Boubekeur | Patrick.Reuter | Christophe.Schlick]@labri.fr



The original point-based surface and the three steps of our reconstruction technique. The final object is an aggregate of independent subdivision surfaces, that offers a convincing smooth visualization despite the lack of geometric continuity.

Abstract

Point-based surfaces (i.e. surfaces represented by discrete point sets which are either directly obtained by current 3D acquisition devices or converted from other surface representations) are well designed for multiresolution storage and transmission of complex objects. Unfortunately, visualization of point-based surfaces requires to develop specific rendering techniques (e.g. splatting) as point sets are not well adapted to existing graphics hardware which are optimized for polygonal meshes. In this paper, we propose an efficient reconstruction and visualization technique of point-based surfaces that takes full benefit from the whole optimized pipeline implemented in graphics hardware. The basic idea is to generate a set of independent meshes using a local 2D Delaunay triangulation of the point set. These meshes are then glued together to get a "visual continuity" by using a subdivision process.

1. Introduction

Point-based surfaces (PBS) can be seen as a geometric paradigm where 3D surfaces are only represented by their geometric component (i.e. a discrete point set, where the points may eventually be equipped with normal vectors) without any explicit representation of the topology. Many techniques to convert a PBS into a continuous surface representation (this process is called *surface reconstruction*) have been developed over the years. They can be divided into two main families: either explicit or implicit reconstruction techniques.

In the case of *explicit reconstruction*, the resulting surface is generated explicitly, usually as a polygonal mesh or a spline surface. One of the first techniques in that family was proposed by Hoppe et al. [18] [19] [17] where a three-step algorithm is used: an initial mesh is generated from a distance function constructed on the point set, followed by a mesh optimization step and a mesh subdivision step.

Explicit reconstruction can themselves be divided into two main categories. On one side, there are *dynamic approaches*, basically derived from the principle of deformable models developed in the field of computer vision [5] [20] where an energy criterion [12] is minimized by surface deformation. For instance, the *Intelligent Balloons* [11] are quiet good for their topological flexibility. Dynamic approaches usually generate high-quality meshes, but they are extremely slow and thus only applicable to PBS with a few thousands of points. On the other side, there are *static approaches*, mainly based on a Delau-

nay triangulation of the point set [7] [9]. The *PowerCrust* [2] and the *Cocone* [10] algorithms are based on a 3D Delaunay triangulation, while the techniques proposed by Gopi et al. [16][15] are based on a local projection operator. Linsen et al. [22] have also explored the idea of local triangulation through their *fan clouds*. Static approaches are much faster and scalable than dynamic ones. They have been successfully applied to PBS of up to few millions of points in reasonable computation time (e.g. 1 hour for 1M points for the Cocone).

In the case of *implicit reconstruction*, the resulting surface is defined implicitly as the zero-set of a function $f : \mathbb{R}^3 \rightarrow \mathbb{R}$. Here again, existing techniques can be divided into two main categories, both borrowed from the scattered data fitting literature [14]. In the first one, an implicit surface that interpolates the point set is computed as a linear combination of *Radial Basis Functions* (RBF). Initially limited to small point clouds [32], this approach has been extended to very large ones by using two different improvements: compactly supported basis functions [26] and hierarchical partitioning combined with *Partition of Unity* techniques [25] [28] [31] [33] [21]. The second implicit reconstruction approach is based on the *Moving Least Squares* (MLS) technique, that allows one to reconstruct a piecewise polynomial function that approximates the point set. MLS can be used either as a projection operator [1] or as an implicit fitting technique [29]. The main advantage of implicit reconstruction techniques is their robustness against non-uniformly sampled PBS. Their main drawback, as usual with implicit surfaces, is that they cannot be directly rendered with existing graphics hardware and thus require an additional tessellation step (e. g. *Marching Cubes* [24] [6]).

Discussion

In recent years, most work has focused on implicit reconstruction of PBS. The main reason for this is that the principle of *Partition of Unity* makes it possible to achieve a purely local reconstruction, which results in a $O(n)$ complexity, where n is the size of the point set. To our knowledge, this is currently not possible with an explicit reconstruction, which offers a $O(n \log n)$ complexity at best. So for very large point sets, implicit reconstruction techniques should definitely be faster than explicit ones. But as stated above, implicit surfaces have to be tessellated before being rendered by the graphics hardware. As many existing work has shown, this tessellation is not that straightforward. So the pros and cons of both approaches are not that easy to balance.

The work presented in this paper is based on the following postulate: *as the reconstruction of a PBS that guarantees geometric continuity cannot be achieved in linear complexity, loosening the constraints and seeking only for a "visual continuity", may reach such a linear complexity*.

It should be noted that point-based splatting techniques [35] [8] offer such a visual continuity by working in the image space. Unfortunately, splatting techniques require a specific modification of the graphics hardware pipeline to be efficient. Even if this modification is relatively easy with recent hardware (by using vertex or fragment shaders), splatting is still limited in terms of appearance (no environment mapping, no casted shadows, etc) and its performance quickly decreases when the resolution of the images is increased. We believe that object-space explicit reconstruction would definitely offer a better quality vs. cost tradeoff.

The approach that may be most closely related to ours is the patch generation proposed in [1] for the visualization of MLS surfaces. Their algorithm generates a collection of patches which are projected onto the MLS surface. Unfortunately, their process produces strong visible artefacts as the patches are restricted to a grid topology, and the boundaries do not blend correctly in overlapping zones.

