Multilevel Representation and Transmission of Real Objects with Progressive Octree Particles

Yücel Yemez and Francis Schmitt

Abstract—We present a multilevel representation scheme adapted to storage, progressive transmission, and rendering of dense data sampled on the surface of real objects. Geometry and object attributes, such as color and normal, are encoded in terms of surface particles associated to a hierarchical space partitioning based on an octree. Appropriate ordering of surface particles results in a compact multilevel representation without increasing the size of the uniresolution model corresponding to the highest level of detail. This compact representation can progressively be decoded by the viewer and transformed by a fast direct triangulation technique into a sequence of triangle meshes with increasing levels of detail. The representation requires approximately 5 bits per particle (2.5 bits per triangle) to encode the basic geometrical structure. The vertex positions can then be refined by means of additional precision bits, resulting in 5 to 9 bits per triangle for representing a 12-bit quantized geometry. The proposed representation scheme is demonstrated with the surface data of various real objects.

Index Terms—Multiresolution 3D models, progressive representation, surface particles, octree, levels of detail, direct triangulation.

1 Introduction

Three-dimensional object models, computer generated or reconstructed from scanned real data, are becoming an essential part of the audio-visual data of the virtual world and the emerging multimedia technologies. These 3D models may represent objects for various applications in many domains such as e-commerce, entertainment, education, architecture, cultural preservation, CAD and medical applications, construction, and automation. Once constructed, they can be stored in local databases, transmitted, for instance, through the Internet and visualized or manipulated when needed. The increasing popularity of 3D models that can sometimes be very large in terms of data size, has activated a relatively recent research area for developing efficient ways of representing 3D object data.

For computer graphics applications, the surface of a 3D real object often has to be densely digitized into a large data set containing millions of surface points in order to achieve a satisfactory level of accuracy. The initial dense data may, for example, be laser scanned range values or voxelized data in the case of shape from silhouette/stereo techniques, or may result from MRI scans in medical imaging. The surface model is often obtained first by transforming the acquisition data into a dense triangle mesh, which is usually huge in size and, therefore, very expensive to store, transmit, and render. The challenge for such huge data is two-fold. First, this necessitates developing flexible representations which allow the user to choose the best compromise between efficient storage, fast data access, and high quality visualization. Progressive schemes seem to serve well for this purpose since an exact representation of the geometry is not always required for applications such as navigation or browsing with limited bandwidth transmission. With these schemes the initial mesh is stored as a coarse model along with the information which incrementally refines the geometry. In a progressive transmission scheme, a possible scenario is as follows: First, the coarse model with the lowest quality is sent and visualized at the receiver part. If the receiver demands further improvement in the quality with respect to its storage, processing, and graphics capabilities, a sequence of refinement operations is additionally transmitted so that finer levels of detail can progressively be received and displayed.

The second part of the challenge is to handle huge initial data in order to obtain such progressive representations in a time and space efficient manner. Building up progressive representations from initial data, containing, for instance, hundreds of thousands of vertices, may need up to dozens of hours of preprocessing time and megabytes of storage space. The initial model, as stated earlier, is usually a dense triangle mesh that can be simplified by various mesh approximation methods [11]. These techniques can provide several versions of the model at various levels of detail. Some of these techniques simplify the geometry by means of incremental refinement operations and can naturally be used to build up progressive schemes. Usually, the preprocessing time needed to obtain a progressive mesh depends on the approximation quality of the produced levels of detail. Some techniques favor accuracy over speed, such as in [17], and others are fast but less accurate such as in [12]. A similar trade off appears also for storage requirements. Even if the representation load of 3D models can be reduced by means of compression schemes, the resulting storage requirement still remains problematic in the case of very large dense meshes. The critical question here is, in fact, how much of the initial accuracy existing in the full data set is required for the highest resolution level of a progressive model. The answer depends on the underlying application. For some applications, details are not so

- *Y. Yemez is with Koç University, Computer Engineering Department, Rumeli Feneri Yolu, 80910 Sariyer-Istanbul, Turkey. E-mail: yyemez@ku.edu.tr.*
- *F. Schmitt is with ENST-CNRS URA820, Signal and Image Processing Department, 46 rue Barrault, 75013 Paris, France. E-mail: francis.schmitt@enst.fr.*

Manuscript received 9 June 2000; revised 22 May 2001; accepted 28 Nov. 2001.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number 112251.

important and the highest resolution level can correspond to a significantly reduced version of the initial model. However, in other cases, for instance, in creating archival data sets or in scientific applications, almost no or very little loss of information is tolerated and storage efficiency becomes a much more critical issue.

This paper[1] presents a novel progressive scheme for representing dense surface data, which addresses this two-fold challenge. The proposed technique is particle-based (or point-based), unlike traditional triangle-based schemes. Particles as modeling primitives are favored in our technique since they are easier to process as compared to triangles and yield very compact representation schemes. In our progressive modeling framework, topology is not preserved when producing different levels of detail and the resulting scheme does not need to store any connectivity information for encoding geometry. The initial data that we model is usually very dense, but it may be downsampled in order to homogenize the spatial distribution of the digitized points on the object surface. Such a high resolution scheme may be of interest, especially when storing the original data at its original fine resolution is required, or when it is necessary to keep pointwise surface properties or measurements together with the object geometry.

The organization of the paper is as follows: Section 2 briefly discusses the related previous work on 3D progressive modeling. Section 3 provides a brief overview of our general modeling scheme. Sections 4, 5, and 6 then follow, in which we respectively describe the partitioning, encoding, and rendering phases of the proposed scheme. Geometry compression issues are discussed in Section 7 and experimental results are shown in Section 8. In Section 9, we briefly discuss the trade offs related to the proposed scheme and, finally, give some concluding remarks in Section 10.

2 Prior Art

Progressive LOD (level of detail) representations existing in the literature can be grouped into two main approaches: triangle-based and point-based schemes.

2.1 Triangle-Based Schemes

The most common surface representation used for modeling 3D objects is the triangle mesh. The existing triangle-based surface construction techniques usually produce very dense triangulations. The load of the resulting representations can be reduced by mesh simplification methods [5], [6], [12], [16], [40] that build several versions of the initial model at various levels of detail. The most convenient representation can then be chosen for transmission and displayed according to the needs of the user and the application [10]. This direct approach, however, is not space efficient and necessitates additional transmission time when switching between different levels of detail is required. More efficient solutions producing a sequence of meshes have been proposed in literature [17], [44], [30], [19], [27], [8], [15]. These solutions can, in general, be divided into two groups which follow two different trends.

1. This work has partially been presented in [47].

The techniques belonging to the first group are recursive subdivision schemes [19], [27], [8]. The idea is to refine a coarse mesh by a sequence of subdivision steps, thereby modifying the connectivity and incorporating geometric details incrementally as the mesh resolution gets higher. These techniques are truly progressive representations and may yield very high compression ratios as in [19]. However, they are applicable to only a limited class of meshes requiring a special type of connectivity that can be obtained at the cost of an expensive remeshing procedure [23].

The second group of techniques [17], [44], [30] are based on the PM (Progressive Meshes) scheme of Hoppe [17]. In the PM scheme, the mesh refinement record contains vertex splits. Each vertex split operation increases the number of mesh vertices by one, modifying the connectivity and geometry accordingly. This flexibility provides a fine granularity for LOD representation as well as view dependent refinement. However, two vertices cannot be split at the same time and the storage size depends heavily on the size of the initial mesh, making the PM scheme impractical for very large meshes. In this respect, the PFS scheme of Taubin et al. [44] and the CPM scheme of Pajarola and Rossignac [30], can be regarded as better compressed versions of the PM scheme. In these methods, the vertex splits are grouped into batches, e.g., into forest splits in the PFS scheme. In this way, the granularity of the resulting progressive representation becomes limited, but the storage cost per triangle for encoding connectivity changes becomes independent of the initial mesh size. At the latest state-of-the-art, Pajarola and Rossignac [30] report a fixed number of 3.6 bits per triangle for encoding connectivity.

Flexibility of the PM scheme offers several other possibilities, and many extensions to this scheme have been proposed in the last half decade. One direction of these possibilities is view-dependent mesh refinement and another is topology simplification. Hoppe [18] extends his work by developing a runtime view-dependent simplification algorithm, which is guided by view frustrum and surface orientation. In a relatively early paper, Rossignac and Borrel [35] address the topology simplification problem and propose an octree-based vertex clustering technique that produces simplified boundary representations of arbitrary polyhedron. Later, Popovic and Hoppe [33] introduce an extension to the PM scheme that can also handle progressive changes to the topology. Finally, the works [28], [9] handle both problems, i.e., view dependent refinement and topology simplification, in a single scheme. Luebke and Erikson [28] construct a tight octree hierarchy to cluster the vertices of a mesh, whereas El-Sana and Varshney [9] achieve a fine control on simplification by constructing a hierarchy of vertex collapses to build a view-dependence tree.

Triangle-based progressive encoding algorithms have, in recent times, become a common practice in public and commercial arenas. The PFS scheme [44], for instance, has been included in the latest MPEG-4 Version 2 standard. However, the performance of triangle-based schemes, especially in preprocessing time, degrades as the size of the initial mesh gets larger, as in the case of scanned real objects. In order to obtain high quality low resolution models, these techniques spend too much effort per primitive to optimize the placement of individual vertices.

As an example, Hoppe [17] reports 10 hours of preprocessing time for a mesh of about 200,000 vertices.

2.2 Point-Based Schemes

An alternative to triangles, as a simpler display primitive, is the use of surface points [24], [14], particles [34], [43], or surfels [31]. The terms "particle" or "surfel" are used in the literature to denote a dimensionless space point attributed with context-dependent surface properties. Point-based representation systems have recently gained attention for progressive modeling of large data sets [47], [36], [37]. These techniques, compared to triangle-based schemes, do not treat acquisition data as exact and spend much less effort per primitive. They do not store any connectivity or topology information. Their advantage comes in terms of efficiency in preprocessing time, storage, and rendering of large acquisition data. However, low resolution approximations produced by these techniques are not as good as those resulting from triangle-based techniques.

A progressive particle-based surface modeling technique was first proposed by the authors in [47]. This paper extends the shape from silhouette technique presented in [47], to a more general framework. This technique is based upon a hierarchical octree representation and can thus be regarded as a volumetric method to model the surface data. In the literature, the octree representation is more commonly used for volume data representation and for extraction of its isosurfaces [1], [2], [41], [46], [32]. In [1], [2], [41], for example, the octree hierarchy is exploited to control the simplification error and thereby to produce isosurfaces at different levels of detail. However, these techniques are not intended to be progressive.

Two recent, very closely related works by Rusinkiewicz and Levoy [36], [37] follow the framework presented in [47], and propose a point-based technique to render their large data sets resulting from the Digital Michelangelo project [25]. Rather than an octree representation, they construct a sphere hierarchy and render the resulting representation via splatting. Splatting techniques avail them with the possibility of implementing a progressive interactive representation that can locally be refined with respect to the viewpoint. The major drawback of this representation is in rendering quality, since splatting techniques result in visual artifacts, especially in rendering low resolution models and in zooming on a detail.

3 Overview

In this work, we consider the following three steps for the whole object representation scheme: partitioning, encoding, and rendering. For each one, we employ a different type of representation. At the first step, the octree space partitioning provides us with a simple and robust way of representing the object surface hierarchy. This partitioning produces a sequence of octree surfaces, $S^0, S^1, \ldots, S^R$, which voxelize the surface geometry at R different levels of detail. In order to represent the object surface with a more precise shape, as well as other surface attributes such as color, at the second step, we associate to each octree surface cube a particle. The associated surface particles yield compact and space efficient encoding of overall object appearance. Triangle meshes are considered at the third step, only as a means to render the object so that the triangulation process becomes the task of the renderer. Thus, the stored representation which has to be transmitted is not a triangle mesh, but just a properly ordered sequence of surface particles $P^0, P^1, \ldots, P^R$. The encoded particles are arranged in such an order that they can progressively be decoded to reconstruct the octree surfaces $S^0, S^1, \ldots, S^R$ and the related surface attributes. These LOD surfaces can then be rapidly triangulated by the viewer for rendering with the help of a look-up table, yielding a sequence of triangle meshes $M^0, M^1, \ldots, M^R$ (see Fig. 1). With the proposed scheme, not only the object geometry, but also other object attributes such as color, normal, transparency, or any other scientific measurement, can be encoded and displayed in a progressive manner.

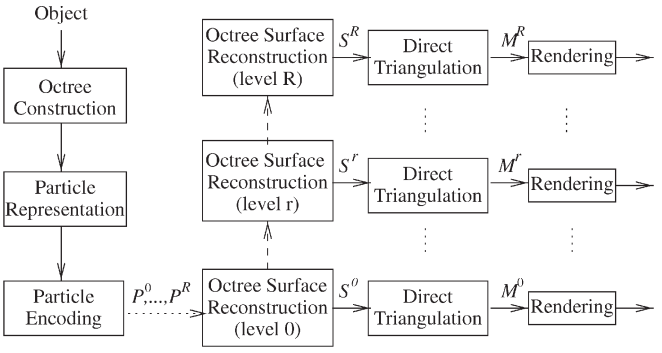


Fig. 1. Block diagram of the progressive multilevel object representation scheme.

4 Octree Partitioning

The octree representation is a well-known hierarchical data structure that can be used to partition the 3D space into cubic voxels of varying sizes [38], [3], [4]. In this section, we first recall the general octree definition and highlight some of its characteristics that we will make use of throughout the paper. The hierarchical partitioning that we use is not new to the object modeling literature [42], [46], [20]. The originality of our approach that we describe in this section comes rather from the way we interpret the octree hierarchy to construct a progressive framework for our overall scheme.

4.1 Definition

In volume modeling applications, each node of the octree typically corresponds to a cube, which is either totally inside the object volume, totally outside the volume, or on the object boundary surface. The object volume to be represented can be assumed to be bounded by an implicit surface $f(x, y, z) = 0$, which provides the necessary information to mark a given node with a label F equal to IN, OUT, or ON, respectively. If the octree space is considered as a 3D grid of $2^R \times 2^R \times 2^R$ unit cubes, where R denotes the highest resolution level of the octree, the octree representation is obtained by recursively subdividing each parent cube into eight child cubes, starting from the root node, i.e., the bounding cube. The OUT and IN cubes need not be further subdivided. The recursive subdivision process continues only for the ON cubes until the unit cubes which correspond to the leaf nodes of the highest octree level are reached [46].

The coordinate system is defined so that one corner of the octree space is located at the origin and its corresponding edges are aligned with the positive coordinate axes.