The remainder of this paper is divided as follows: Section 2 presents a general overview of our algorithm while Sections 3 to 6 focus more deeply on the different steps involved (reconstruction, partitioning, subdivision, rendering). Section 7 presents some experimental results and Section 8 concludes by proposing some future research directions.

2. Overview of the Algorithm

As stated above, we are seeking for an explicit object-space reconstruction that should offer convincing visual continuity in linear time. One of the central ideas of our approach is to clearly separate the global topological extraction and the local geometric reconstruction of the surface. An overview of the algorithm is illustrated in Figure 1: after having space-partitioned an unstructured PBS, a local reconstruction, done at each cell of the partition, provides a set of disjoint 2-manifolds with boundaries, all of genus 0. Finally a visually continuous surface is obtained by an aggregation of these 2-manifolds.

Note that our current implementation requires a PBS where the points are all equipped with a normal vector. This is a quite usual assumption when working with PBS. When this normal vector information is not available, it is possible to generate it at each point with a *principal component analysis* of the neighborhood [18] [16].

To be able to account for large point clouds, a totally localized approach is chosen: all testing, sorting or selecting operations involved in the process will only be done on a

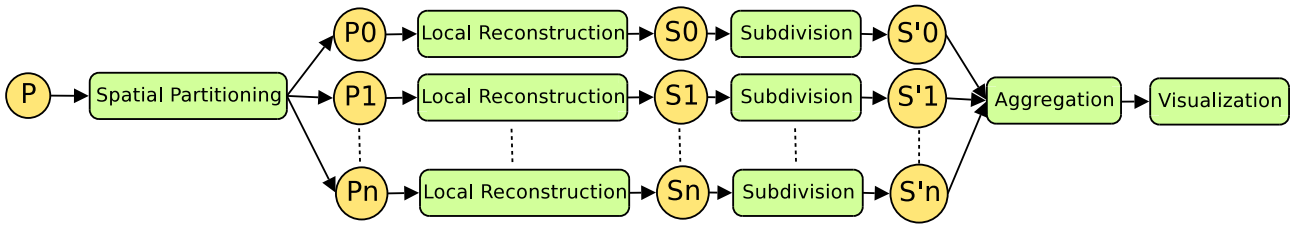


Figure 1: Overview of the algorithm.

reduced set of points. This principle would enable several extensions that we are currently studying: massively parallel implementation of the algorithm, out-of-core reconstruction [30] or lazy reconstruction based on different criteria (see Section 6). Our algorithm can be decomposed in three steps:

1. Partitioning of the PBS with an octree (Section 4).
2. Local piecewise mesh reconstruction of each partition (Section 3), temporarily extended to points of its neighborhood. The goal is to discard holes by creating *overlapping zones* between neighboring surfaces.
3. Subdivision of the piecewise meshes that are locally aggregated to increase the visual quality (Section 5).

Before detailing more deeply these steps in the next three sections, let us summarize the notations used in the remainder of the paper:

P	the set of input points
P_i	the ith partition of this set $(\bigcup_i P_i = P)$
C_i	the cell associated to the partition P_i
r_i	the radius of the cell C_i
k_i	the number of points in the cell C_i
S_i	the locally reconstructed surface on P_i
V_i	the set of points lying in neighboring cells of C_i
p_{ij}	the jth point of the partition P_i
n_{ij}	the normal vector of the point p_{ij}
c_i	the centroid of the partition P_i
n_i	the average normal vector of the partition P_i
$u.v$	the scalar product between two vectors u and v

3. Reconstruction

3.1. Local Surface Reconstruction

The kernel step of our algorithm is the local reconstruction process in the cells of the octree. For clarity reasons, the initial partitioning step will only be explained in Section 4, because it strongly relies on the way the local reconstruction is done. The goals that we seek for this local reconstruction are the following:

- The process should take a small unorganized point set P_i and produce a triangular mesh S_i;
- This reconstructed mesh S_i should interpolate or, at least, be a good approximation of P_i;
- As S_i will be used as a coarse mesh for subdivision, it should fulfil usual quality criteria for subdivision (namely constraints on vertice degrees [34]).

The generation of the triangular mesh is based on the *Delaunay triangulation*, which has mainly be chosen for its nice properties (quite regular degrees for the vertices, maximization of the minimal angle between two edges, etc). In the case of a 3D data set, one of the main reproach usually made against Delaunay triangulation is its computational cost: indeed, after having generated a 3-manifold made of tetrahedrons interpolating the data set (i.e. Delaunay tetrahedrization), a 3D Delaunay triangulation requires to select only the triangles that are localized on the surface. This selection is not only difficult but also implies to reject most of the data computed during the first step.

Actually, as our data set is a PBS, we know that all points lie on a given surface: in other words, if we are sufficiently close to the surface, the problem can be considered as a 2D reconstruction one. In a reasonably small neighborhood, the surface can be considered as a *local height map*: each point $\{x,y,z\}$ can be expressed as a height function $z = f(x,y)$ according to some local average plane. If this assumption is fulfilled (see below), we can indeed use a 2D triangulation to reconstruct the topology between points. This will generate a 2-manifold made of triangles, interpolating the point cloud.

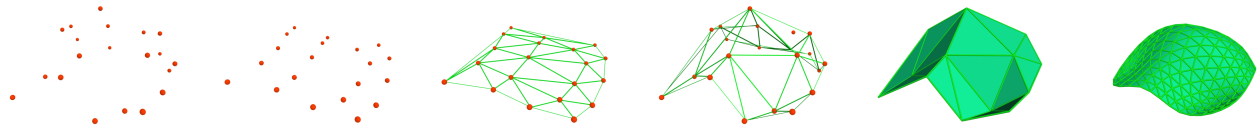
So all we need is a predicate κ that ensures us that the local point set P_i can be expressed as a height map. We consider that $\kappa(P_i)$ is true when:

$$\forall j \in [0, k_i - 1],\ n_{ij}.n_i > \delta_a,\ \delta_a \in [0,1]$$

and

$$\frac{|(p_{ij} - c_i).n_i|}{\max_k(||p_{ik} - c_i||)} < \delta_d,\ \delta_d \in [0,1]$$

The parameter δ_a represents the maximal deviation angle that the normal of a point can make according to the aver-



(a) (b) (c) (d) (e) (f)

Figure 2: Local surface reconstruction. (a) Initial partition P_i. (b) Projection onto Π_i. (c) 2D reconstruction by Delaunay triangulation. (d) 3D projection of the set of generated triangles. (e) Reconstructed surface S_i before subdivision. (f) Smooth surface obtained by subdivision of S_i.

age normal of its current cell, and must be greater than 0 for a height map (no folding). Our experiments show that $\delta_a = 0.2$ provides good results in general. Similarly, the parameter δ_d (the distance between a point and its projection onto the local average plane defined by c_i and n_i) helps to perform the partitioning according to the geometry properties of the PBS, and can range from 0 (all the points are in the same plane) to 1. We have set δ_d to 0.2 in our experiments. This value can also be seen as the approximation precision imposed to the average plane of a partition.

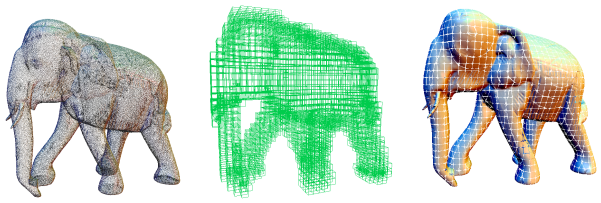
Both δ_a and δ_d are intuitive enough for being interactively set by the user who can quickly tune the partitioning (see Section 4) according to the specific characteristics of the PBS. Note that this approach is only sub-optimal, because P_i is only determined by the partitioning step and thus does not maximize the area of height surfaces. We mainly choose this formulation for its speed and low implementation cost. One of the main advantages to have a double criteria, acting simultaneously on point distribution and normal distribution, is that a whole set of "difficult" cases are correctly detected (see Section 7).

When $\kappa(P_i)$ is true for all i, the reconstruction can be totally achieved in 2D, and a piecewise 2D Delaunay triangulation will generate a collection of 2D manifolds. Note that Gopi et al. [16] have explored a similar idea of lower dimensional reconstruction, but they considered only isolated points. We propose a more general approach working in a whole local cell C_i provided by the partitioning step. Here is the algorithm used for each cell C_i:

1. Compute the average plane Π_i defined by the centroid c_i of P_i and the average normal vector n_i
2. Project each point of P_i onto Π_i;
3. Compute a 2D Delaunay triangulation on Π_i;
4. Re-project the generated triangles in the 3D space by the inverse transformation of Step 2.

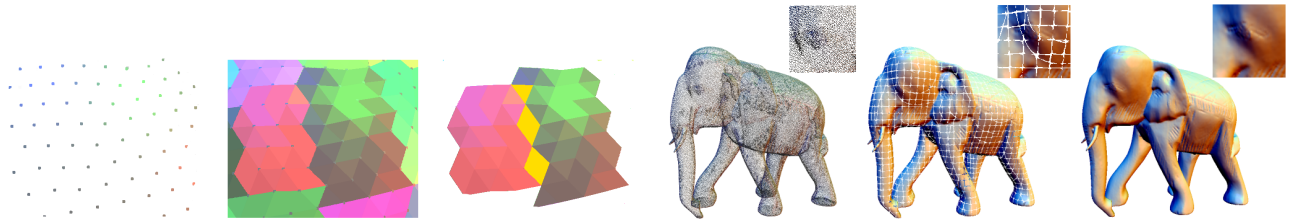
Figure 2 shows the different steps of this algorithm, including the final subdivision step that will be discussed in Section 5. The reconstructed mesh interpolates P_i in each cell C_i, and offers a good distribution of nice looking triangles, thanks to the properties of the 2D Delaunay triangulation. The number of extraordinary vertices is low, and no strong singularities will be developed after the application of some subdivision passes. Of course, some deformations will occur during the 3D reprojection step, but our intensive testing has shown that the algorithm behaves robustly for a large variety of models (see Section 7).

Let us give some additional comments about this algorithm. First, the computation of the average plane is extremely fast when all points are equipped with normal vectors. As said above, if the normal vector information is missing, it can be retrieved at a reasonable cost by performing a *principal component analysis* on P_i, see [18] for more details. Second, our projection is similar to the one proposed by Gopi et al. [16], but we consider a common local plane for the whole partition P_i that belongs to the cell C_i, which is much more efficient. Finally, we choose to implement an incremental Delaunay triangulation. This is not the optimal choice (in general, the Fortune's algorithm would be better) but as the local sets P_i are small, there is no strong difference. On the other hand, incremental Delaunay is interesting as we aim to deal with progressive transmission of PBS, such as raw data coming from the 3D range scanner.



(a) (b) (c)

Figure 3: Reconstruction in each cell. (a) The initial point-based surface. (b) The cells of the space partition (in our case, the leaves of an octree). (c) A piecewise mesh independently reconstructed in each cell.