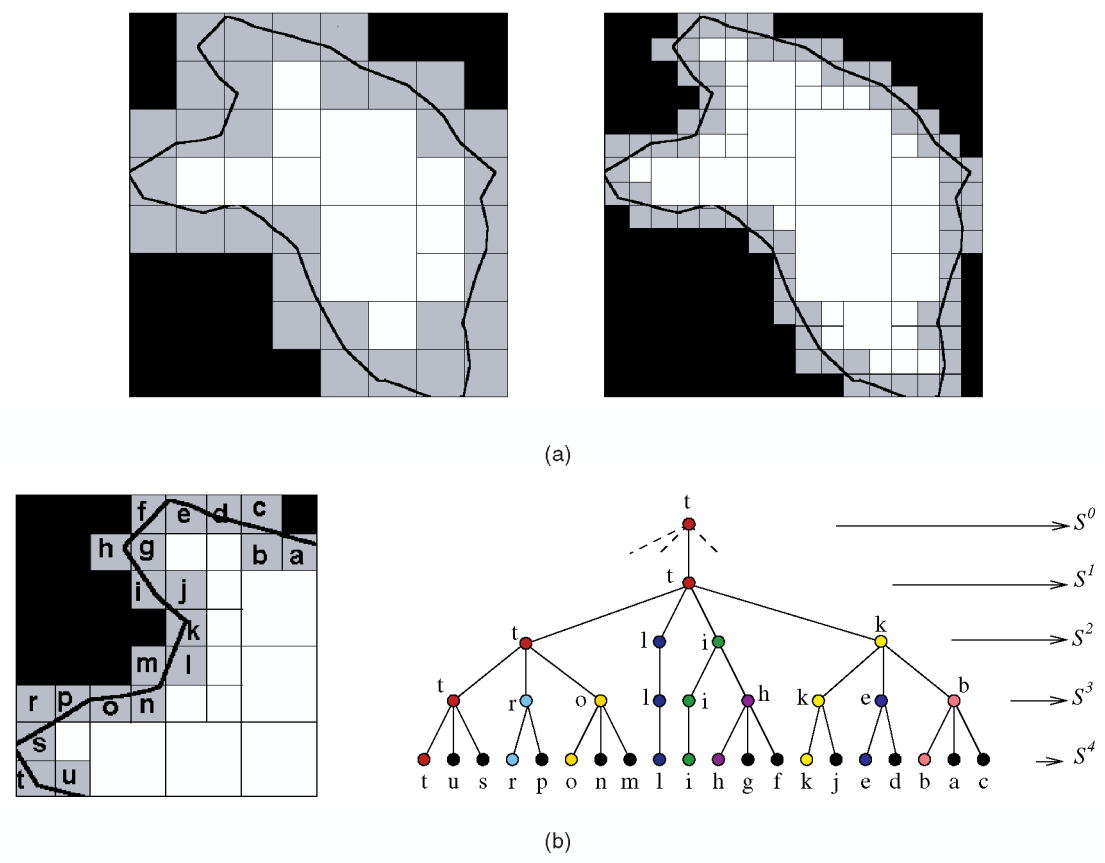


Fig. 2. (a) 2D cross section of a 3D surface represented in terms of quadtree regions at two different resolutions with (left) $r = 3$ and (right) $r = 4$. Gray regions correspond to ON cubes in the 3D octree space, whereas black and white regions represent OUT and IN cubes, respectively. (b) Octree representation of the 3D surface cross section (limited to one quadrant). The nodes with the same letter and color are those specified with the same leaf node. The sequence of ON leaf nodes ordered as $\{t, l, i, k, r, o, h, e, b, u, s, p, n, m, g, f, j, d, a, c\}$ suffices to progressively construct the octree structure. The ordering strategy will be addressed in Section 4.4.

Definition 1. *Each cube (or node) of the octree space is denoted by s_i^r, $i \in [0, 2^{3r} - 1]$, where r is its level (depth) in the octree hierarchy and i is the index of its corner closest to the origin in the 3D grid. The indices r and i are used in the paper to identify a node, but they are both implicit by the position of the node in the octree structure.*

Every node s_i^r is then uniquely specified by its level r and the discrete coordinates (x_i, y_i, z_i). Alternatively, an octree node s_i^r can be represented by a path (a sequence of nodes) originating from the root node s_0^0 and ending up with that node. This path can easily be encoded with a binary address by tracing the nodes belonging to the path and keeping, for each parent node, the code (0 to 7) of the child node traversed by the path. The code C of a child node defines a lexicographic order and is simply deduced from the geometrical position (c_x, c_y, c_z) of the corresponding child cube within its parent:

$$C = c_x + 2c_y + 2^2 c_z, \tag{1}$$

where c_x, c_y, and c_z take values 0 or 1.

4.2 Octree Surfaces

The hierarchical structure of the octree can be exploited to display the object surface at R different levels of detail. Due to the octree construction described in Section 4.1, there exist no two neighboring cubes with one labeled OUT and the other labeled IN. Thus, there exists between the interior and exterior of the object, a layer of ON cubes along the object boundary assumed to be defined by $f(x, y, z) = 0$. Such a layer exists at any hierarchy level r since an ON cube should necessarily be further subdivided due to the octree definition.

Definition 2. *The octree surface S^r, i.e., the ON cube layer at level r, is defined by $S^r = \{s_i^r \mid F(s_i^r) = \text{ON}\}$. The surface S^r is made up of cubes all with the same size $2^{R-r} \times 2^{R-r} \times 2^{R-r}$.*

To illustrate an octree surface, we use the analogy between octrees in 3D and quadtrees in 2D. In Fig. 2, a quadtree construction is visualized for a 2D region represented at two different resolutions $(r = 3, 4)$, the resulting structure also being sketched at the bottom of the same figure with $R = 4$. This example can be thought of as a cross section from the octree representation of a surface in 3D space.

4.3 Octree Surface Thinning

Octree surfaces, as defined by Definition 2, contain some cubes which are not necessary for the surface visualization. These ON cubes have edges which intersect the surface, but do not have visible faces when viewed from the exterior of the object. Such occluded cubes can be regarded as topologically redundant. They unnecessarily increase the load of the surface representation and complicate the direct surface triangulation process that will be addressed later in Section 6.1. Therefore, they can be eliminated from the corresponding octree surface S^r, just by changing their

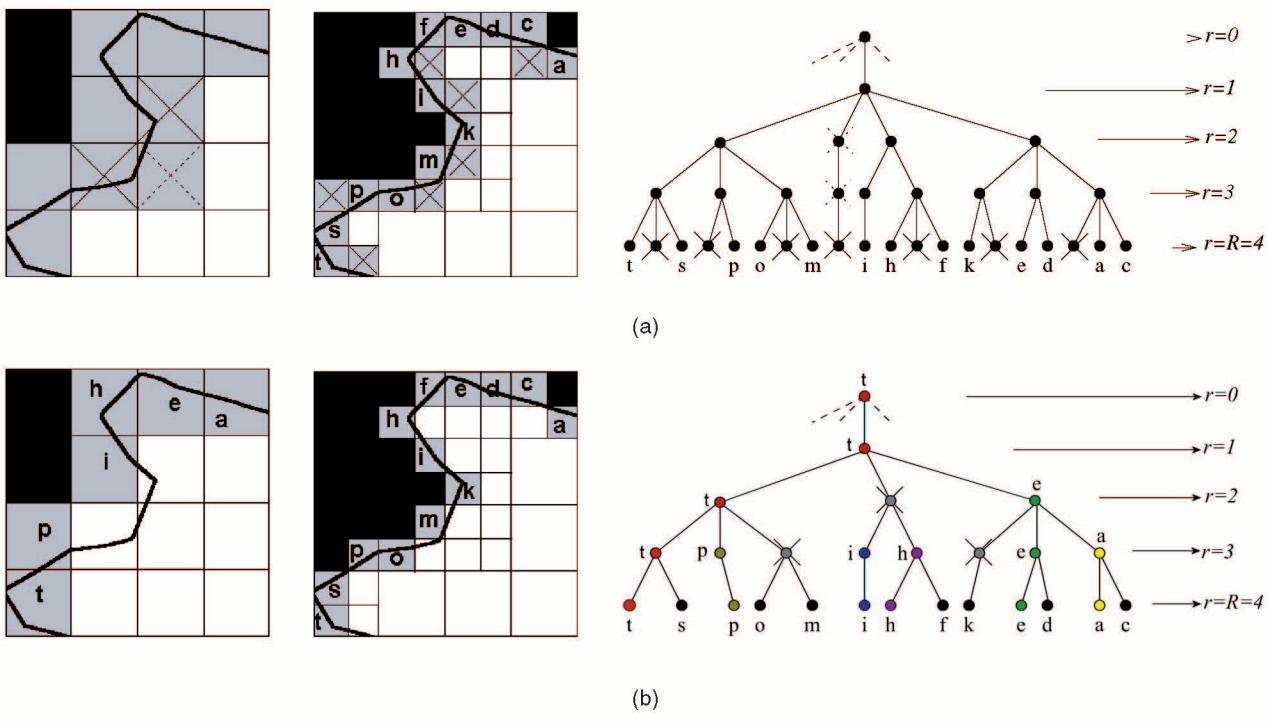


Fig. 3. (a) Octree surface thinning at levels $r = R - 1$ and $r = R$, and the resulting octree structure. The nodes crossed with solid lines denote the eliminated cubes that are occluded in the context of the entire object, as illustrated in the upper right of Fig. 2. Note that the node r in Fig. 2, having no IN face-neighbor, has also been eliminated and transformed into a black OUT cube. The nodes crossed with dotted lines are the parent nodes which are eliminated since they have no more ON child nodes. (b) Octree surface thinning after all levels $r \leq R$ have been processed. Note that the hanging nodes having no ON node descendant have been removed from the structure (see in the upper right the node crossed with dotted lines). The ordering of the leaf nodes is now different from that of Fig. 2. The sequence of the difference sets $S^{R,r} \setminus S^{R,r-1}$, $r = 0, 1, \ldots, 4,$ in this case, is given by $\{\{t\}, \emptyset, \{e\}, \{p, i, h, a\}, \{s, o, m, f, k, d, c\}\}$. The ordering strategy will be addressed in Section 4.4.

labels to IN. This elimination corresponds to a thinning operation on the octree surfaces as defined in morphological mathematics. It can also be viewed as a downsampling process, which slightly simplifies the octree so that it can more easily be visualized at the rendering step.

The octree surface thinning process can be realized by postprocessing the whole octree starting from level R up to level 0. The occluded cubes at each level r of the iteration are identified by verifying the labels of its six face-adjacent cubes: If a cube s_i^r has no OUT face-adjacent neighboring cube it is eliminated from the set S^r as illustrated in Fig. 3 (see the nodes crossed with solid lines).