(a) (b)

Figure 4: Hole filling through overlapping. (a) In yellow, the overlapping zone between the two neighboring surfaces. (b) From left to right: the original point-based surface, the aggregation of generated surfaces respectively without and with overlapping. Even under a strong close-up, the visual continuity is maintained.

3.2. Overlapping

To ensure a visual continuity, we propose to create an overlapping between neighboring surfaces. This can simply be done by applying the local reconstruction algorithm on partitions that are temporarily enlarged to their neighborhood (see Figure 4(a)). More precisely, each cell is enlarged by a factor β to include points belonging to neighboring cells. The value of β can be set interactively by the user or can be deduced from the overall density of the PBS. If the PBS has a known γ-density (i.e. γ is the maximal value such that each ball of radius γ, centered at each point of P_i contains no other point), we just have to enlarge the support with $\beta = \frac{2\gamma}{r_i} + 1$ for ensuring a right overlapping. Unfortunately, this density information is not always available. In that case, we may use the following value of β on the cell size: in each cell C_i, the local reconstruction operator is applied to $Q_i = P_i \cup R_i$ where R_i is the set defined by: $p \in R_i \Longleftrightarrow p \in V_i$ and $||p - c_i|| < \beta.r_i$. Experimental results have shown that $\beta = 1.3$ offers good results on our whole set of various models. Note that we need to preserve the height map property during the enlarging process, and thus we have to test the validity of the predicate $\kappa(Q_i)$.

The local reconstruction operator is interpolating which means that the local meshes reconstructed in neighboring cells share the points that belong to the overlapping zone. As the Delaunay triangulation always generates a locally optimal triangulation, very often the overlapping zone shared by two neighboring cells is triangulated exactly in the same way for both cells, which offers a perfect correspondence of the generated triangles. Sometimes, the overlapping is only approximate but even in that case, there is no strong degeneration which means that a visually continuous feeling is always provided during the rendering, even under a strong close-up (see Figure 4(b)).

3.3. Study of the Complexity

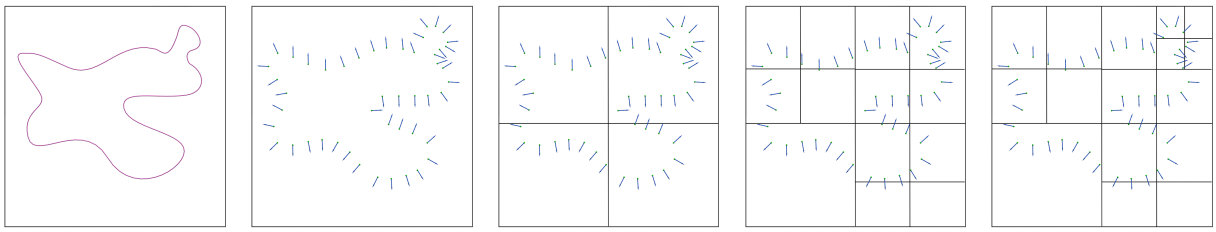
The cell enlarging mechanism used by our algorithm implies that the total number of points n used for the triangulation is higher than the number of input points (about 25%). The memory complexity of the Delaunay triangulation in dimension d is $\Theta(n^{\lceil \frac{d}{2} \rceil})$, whereas it is $\Theta(n^{1.8})$ for point surfaces [3]. In our case, as the problem is solved in 2D, the memory complexity is linear compared to the number of points.

Each point added to the triangulation needs to be tested toward the whole set of generated triangles. Thanks to a randomized incremental triangulation, the computational complexity can be reduced to $\Theta(n \log n)$ when inserting n points into the triangulation. In our approach, the n points of the initial point set are distributed on m octree leaves with $m << n$. If we raise the number of cell points to an arbitrary constant value k, the total number of leaves will be $m = \lceil n/k \rceil$ cells. The computational complexity for a cell can then be bounded by $\Theta(k \log k)$, which is a constant. So the complexity of the complete triangulation is about $m\Theta(k \log k)$ which is equivalent to $\Theta(n)$. To be exhaustive, we also have to include the complexity of projection and re-projection operators, but both of them are also linear. So, as expected by our initial postulate, the global complexity of our reconstruction technique is linear in the number of points, both for the storage and the computation. Note that the partitioning time, in $\Theta(n \log n)$, has no strong influence in pratical cases (see Figure 9).

4. Partitioning

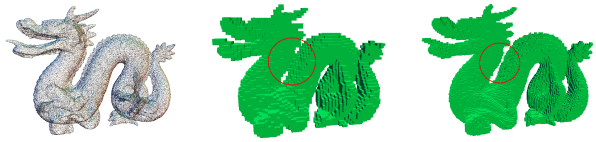
4.1. Partitioning structure

Our local approach imposes a partitioning structure that approximates the global topology to allow the enlarging process of the partitions for the overlapping. Among possible



(a) (b) (c) (d) (e)

Figure 6: A 2D example of partitioning (i.e. quadtree instead of octree) according to the height map criterion κ. (a) The original shape. (b) The corresponding point cloud with normals. (c,d,e) Successive refinement steps: each cell C_i whose associated point cloud P_i does not satisfy κ is subdivided.



(a) The dragon model with 100 251 points (b) Depth = 5 (c) Depth = 6

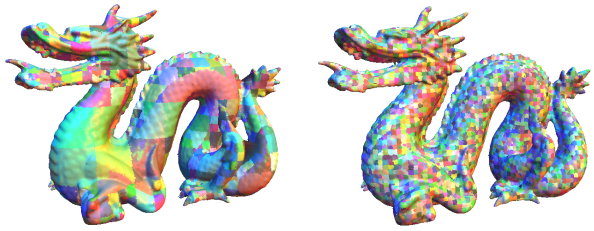
Figure 5: We can see a topological error into the red circle. On the right, we have increased the maximal depth, and the topology is correctly retrieved (the corresponding octree is homeomorphic to the sampled object).

candidates, we have chosen the *octree* for its hierarchical multiresolution structure with cells of regular shapes, and for the geometrically adaptive approximation that it provides for the topology of the point cloud. In particular, the genus of the surface will be easily retrieved (see Figure 5), which simplifies the local reconstruction step. Moreover, as we will see, one drawback usually pointed out with the octree, the affine dependance of the generated partitioning (i.e. the partition is different according to the position and the orientation of the initial bounding box) can be dramatically reduced in our case (see below).

4.2. Partitioning by octree

With the help of the $\kappa(P)$ predicate defined in Section 3.1, the partitioning step becomes straightforward. We begin with C, the bounding cube of P. If $\kappa(P)$ is false, then we cut C in 8 cells (8 equal cubes). The initial point set P is splitted according to the cell boundaries, and we run again the partitioning algorithm on each cell. The Figure 6 illustrates this principle in 2 dimensions.

In order to preserve a good balance between the computational cost of all the local triangulations, a cell C_i that satisfy κ will nevertheless be subdivided if k_i exceeds a given threshold value k (we use $k = 40$ in our implementation). In addition to balancing the computation cost, we have noted that a small maximum number of points by cell also decreases the affine dependance of the partitioning as it also balances the size of the leaves (see Figure 7).



(a) (b)

Figure 7: Partitioning criteria. (a) The piecewise mesh generated with a partitioning step only based on the κ criterion. (b) Here, we have also included a maximum number of points by cell criterion.

4.3. Discrete topology

To achieve an overlapping between locally reconstructed surfaces that respects the global topology of the object, we use the volumetric neighborhood provided by the octree. As the leaves of the octree are exclusively made of cubes, the neighborhood searching can be done with a very simple algorithm:

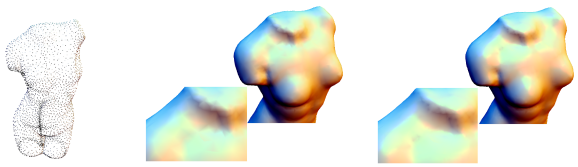
Let A and B be two cells of the octree O, let r_A and u_A (respectively r_B and u_B) be the edge length and the center of A (respectively B). Let r_{min} be the edge length of the smallest cell of O and $\zeta_X(u)$ be the Euclidean norm of the projection of u onto the axis X, ie. the absolute value of its

X-coordinate. Then, $A \in V_B$ if and only if:

$$\max(\zeta_X(u_B - u_A), \zeta_Y(u_B - u_A), \zeta_Z(u_B - u_A)) - r_{min} < \frac{r_A + r_B}{2}.$$

In words, it means that the surface is localized in the volumetric *layer* approximated by the *leaves* of the octree, this layer represents a good approximation of the neighborhood onto the surface. By using this algorithm, computing the neighborhood of the whole set of cells for a point cloud made of 450 000 points needs less than a second on a standard workstation. Such a speed allows the user to interactively tune the partitioning parameters if the topology does not seem to be correctly retrieved.

5. Subdivision



(a) (b) (c)

Figure 8: Subdivision on the collection of surfaces. (a) Initial point-based surface. (b) After a local overlapping reconstruction. (c) After two passes of Loop subdivision.

After the partitioning (Section 4) and the local triangulation (Section 3), we have a collection of 2-manifolds that are disjoint but overlap. Each manifold is defined by a triangular mesh made of about forty triangles. The next step of our algorithm is the application of a subdivision scheme on each surface S_i. After having tested several classical schemes, it appeared that the Loop subdivision scheme [23] [34] achieves the best results for our purpose: it ensures a limit surface that is C^2 almost everywhere (C^1 only at extraordinary vertices) which offers a nice noise filtering feature, it is able to reproduce sharp edges if required [17] [4]. In our algorithm, the interest of using subdivision is double. First, a few subdivision passes applied on the set of overlapping surfaces S_i will increase the final visual quality by converging all S_i to a smooth surface (see Figure 8). Second, the subdivision step makes it possible to better glue together overlapping surfaces S_i. Let us detail this process: S_i is constructed as a good local approximation of the underlying surface in the cell C_i, but not necessarily in its neighborhood V_i (i.e. in the overlapping zones between S_i and its neighboring surfaces). Let S_i^R be the part of S_i localized outside of C_i (overlapping zones). To glue S_i^R on S_j, we simply propose to project each point p_{jk} of S_i^R generated during the subdivision on the surface S_j. An efficient projection is to compute the intersection between S_j and the ray starting from p_{jk} along its normal vector n_{jk}. This solution does not guarantee to find an intersection for each point p_{jk} but is sufficient in practice. A more robust projection operator could be developed, based for instance on the MLS surface defined by the points of an overlapping zone, but it will obviously be much more slower.