Since transforming an ON cube into an IN cube changes the visibility of its neighboring cubes, the elimination process must be sequential. Besides, for each ON node s_i^r eliminated, the label of its parent should also be reconsidered. If the parent node of s_i^r becomes a hanging node so that it no longer has an ON child as a result of the elimination, it is also eliminated from the surface S^{r-1} (see in upper right of Fig. 3, the nodes crossed with dotted lines at levels $r = 2$ and $r = 3$). The whole thinning process can be described as follows:

for $r = R : 1$
  for all $s_i^r \in S^r$
    if (s_i^r is occluded)
      $F(s_i^r) \rightarrow$ IN;
      for all the ancestors s_j^p of s_i^r, $p = r - 1 : 1$
        if (child nodes of s_j^p are all IN) $F(s_j^p) \rightarrow$ IN;
  repeat until no s_i^r is occluded;

This process eliminates the redundant cubes on the inner boundary of the surface. The same elimination process can also be applied to ON cubes on the outer boundary, i.e., to those that do not have any IN face-neighbors, transforming them into OUT cubes (e.g., the node r in Fig. 2). However, the consequences of such an elimination process need to be analyzed with particular attention. First, this may yield in the loss of some visual information. At high resolutions, considering that the initial data points are not exact due to acquisition noise, this loss can be regarded as not so significant. At low resolutions, however, elimination of ON nodes on the outer boundary may filter out some visually significant parts of the object surface, e.g., those which are thin enough to be included inside a single octree cube of the current resolution. Second, artificial holes may appear if, for instance, two separate parts of the object surface get so close to each other that the octree cube size at the current resolution becomes too large to differentiate the separation, yielding locally, a layer of one elementary cube size. As a consequence, the cubes on this layer, having no interior neighbor, would be eliminated creating an artificial hole (see Fig. 4).

A partial remedy to the problems stated above is to leave the thinning process for the outer boundary as optional; depending on the object complexity and detail, one may choose to keep ON cubes having no IN face-neighbors. In this case, we say that the thinning process is relaxed. However, when the thinning process is applied both for the inner and outer boundaries, an additional test on local topology has to be performed to verify that elimination of an outer ON node does not create a hole. This test is based

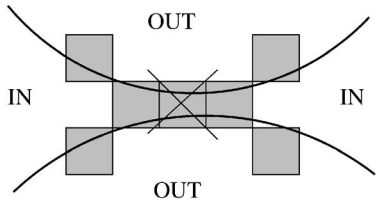


Fig. 4. Illustration of an artificial hole which might be produced by the octree surface thinning process. The crossed cube has no face common with the interior. However, it is preserved since its elimination would change the local connectivity and create a hole.

on a labeling process, which first identifies the connected OUT regions in the $3 \times 3 \times 3$ neighborhood of the underlying ON cube, the connectivity being defined along the six faces of a cube. If the elimination of the ON cube causes any two formerly separate OUT regions to be connected and, thus, creates a hole, the cube is not eliminated.

Provided that the current octree resolution is sufficiently fine to represent the surface, the thinning process eliminates at most half of the ON nodes to be coded for each level as it will later be verified with the experimental results. In Fig. 3b, we observe that the eliminated leaf nodes and the hanging nodes with no child have all been removed from the final octree structure. Some of the nodes which have been eliminated at an intermediate level r, may indeed have ON child nodes to be coded at higher levels (see the crossed nodes in gray color in Fig. 3b). Such nodes need to be kept in the data structure for compression purposes and will be referred to as phantom nodes later in Section 7.1.

Although the described thinning process brings in storage efficiency at the cost of some information loss at low resolutions, the main purpose of this step is closely related to the rendering phase of our scheme, for which we employ triangle meshes as we will see later in Section 6.

4.4 Progressive Octree Surfaces

One key observation in defining a progressive octree representation is that the set of all ON leaf nodes suffices to construct the whole octree structure. Furthermore, the ON leaf nodes can be arranged in such an order that the successive leaf nodes of this sequence incrementally reconstruct the octree surfaces S^r, $r = 0, 1, \ldots, R$ (see Definition 2). To simplify the description of our ordering strategy, we proceed with the following definition:

Definition 3. *The suboctree which originates from the node s_i^r and includes all the nodes under s_i^r, is denoted by T_i^r.*

For every ON node s_i^r, there exists at least one ON leaf node s_j^R such that $s_j^R \in T_i^r$, since every ON node s_i^r has at least one ON child node due to the octree construction. This property is also preserved during the octree surface thinning process. This allows us to specify every node s_i^r by a leaf node $s_j^R \in T_i^r$ and its depth r. Conversely, for any leaf node s_j^R, the path originating from the root and terminating at that leaf node includes, at each level $r < R$, a node s_i^r.

Definition 4. *In a given octree surface S^{r-1}, each node s_i^{r-1} is associated with an ON leaf node $s_j^R \in T_i^{r-1}$. We denote the set of these ON leaf nodes, for a given level $r-1$, by $S^{R,r-1}$.*

The set $S^{R,r-1}$ suffices to specify the octree surface S^{r-1}. Since the path passing through s_i^{r-1} and terminating at s_j^R necessarily traverses the level r, the ON leaf nodes of the set $S^{R,r-1}$ can all be reutilized to specify the traversed nodes of the octree surface S^r. In this case, we have the inclusion property: $S^{R,r-1} \subset S^{R,r}$. The set $S^{R,r-1}$ and the difference set $(S^{R,r} \setminus S^{R,r-1})$ are then sufficient to construct the surface S^r:

$$S^{R,r} = S^{R,r-1} \cup (S^{R,r} \setminus S^{R,r-1}) \longrightarrow S^r. \quad (2)$$

This hierarchical relation can be exploited for constructing a progressive representation by ordering the difference sets $(S^{R,r} \setminus S^{R,r-1})$ with increasing r. With $S^{R,0}$ associated to the root (which contains a single leaf node), the resulting sequence of difference sets can be decoded successively, giving the octree surfaces series $S^0, \ldots, S^r, \ldots, S^R$. Note that the total number of nodes in the sequence $\{S^{R,r} \setminus S^{R,r-1}\}$ is equal to the number of leaf nodes in S^R. The multilevel representation, in this way, is completely coded by properly ordering the difference sets in the sequence (see Figs. 2 and 3). The order of the nodes inside a difference set has no importance and can be arbitrary for progressive reconstruction. Nor does the decoder require any extra information to detect the transitions between successive resolution levels of the incoming data. If the current level is r, the arrival of a node which belongs to the level $r+1$, can easily be identified by the decoder during the octree reconstruction process since such a node necessarily traverses an octree node of level r, which has already been traversed.

The choice of an ON leaf node to specify a given s_i^r among a number of candidates $s_j^R \in T_i^r$ (see Definition 3), can be arbitrary for the construction of the octree surfaces. However, the cubes corresponding to the chosen leaf nodes at a given resolution level r will need to be distributed as regularly as possible in the octree space later when we introduce surface particles. Note that the child nodes of a parent node have a lexicographic order with respect to their codes (0 to 7) as defined by (1) in Section 4.1. This lexicographic order propagates to the full octree. For a given s_i^r, the ON leaf nodes belonging to T_i^r inherit this lexicographic order and, then, the leaf node s_j^R specifying s_i^r is chosen through a path searching algorithm privileging this order. Due to the above progressive octree surface construction strategy, all the child nodes on the path linking s_i^r to s_j^R (see Definition 4) are also associated to s_j^R directly without path searching. The lexicographic order, imposed by (1), also has a geometric meaning: The path searching algorithm always privileges the child cube which is closest to the origin. Consequently, the resulting leaf nodes specifying the ON nodes at a given level tend to be regularly distributed over the surface.

5 Particle Encoding

The octree representation of a 3D object describes its shape in a finite discrete grid of $2^R \times 2^R \times 2^R$ unit cubes. The precision of the representation is thus limited with the highest resolution level R. A proper object model, however, should represent a more precise shape as well as surface color or other attributes. In order to achieve this, in this section, we introduce octree surface particles.

5.1 Particle Definition

A particle is defined by a set of attributes which includes, necessarily, a vertex and, if available, other geometrical or appearance properties of the object such as the surface color or normal:

Definition 5. *We associate with each surface cube s_i^r of the octree, a particle $\{\boldsymbol{v}_i^r, \boldsymbol{c}_i^r, \boldsymbol{n}_i^r\}$, where $\boldsymbol{v}_i^r = \{x, y, z\}$ stands for the surface position of the particle, $\boldsymbol{c}_i^r = \{r, g, b\}$ for the color, and $\boldsymbol{n}_i^r = \{n_x, n_y, n_z\}$ for the normal to the object surface at the particle vertex. The vertex $\boldsymbol{v}_i^r$ is defined as a space point (x_i, y_i, z_i), which is inside the cube s_i^r and which lies on the object surface. We will denote the color of a child node $s_{i,j}^r$, $j = 0, 1, \ldots, 7$, by $\boldsymbol{c}_{i,j}^r$.*

5.2 Attribute Assignment

Assuming that the particle attributes of the leaf nodes at level R are known a priori, the first step is to deduce from them the attributes of the nodes at lower octree levels $r < R$. The way an attribute is assigned to a parent node depends on the type of the attribute. Here, we consider only the vertex, color, and normal attributes. The strategy that we follow for vertex assignment is different from the strategy for color and normal attributes.

The color (or normal) assignment of a parent node is done by averaging the colors (or normals) of its child nodes. Thus, starting from the level R, each $\boldsymbol{c}_i^r$ is assigned to the average of the colors $\boldsymbol{c}_{i,j}^{r+1}$ (see Definition 5) of its ON child nodes so that

$$\boldsymbol{c}_i^r = \frac{1}{m} \sum_j \boldsymbol{c}_{i,j}^{r+1}, \tag{3}$$

where m is the number of ON child nodes of s_i^r. The normals are determined likewise. We should point out that no particle attribute is assigned to the phantom nodes defined in Section 4.3. They are therefore disregarded in color averaging. Consequently, the ON child nodes of a phantom node have no contribution to the particle attributes of the ON nodes at coarser levels, for example, the leaf node k in Fig. 3 does not contribute to the attributes of the node e at level $r = 2$.

The averaging strategy used for color and normal attributes will cost extra encoding bits (see Section 5.5), but in turn, this type of smoothing gives much better visual results. The same strategy, however, cannot be applied to the vertex attribute assignment since the precision of vertex positions at intermediate levels is strictly required for reconstruction of the octree structure during the incremental decoding process. Therefore, the vertex attributes at intermediate levels are assigned as follows: A node s_i^r, which is specified with a leaf node s_j^R (see Definition 4), has the same vertex v_i^R as its leaf node.

5.3 Particle Ordering

We represent the whole object surface as an ordered sequence of particle sets that we will denote by $\{P^r\}$. The ordering in this sequence has similarities with that of the octree surface representation described in Section 4.4, but the structure of the representation is quite different since now, the other attributes such as color and normal are also to be encoded together with the surface geometry. The sequence $\{P^r\}$ can be formed by establishing a one-to-one correspondence with the difference set sequence $\{S^{R,r} \setminus S^{R,r-1}\}$ (see Definition 4 and (2)):

Definition 6. *The set P^r is defined as the set of particles $\{\boldsymbol{v}_i^r, \boldsymbol{c}_i^r, \boldsymbol{n}_i^r\}$ associated with the nodes s_i^r, which are specified by the leaf nodes s_j^R in the difference set $(S^{R,r} \setminus S^{R,r-1})$.*

Note that the number and the order of particles in the sequence $\{P^r\}$ are the same as those of the associated nodes in the sequence $\{S^{R,r} \setminus S^{R,r-1}\}$.

5.4 Particle Decoding

The particle sequence $\{P^r\}$, as defined in Definition 6, can be decoded progressively so that the octree surfaces S^r are incrementally reconstructed together with the associated particle attributes. At each level r, the particle decoding is a two-step process. At the first step, the vertices of the incoming particle set P^r together with the previous ones $P^0, P^1, \ldots, P^{r-1}$ directly give us the vertex attributes $\boldsymbol{v}_i^r$ of all the nodes in S^r. The position value of each $\boldsymbol{v}_i^r$ can then be used to deduce the address of the corresponding octree node s_i^r.

Once the octree structure and the vertex attributes of level r are decoded, at the second step, color and normal attributes are retrieved. Since the color decoding process is the same with normal decoding, here we will consider only the color attribute. First, the incoming particles in P^r directly provide the colors of the nodes that they specify. If we recall (see Definition 6) that the number of particles in P^r is in fact lower than the total number of particles of level r, there still remain some nodes at level r to be processed. The vertex attributes of these remaining nodes have already been specified by the previous sets of particles $P^0, P^1, \ldots, P^{r-1}$, following the paths linking them to leaf nodes. The color attributes of these nodes are indirectly retrieved by using both the color attributes of their parents at level $r-1$, and those of the nodes at level r that have already been determined by P^r. Recall that the color of a parent is defined as the average of the colors of its ON child nodes. Among the child nodes $s_{i,j}^r$ (see Definition 5) of a parent node s_i^{r-1}, only a single child $s_{i,k}^r$ inherits the vertex attribute of its parent and, thus, its color remains to be determined. Since the colors of the other children, if they exist, have already been determined directly by particles in P^r, the color $\boldsymbol{c}_{i,k}^r$ of this single child node $s_{i,k}^r$ can be retrieved by

$$\boldsymbol{c}_{i,k}^r = m\boldsymbol{c}_i^{r-1} - \sum_{j \neq k} \boldsymbol{c}_{i,j}^r, \tag{4}$$

where $m (1 \leq m \leq 8)$ is the number of ON child nodes of s_i^{r-1}, and $\boldsymbol{c}_{i,j}^r$, $j \neq k$, are the colors of the $(m-1)$ corresponding particles belonging to P^r. With this strategy, the color and normal information is simultaneously refined with the surface geometry.

5.5 Representation Load

The encoding of the color and normal attributes has to compensate the loss of numerical precision, which accumulates due to the averaging process as the higher levels of detail are reconstructed. The color is represented in the usual RGB format of 24 bits and its value, calculated by (3), is truncated to its integer part. Since a parent can have at most eight ON child nodes, the loss of precision due to this truncation can be avoided by providing 3 bits of correction to each color component, which has to be computed by (4). The encoder registers the truncation error each time the color of a parent node is averaged by (3), and attaches the resulting 3 bits to the color attribute of that parent node. The number of times where the RGB color has to be retrieved by the decoder in this way is equal to the number of particles in $P^0 \cup P^1 \cup \ldots \cup P^{R-1}$, which is, in practice, with nonfractal objects, fewer than

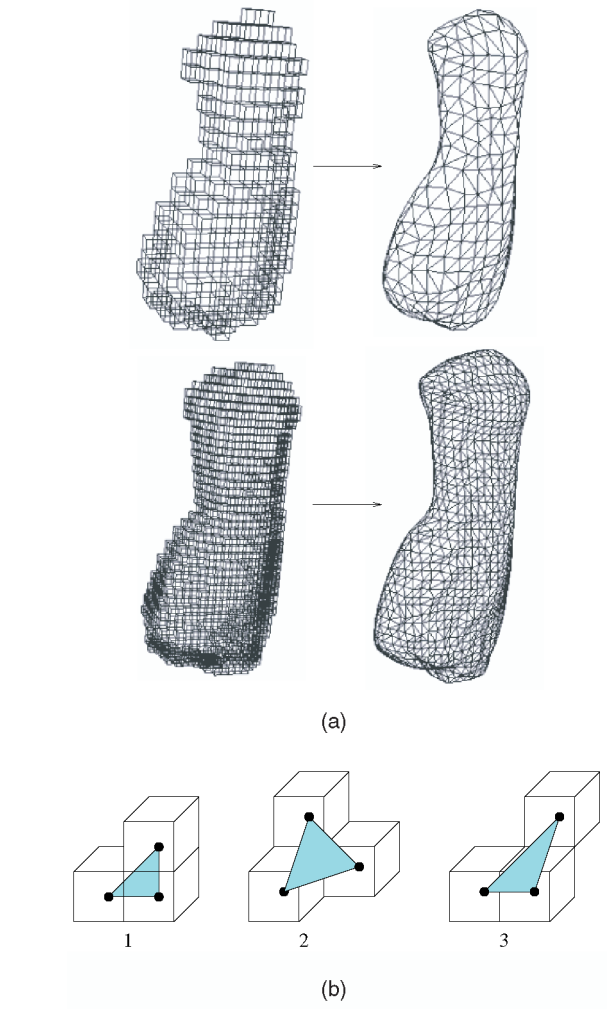


Fig. 5. (a) Direct triangulation of two octree surfaces at level $r = 5$ and $r = 6$. (b) The three different types of generated triangles.

half of the particles in P^R. Thus, if the total number of particles in P^R is n, the whole object color is encoded by $\approx 24n + 9(n/2) < 29n$ bits. The strategy to encode the normals is the same, the bit size needed for each normal component depending on the required precision.