6. Rendering

6.1. Hardware Rendering

Like all explicit reconstruction techniques, our algorithm generates polygonal surfaces which can be directly handled by the graphics hardware pipeline. Moreover, the octree partitioning provided for the resulting surface aggregate can be used to speed up the rendering by improving occlusion detection algorithms [13]. Our current implementation is based on the OpenGL API and offers an interactive visualization of point-based surfaces that takes full benefit from the whole optimized pipeline implemented in graphics hardware. The Figure 10(a) shows an interactive rendering of the Stanford dragon using a conventional Gouraud shading with 3 colored point light sources. The Figure 10(c) shows a bottle rendered with environment mapping. Note that even if the bottle is very non-uniformly sampled, the reconstruction is very smooth and does not show any artefact in the specular reflection.

6.2. Ray-Tracing

Ray-tracing is another common way to obtain high-quality rendering. A straightforward approach could be to perform the surface reconstruction with our algorithm, and then to submit the resulting object to the ray-tracing engine. However, there is no doubt that for an image, or even an animated sequence, a large part of the surface is likely to stay hidden, so reconstructing these hidden parts is of no use. We propose here to perform an *lazy* reconstruction that takes benefit from our local approach. The idea is to achieve a tree pruning and to reconstruct the surface only in the leaves of the octree that are actually intersected by the rays. With such an lazy reconstruction, the smaller the leaves, the better the pruning, so the optimization is particulary efficient for large point-based surfaces. Our current implementation provides a plugin to the well-known *Povray* ray-tracing engine [27]. The Figure 10(b) presents the Stanford dragon rendered with that plugin.

7. Results

Table 9 summarizes some reconstruction times obtained for a various set of point-based surfaces, with sizes going from 3k to 530k points. The corresponding pictures can be found in Figure 11. The visual quality of the rendering is totally equivalent to the quality obtained by existing explicit or implicit reconstruction techniques, but the computation times are reduced by one or two orders of magnitude according to the experimental results given by the different authors. We have done an exhaustive comparison with the implicit reconstruction technique proposed in [25], which is known to be one of the fastest and for which the source code can be downloaded on the web. Compared to their technique, we observe an acceleration factor from 2 to 10, depending on the model, for an equivalent visualization.

As expected, the main problem generated by our method may occur in the overlapping zones. It is vital for the sampling to be dense enough in high curvature zones (actually, this is more a sampling problem than a reconstruction one) because the projection operator used for the 2D triangulation may degenerate. Fortunately, the problem can often be solved by allowing an additional subdivision of the octree cell. As mentioned above, our partitioning is fast enough to tune the intuitive parameters of the partitioning according to the features of the object. The Figure 10(d) shows the evolution of computation time according to the number of points. We used the Stanford dragon at different resolutions that were obtained by down-sampling the original high resolution model. As expected, the curve presents a linear behavior when the size of the model is increased.

Model Name	Model size	Partitioning	Rec.
Torso	2 941 pts	0.01 sec	0.237 sec
Bottle	42 736 pts	0.02 sec	2.13 sec
AdamKraft	70 073 pts	0.45 sec	4.12 sec
Pipe	71 481 pts	0.09 sec	3.51 sec
Dragon	100 251 pts	0.20 sec	4.83 sec
Elefant	148 689 pts	0.16 sec	7.63 sec
Buddha	144 648 pts	0.22 sec	8.64 sec
Dragon	437 646 pts	0.91 sec	23.47 sec
Buddha	530 000 pts	1.01 sec	24.01 sec

Figure 9: Timing (in seconds) for partitioning and reconstruction of several PBS models (Intel P4, 3.4GHz, 1.5Go RAM)

We have also tested the variation of the computation time, according to the threshold value k (i.e. maximum number of points allowed per cell) used during the partitioning step. The Figure 10(e) shows that good result are kept, between about 5 and 40 points by cell. This test has been done with the Stanford dragon model with 437 646 points.

8. Conclusion and Future Work

The present work has enlightened several aspects of point-based surface reconstruction:

- An explicit object-space reconstruction technique with linear complexity, both in storage and computation time, can be developed by using a totally local approach.
- Ensuring a unique 2-manifold for the whole surface is not necessary when dealing only with rendering purposes: a surface may offer a *visual continuity* (even under a strong close-up on the surface) while there is no true geometric continuity.
- Subdivision techniques can help to obtain such a visual continuity by smoothly gluing together independent overlapping meshes.

The technique proposed here should be considered as a complement to other reconstruction techniques that offer guaranteed geometric continuity at the price of much longer computation times. Due to its linear complexity, our algorithm is particulary well-adapted to large point clouds. Moreover, the performance of our system allows the user to interactively tune the parameters in some difficult cases, but we have noted that the automatic settings offered by our current implementation works well for an extremely large set of models. One nice property of explicit reconstruction is that it avoids any modification of the standard hardware pipeline which is not the case with *splatting* approaches. The main weakness of our technique is the difficulty to resolve the topology for very non-uniformly sampled point-based surfaces. In such a case, implicit approaches definitely provide better results.

The results provided by our first implementation are encouraging and offer some interesting directions for future work:

- **Use of multiresolution:** Modification of the algorithm to account for progressive transmission of PBS.
- **Use of locality:** Modification of the algorithm to allow out-of-core reconstruction for very huge point sets, this may later lead to an automatically load-balanced implementation on a PC cluster.
- **Up-sampling/Down-sampling:** Modification of the local reconstruction to enable automatic down-sampling during triangulation and automatic up-sampling during sudivision.

- **Improved subdivision scheme:** Convert the subdivision gluing phase into a true single manifold fitting.

References

[1] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. Point set surfaces. *IEEE Visualization 2001*, pages 21–28, October 2001. ISBN 0-7803-7200-x.