Encoding of geometry, i.e., vertex locations, will be addressed in more detail later in Section 7. For the moment, without using any compression scheme, the representation load for geometry is a factor of the number n of particles and of the required number of precision bits for the vertex locations that we denote by N, $N \geq R$. Each vertex having three coordinate components, the whole object geometry requires $3Nn$ bits. In practice, $N = 12$ usually suffices, resulting in 36 bits per vertex.

6 Rendering

Compared to triangles, particles are not well-adapted for real-time display with 3D graphic boards. Moreover, usual graphic devices can efficiently antialias triangle representations, but additional processing would be required for particles. Another problem is that the representation breaks up when we zoom on the object: Particles become sparse resulting in holes which have to be filled by using, for example, splatting techniques [36]. An object stored or transmitted in terms of surface particles, should therefore be transformed to a representation which is convenient for visualization. At the rendering step of our representation scheme, we have chosen triangle meshes as a means to visualize the object surface.

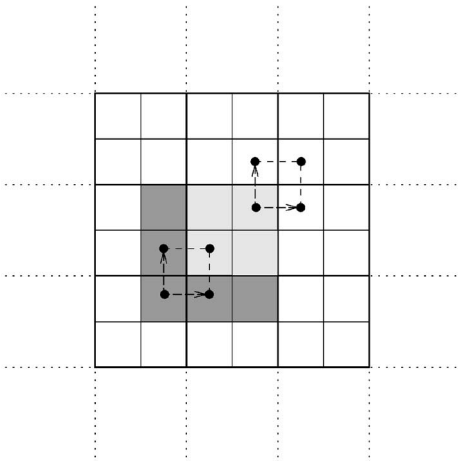


Fig. 6. An instant of the surface scanning procedure illustrated in 2D. The $6 \times 6 \times 6$ window is positioned on the surface with respect to a parent node (the central light gray region) at level $r-1$, having necessarily at least one ON child node. The light gray child nodes at level r are each considered as an origin reference for a $2 \times 2 \times 2$ neighborhood to be triangulated, whereas a back-neighbor (dark gray nodes at level r) is considered as reference, provided that its parent does not have any ON child node. This is to avoid processing duplication since a parent with an ON child node becomes central at another instant of the surface scanning procedure. In the figure, two such neighborhoods are sketched, one for a light gray and one for a dark gray (back-neighbor) reference node.

6.1 Mesh Generation

The particle representation inherently contains the hierarchical octree information. This octree structure deduced from the ordering and positioning of the particles in the discrete octree space, can be exploited to transform the sequence of particles $\{P^r\}$ into a sequence of triangle meshes $\{M^r\}$ via a direct triangulation technique. The direct triangulation is applied separately to each level r (see Fig. 5). It connects the particle vertices $\boldsymbol{v}_i^r$ so as to form a mesh M^r of connected triangles, using the local neighborhood information that can easily be determined thanks to the octree information S^r. It is possible to limit the triangle construction to three different types of triangles due to symmetrical and rotational equivalence [13]. These triangles are all included in a neighborhood of $2 \times 2 \times 2$ cubes belonging to the currently processed level r, as shown in Fig. 5.

The direct triangulation is implemented by scanning all the $2 \times 2 \times 2$ neighborhoods, each of which contains at least one ON cube. For each scanned neighborhood, the states (ON or not) of its eight cubes are determined and the resulting configuration is triangulated according to a look-up table and a set of construction rules. Now, we describe, step by step, the whole mesh generation process.

6.1.1 Surface Scanning

The access to the information in each neighborhood of $2 \times 2 \times 2$ cubes intersected by the object surface has to be fast, since the direct triangulation is applied at the rendering step for visualization. This information can be retrieved

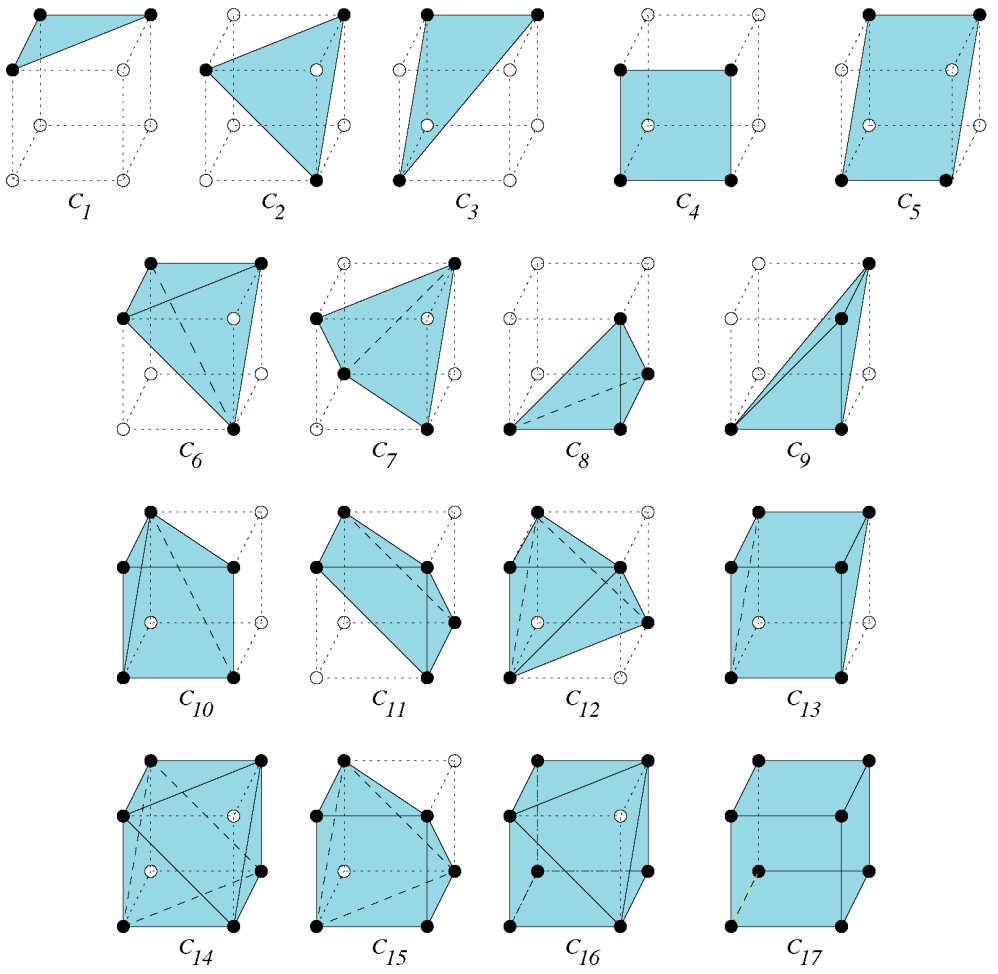


Fig. 7. The 17 distinct configurations $C_i, i = 1, \ldots, 17$, in a $2 \times 2 \times 2$ neighborhood.

through the octree representation via relatively time consuming path traversals, each starting from the root. Therefore, the number of these traversals should be kept as small as possible.

Our surface scanning strategy exploits the octree hierarchy. Suppose that the current resolution of triangulation is r. Around each parent cube at level $r-1$, having at least one child node, a window of $3 \times 3 \times 3$ cubes is constructed. Once the addresses of these 27 parent nodes are retrieved by path traversals, the addresses of their 216 child nodes at level r can be directly stored in a local array without any exhaustive path traversal. Then, all the $2 \times 2 \times 2$ neighborhoods of level r included in this local array can be triangulated at once (see Fig. 6). This local array also provides us with the information in the vicinity of each neighborhood, which is required for resolving some ambiguous configurations as we will see later in Section 6.2.

6.1.2 Look-Up Table

The number of possible configurations with ON and nonON nodes in a $2 \times 2 \times 2$ neighborhood is 2^8. However, due to symmetrical and rotational equivalence, the possible configurations can be reduced to 17 distinct cases, $C_i, i = 1, \ldots, 17$, as illustrated in Fig. 7. For each of these 256 possible configurations, the look-up table provides: 1) the corresponding index C_i, 2) a connected mesh graph $G_i = \{S_i, E_i\}$, and 3) an associated triangle face list T_i. The graph nodes in S_i are the ON cubes of the corresponding configuration C_i, whereas the graph edges in E_i correspond to the edges of the triangles listed in T_i. A triangle in T_i corresponds necessarily to one of the three different types of triangulation displayed in Fig. 5.

6.1.3 Construction Rules

The configurations $C_i, i = 1, 2, 3$, consisting of exactly three nodes, are those already shown in Fig. 5. These three configurations can be triangulated regardless of the neighboring information, each resulting in a single triangle as shown in Fig. 7. Among the different configurations that consist of exactly four nodes, those which occur most frequently are C_4 and C_5 (see Fig. 7). Triangulation of these two configurations necessitates a choice between two different local triangulations. The best one can be chosen so as to have a mesh surface which is locally as smooth as possible, for example, by minimizing the dihedral angles between adjacent triangles or by using the surface normal directions at the four vertices if the normal information is available.

All the other configurations with four or more nodes, $C_i, i \geq 6$, are ambiguous cases and need particular care in order to avoid topological problems such as holes or nonmanifold faces. Holes must definitely be avoided since

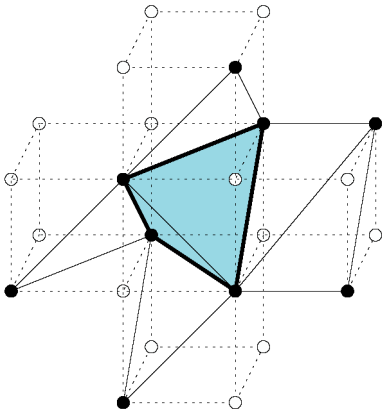


Fig. 8. Illustration of the unique edge cycle search for the configuration C_7. The cycle drawn with thick lines consists of the edges shared by other neighborhoods and traverses all the nodes of the configuration. Therefore, it cuts the graph G_7 into two meshes, each giving a consistent triangulation with the local surface. In the figure, only one of these two choices has been drawn.

they are visually very disturbing, whereas nonmanifold internal triangles can be tolerated up to a certain level since they do not cause visualization artifacts.

6.2 Disambiguation

Disambiguation of surface data is, in fact, a problem commonly encountered in surface triangulation techniques. A typical example is in the classical marching cubes algorithm [26]. In this case, the disambiguation methods proposed in the literature [29], [45] are based basically on the inside-outside information available in the vicinity of the surface. However, in our case, this information is not always available at the decoder; although it is required by the coder during the octree surface thinning process, the related normal information may be discarded during particle encoding for compression purposes. Therefore, the decoder can rely only on the neighborhood relations of the surface cubes in order to resolve possible ambiguous configurations.

The graph G_i, encoded for each $i \geq 6$ in the look-up table, gives as a whole two-manifold closed convex surface mesh, each edge in E_i being incident exactly with two triangles of T_i (see Fig. 7). Thus, if a given configuration C_i cannot be disambiguated, the construction of all triangles in T_i guarantees a hole-free triangulation, but in turn, creates nonmanifold internal faces when combined with adjacent configurations. Thanks to the octree surface thinning described in Section 4.3, the ambiguous cases arise very rarely in our modeling framework, as we will see later in the experimental results of Section 8.2, and those encountered can mostly be disambiguated.

Since the triangulation has to be rapidly performed by the decoder, the disambiguation process will not be exhaustive. What we need for disambiguation of a given configuration is a sufficient condition which can be quickly tested and which guarantees a topologically correct triangulation. This disambiguation process relies on the neighborhood information stored in the $3 \times 3 \times 3$ local array described in Section 6.1.1: For a given configuration C_i with $i \geq 6$, first, every edge in E_i that is shared with any of the neighboring (overlapping) $2 \times 2 \times 2$ configurations is identified. If all these shared edges form a unique Hamiltonian cycle traversing exactly once each node in the configuration (see Fig. 8), then they define an edge boundary which cuts the mesh graph G_i into two subgraphs. These subgraphs correspond to two different choices for a topologically correct triangulation since the edges resulting from any of these triangulations are locally guaranteed to be incident either with two triangles or for those at the edge boundary with one triangle. We then select the one which minimizes the average of all resulting diedral angles. If such a unique cycle does not exist, meaning that the configuration cannot be disambiguated, all the triangles in T_i are constructed resulting in a nonmanifold but hole-free triangulation. The search algorithm for a unique Hamiltonian cycle in a given configuration is very simple and straightforward. Moreover, it is employed only for the cases, $C_i, i \geq 6$, which are relatively rare. Therefore, the computational load of the cycle searching process is not significant on the overall triangulation time.

6.3 Decimation

A decimation task can also be embedded within the triangulation process: If a triangle with an edge smaller than a threshold is encountered, this triangle is not constructed and the corresponding two octree nodes are implicitly merged by averaging the attributes of their particles, preserving the octree structure. The decimation threshold is chosen proportionally to the octree cube size at the current level of visualization. In Fig. 9, we show the triangle mesh representation of a 3D object constructed with decimation.

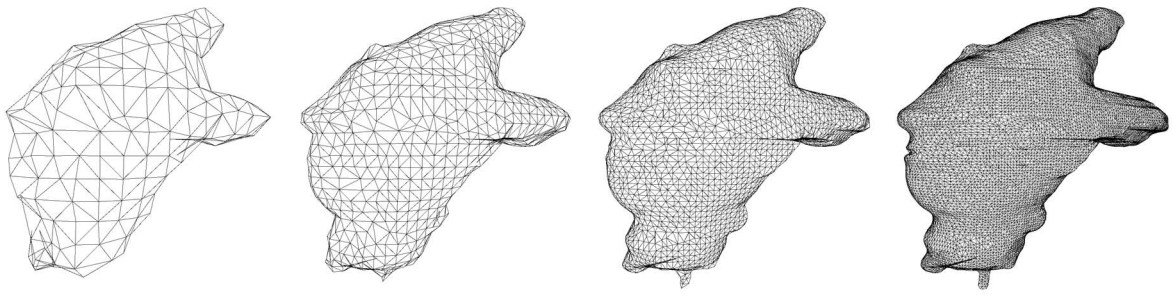


Fig. 9. Anyi statuette: Triangular meshes after decimation at increasing level of details $r = 5, 6, 7, 8$. These wireframe models contain 582, 2,410, 9,622, and 38,600 triangles, respectively.

6.4 Discussion

We recall that the classical marching cubes algorithm [26] works only with one cube at a time and needs the isosurface function values at the vertices of the cube in order to deduce the intersections of cube edges with the object surface. Compared to this, the proposed direct triangulation method treats mainly eight cubes at a time with surface particles inside and uses neighboring information to disambiguate the configurations $C_i, i \geq 6$, where possible. The direct triangulation process performed by the decoder works only on surface particles without requiring the inside and outside knowledge, and the resulting triangles are less than half of those resulting from the marching cubes. This is also how the direct triangulation technique that we have developed differs from the one proposed in [13], where rather a triangle-based shape-from-silhouette reconstruction method is presented. The direct triangulation technique that they use, therefore, can rely on the available inside-outside information without need for a look-up table that treat all distinct cases in real time as we do.

The octree surface thinning process and the direct triangulation phase are very closely related. As long as the octree cubes are properly eliminated, we obtain triangulations which are almost free of nonmanifold triangles. The rare configurations $C_i, i \geq 6,$ occur mostly when the thinning process is relaxed at low resolutions. This tends to increase the number of ambiguous cases and, thus, the number of internal triangulations which are, however, not visually disturbing. Due to the topological test described in Section 4.3 (see Fig. 4), some nonmanifold triangles which are not internal may also be produced depending on the object complexity and the current resolution level. Such triangles, which are usually very small in number, do not cause visual artifacts when rendered only with color information, but may yield discontinuities in the estimated surface normals.

Once the triangle mesh M^r is constructed from S^r by direct triangulation, the object can be rendered with the color and normal information assigned to mesh vertices via particle attributes. The color within each triangular surface can then be interpolated from the corresponding three vertex colors using Gouraud or Phong shading techniques. The generated meshes, when rendered, produce visually pleasant surfaces with reduced aliasing effects, since the mesh vertices are particle vertices belonging to the object surface.

6.5 Error Analysis

A common metric to measure the error between an original surface $\mathcal{S}$ and its approximation $\mathcal{M}$ is the Hausdorff distance [12], [30] $H(\mathcal{S},\mathcal{M}) = \max(h(\mathcal{S},\mathcal{M}), h(\mathcal{M},\mathcal{S}))$, where $h(A,B) = \max_{a\in A} \min_{b\in B} \|a-b\|$. The function $h(A,B)$ is referred to as the directed Hausdorff metric which measures the distance of the point $a \in A$, which is the farthest from B. The directed Hausdorff metric is not a true distance function, but it provides a good evaluation of the approximated surface with respect to the original.