[2] N. Amenta, S. Choi, and R. K. Kolluri. The power crust. In *Proceedings of the sixth ACM symposium on Solid modeling and applications*, pages 249–266. ACM, ACM Press, 2001.

[3] D. Attali and J.-D. Boissonnat. Complexity of the delaunay triangulation of points on polyhedral surfaces. Technical report, INRIA, 2001.

[4] H. Biermann, A. Levin, and D. Zorin. Piecewise smooth subdivision surfaces with normal control. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 113–120, 2000.

[5] A. Blake and M. Isard. *Active Contours*. Springer-Verlag, 1998.

[6] J. Bloomenthal. Polygonization of implicit surfaces. *Computer Aided Geometric Design*, 5:341–355, 1988.

[7] J.-D. Boissonnat and F. Cazals. Smooth surface reconstruction via natural neighbour interpolation of distance functions. In *Proceedings of the sixteenth annual symposium on Computational geometry*, 2000.

[8] M. Botsch, M. Spernat, and L. Kobbelt. Phong splatting. In *Symp. Point-Based Graphics*, 2004.

[9] D. Cohen-Steiner and F. Da. A greedy delaunay based surface reconstruction algorithm. Technical report, INRIA-Sophia Antipolis, 2002.

[10] T. K. Dey, J. Giesen, and J. Hudson. Delaunay based shape reconstruction from large data. In *Proceedings of the IEEE 2001 symposium on parallel and large-data visualization and graphics*, pages 19–27. IEEE, IEEE Press, 2001.

[11] Y. Duan and H. Qin. Intelligent balloon. In *Proceedings of the sixth ACM symposium on Solid modeling and applications*, 2001.

[12] Y. Duan, L. Yang, H. Qin, and D. Samaras. Shape reconstruction from 3d and 2d data using pde-based deformable surfaces, 2004.

[13] F. Durand. A multidisciplinary survey of visibility. ACM Siggraph course notes Visibility, Problems, Techniques, and Applications, July 2000.

[14] R. Franke and G. Nielson. Smooth interpolation of large sets of scattered data. *International Journal for Numerical Methods in Engineering*, 15(11):1691–1704, 1980.

[15] M. Gopi and S. Krishnan. A fast and efficient projection based approach for surface reconstruction, 2000.

[16] M. Gopi, S. Krishnan, and C. Silva. Surface reconstruction based on lower dimensional localized delaunay triangulation. In *Proceedings of Eurographics*, volume 19. ACM, 2000.

[17] H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, and W. Stuetzle. Piecewise smooth surface reconstruction. *Computer Graphics*, 28(Annual Conference Series):295–302, 1994.

[18] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. *Computer Graphics*, 26(2):71–78, 1992.

[19] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. *Computer Graphics*, 27(Annual Conference Series):19–26, 1993.

[20] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *Computer Vision*, 1:321–331, 1987.

[21] D. Lazzaro and L. B. Montefusco. Radial basis functions for the multivariate interpolation of large scattered data sets. *Journal of Computational and Applied Mathematics*, 140:473–483, 2002.

[22] L. Linsen and H. Prautzsch. Fan clouds - an alternative to meshes. In *Proceedings of 11th International Dagstuhl Workshop on Theoretical Foundations of Computer Vision*. Springer-Verlag, 2003.

[23] C. Loop. Smooth subdivisions surfaces based on triangles. Master's thesis, Department of Mathematica, University of Utah, August 1987.

[24] W. Lorensen and H. Cline. Marching cubes : a high resolution 3d surfaceconstruction algorithm. *Computer Graphics*, 21:163–169, 1987.

[25] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H. Seidel. Multi-level partition of unity implicits. In *Proceedings of ACM SIGGRAPH 2003*, Aug. 2003.

[26] Y. Ohtake, A. Belyaev, and H. Seidel. 3d scattered data approximation with adaptive compactly supported radial basis functions. In *Proceedings of Shape Modeling International 2004*, 2004.

[27] Povray. Povray, 2004. http://www.povray.org.

[28] P. Reuter. *Reconstruction and Rendering of Implicit Surfaces from Large Unorganized Point Sets*. PhD thesis, LaBRI - Universit Bordeaux 1, 2003.

[29] C. Shen, J. F. O'Brien, and J. R. Shewchuk. Interpolating and approximating implicit surfaces from polygon soup. In *Proceedings of ACM SIGGRAPH 2004*. ACM Press, Aug. 2004.

[30] C. T. Silva. Out-of-core algorithms for scientific visualization and computer graphics. IEEE Visualization 2003, Tutorial No. 4, October 2003.

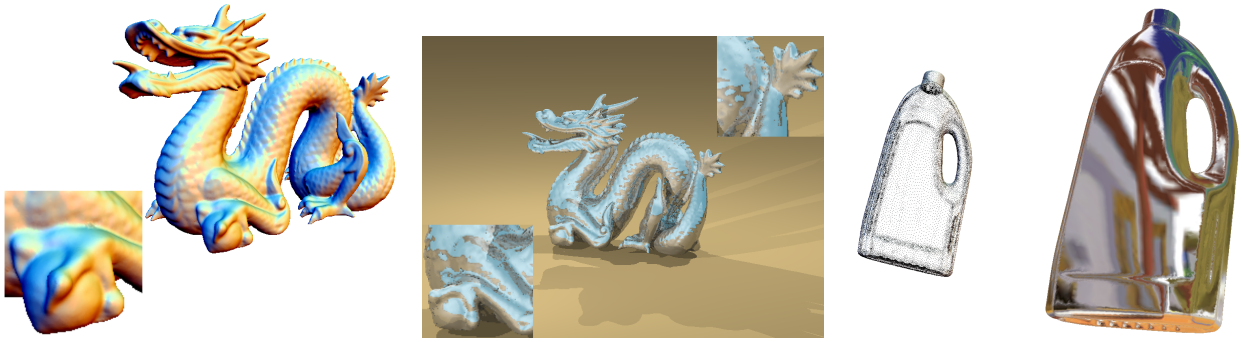
[31] I. Tobor, P. Reuter, and C. Schlick. Multiresolution reconstruction of implicit surfaces with attributes from large unorganized point sets. In *Proceedings of Shape Modeling International 2004*, 2004.

[32] G. Turk and J. F. O'Brien. Implicit surfaces that interpolate. In *SMI*. ACM, 2002.

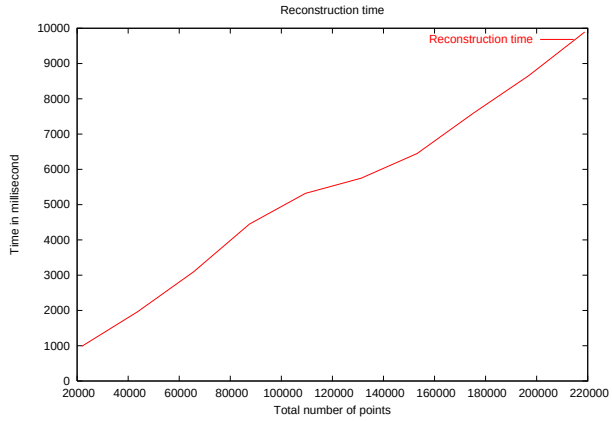
[33] H. Wendland. Fast evaluation of radial basis functions: Methods based on partition of unity. *Approximation Theory X: Wavelets, Splines, and Applications*, pages 473–483, 2002.

[34] D. Zorin and P. Schrder. Subdivision for modeling and animation. In *SIGGRAPH Courses Notes*. ACM, 2000.

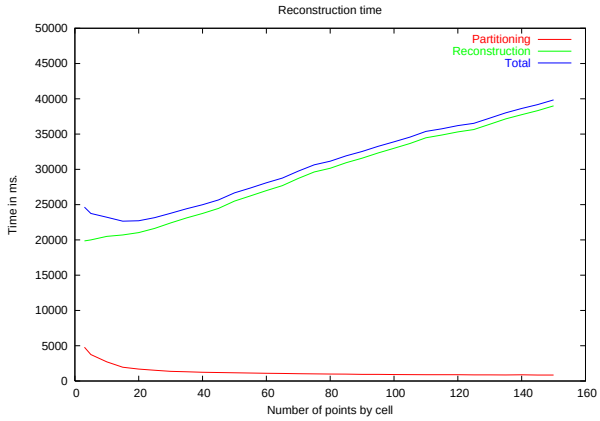
[35] M. Zwicker, H. Pfister, J. V. Baar, and M. Gross. Surface splatting. In E. Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 371–378. ACM, ACM Press / ACM SIGGRAPH, 2001.



(a) (b) (c)



(d)



(e)

Figure 10: Different rendering of our surfaces. (a) Realtime rendering with OpenGL. (b) A raytraced image with a reflexion shader. (c) Direct use of *Cube Mapping* in OpenGL, an example of the immediat benefit of our technique. The artefacts coming from overlappings are few or not visible. (d) Computation time (in milliseconds) for the reconstruction of the Stanford dragon at different resolutions. (e) Variation of the computation time (in milliseconds) for the Stanford dragon with 437 646 points, according to the threshold value k (i.e. maximum number of points allowed per cell). An optimum can be seen at $k = 20$.

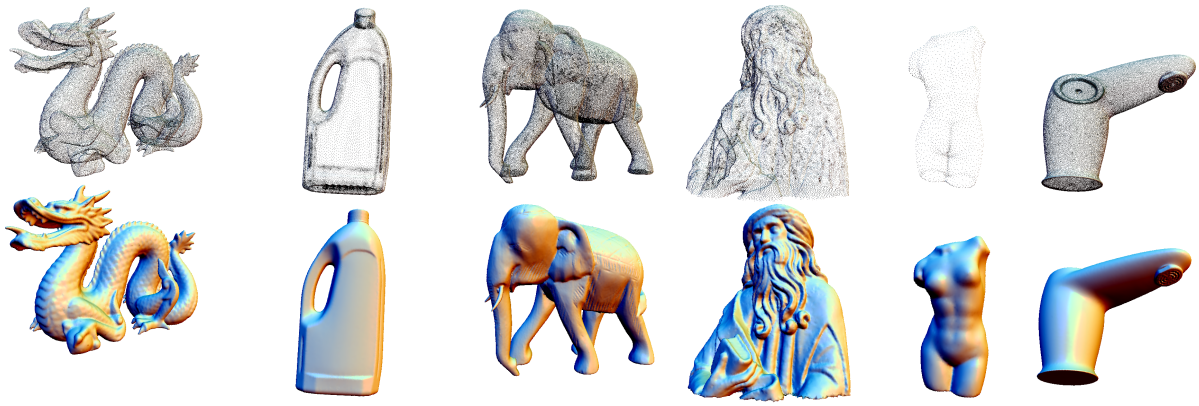


Figure 11: Reconstruction and rendering of various point clouds (times present in the Table 9).