Recalling that, in our case, the mesh vertices correspond to the particle vertices lying on the object surface and that the triangulation is reduced to three types of triangles as shown in Fig. 5, the directed Hausdorff distance from the mesh M^r to $\mathcal{S}$ is bounded by $h(M^r,\mathcal{S}) \leq \epsilon_r$, where ϵ_r denotes the interior diagonal length of an octree cube at level r. This upper bound can occur only in the worst case where the particles happen to be at the extreme corners of the configuration corresponding to a triangle of type 2 (see Fig. 5) and is given by the radius of the circumcircle of those particle vertices. The validity of this bound for the distance $h(\mathcal{S},M^r)$ from $\mathcal{S}$ to M^r, however, depends on the complexity and the resolution at the current level of detail. In general, the same error bound applies to $h(\mathcal{S},M^r)$ as well, provided that the relatively thin parts of the surface can be differentiated by the current octree resolution r. If not, some long thin linear details may be lost even at the finest resolution level; but, this loss may become really significant, especially at very coarse levels yielding local approximation errors larger than ϵ_r. This can happen either during the octree surface thinning process or at the rendering step when, for instance, an ON node has only one neighboring ON node and, therefore, is not included in any constructed triangle according to the look-up table.

7 Geometry Compression

The object representation described above encodes an object in terms of n octree particles (or vertices), which can then easily be transformed into a mesh of approximately $2n$ triangles. The proposed scheme, therefore, can be regarded as an efficient way of encoding dense uniform meshes in a progressive multilevel form without need of connectivity information. The bitload of this scheme in expressing the whole geometry with $36n$ bits (see Section 5.5), i.e., 18 bits per triangle, can further be reduced since the described representation still contains structural redundancies that can be exploited by improving the particle encoding strategy.

7.1 Hierarchical Particle Ordering

The order of the particles in P^r at each hierarchy level has not been specified in Section 5.3, and can be arbitrary for progressive decoding. This flexibility allows us to further order the particles at each level so that their vertex locations can be encoded relatively to the particle vertices of the previous hierarchy level by using a less number of bits overall. The hierarchical encoding problem can be separated into two parts: encoding of the octree structure, i.e., the node locations, and encoding of the precision, i.e., the particle vertex locations.

Octree encoding consists of arranging all the nodes of the octree hierarchy in the lexicographic order as specified by the octree addressing strategy in Section 4.1. A single byte (8 bits) associated to each parent node at level $r-1$ would then be sufficient to identify its eight child nodes at level r, each bit indicating if the corresponding child is ON or not. We recall from Definition 6, that each octree node is encoded by a particle whose vertex position requires a minimum precision of $N = R$ bits. This particle implicitly identifies at the same time one of the ON child nodes of the underlying parent node, making one of the 8 bits redundant. Therefore, a sequence of 7 bits, instead of a byte, suffices, in fact, to identify the remaining child nodes. When the incoming particles belonging to set P^r are arranged in the same order as the octree nodes, the octree cubes at level r to which they belong can directly be localized with this sequence of 7 bits. The vertex attributes of these particles require only $N-r$ additional bits of

precision per coordinate to achieve a global geometrical precision of N bits, since each vertex is already positioned inside an octree cube of size $2^{N-r} \times 2^{N-r} \times 2^{N-r}$.

The phantom nodes at an intermediate level $r < R$, resulting from the octree surface thinning process, have to be handled specifically (see Fig. 3). Since a phantom node is not associated to a particle, none of its child nodes is implicitly specified. A sequence of 8 bits, instead of 7 bits, is then required to fully identify the location of its ON child nodes. Furthermore, when a parent node has a phantom child node, the presence of this phantom child has to be signaled just as its nonphantom ON child nodes. This phantom child node must further be identified in the associated sequence of 7 or 8 bits since, otherwise, at this stage, the decoder would not know that the encoded child node is a phantom and would then be waiting for a particle, which is not defined for a phantom node. Therefore, a flag of 1 bit must be added to each encoded node to specify whether it corresponds to a phantom or not. The nodes at level R need not be flagged since phantoms at this level have all been discarded from the data structure.

It is possible to obtain a good estimate of the bitload resulting from the hierarchical particle encoding described above. This is achieved throughout some assumptions on the number of eliminated ON nodes and that of phantom nodes. As explained in detail in the appendix, the estimated bitload per particle that we come out with is $5.1 + 3(N - R)$ bits, where N is the required number of precision bits and R is the octree resolution. This means that the whole geometry is encoded per particle by a fixed number of 5.1 bits for the basic octree structure and 3 bits for each additional bit that increases the vertex geometric precision beyond the octree resolution R. This is to be compared with the initial 36 bits per particle without geometry compression. Assuming that the resulting surface triangulation is manifold and that the number of triangles is twice the number of particles, the bitload per triangle for the geometry becomes approximately $2.5 + 1.5(N - R)$.

7.2 Scalability

Scalable coding offers a compromise between visual quality and bit-rate by adjusting the precision of the encoded data, the most significant bits of the model geometry being first selected for transmission. The space partitioning provided by the octree representation facilitates the implementation of such a scalable encoding scheme. With our representation scheme, a precision of R bits is necessary for the decoder to reconstruct the octree and the corresponding mesh structure. The remaining $N - R$ precision bits can be incorporated in different ways, for example, at the end of the bitstream corresponding to each different hierarchy level, or at the end of the whole bitstream after the transmission of the level R. One disadvantage of the second strategy is that at higher levels of detail, staircase effects may appear on the surface. Another possibility is to adjust the precision adaptively by increasing it with one bit each time a new resolution level is transmitted. With most object model geometry and for most visual applications, a precision of 12 bits is sufficient, thus the encoded precision for the intermediate levels can be limited up to 12 bits. The higher precision bits up to N, $N = 16$, for example, can be transmitted at the end if required. A common problem with scalable coders is that reducing the geometrical precision without particular care may cause topological anomalies such as intersecting triangles. In our case, we never have such anomalies as long as the precision is kept equal or higher than the octree resolution, since each vertex location is constrained to lie inside a specified octree cube.

The scalability of our scheme is demonstrated on the Isis statuette in Fig. 10. In this case, an octree resolution $R = 8$ is sufficient and, thus, the minimum number of precision bits necessary to reconstruct the octree structure is $N = 8$. In Fig. 10, by zooming on the right arm of the statuette at level $R = 8$, we observe that this minimum precision yields staircase effects on the model surface. When $N = 12$, it is very difficult to visually distinguish the resulting model from the one with $N = 16$. However, at the intermediate level $r = 6$, a precision of 8 bits seems sufficient. The scalability of our encoding scheme then allows us to adjust the precision depending on the resolution level. For example, the number N of precision bits can be increased from 8, which is acceptable at $r = 6$, to 12 before visualizing the statuette at the maximum resolution level $R = 8$.

8 Experiments

8.1 Construction Techniques

The information needed to construct the octree representation of a 3D object is theoretically provided by an implicit surface $f(x, y, z) = 0$. In practice, the source of this information varies depending upon the application. The object modeling scheme that we propose in this paper is independent of the technique used to obtain this information. However, its performance in terms of object representation and visual quality is highly dependent on the construction method used. We consider here, only two distinct cases on which we demonstrate the proposed scheme. In the first case, the object is constructed by a shape from silhouette technique, whereas in the second one, a uniform dense 3D range data of the object surface is directly available.

8.1.1 Shape from Silhouette

For shape from silhouette construction, we start with 2D photographs of the object pictured from various viewpoints by a calibrated camera [22]. The function $f(x, y, z)$ needed for octree construction is deduced from the object silhouettes [39]. The basic idea relies on the observation that the object volume must lie in the intersection of the visual cones defined by the silhouettes and the optical center of the associated cameras [21]. Our methodology to find this intersection corresponds to carving the bounding box in terms of octree cubes by iteratively excluding the parts falling outside the silhouettes. Once we thereby obtain the initial octree representation, we then determine the particle attributes associated to each leaf node located on the surface. The vertex of a particle is given by the center of gravity of the surface points lying inside the associated cube at the maximum resolution. The normal vector for a given particle is estimated from the local neighborhood information provided by the octree representation, whereas the particle color is determined by projecting the particle vertex onto the image planes aligned with its normal direction [39].

8.1.2 Range Data Modeling

For range data modeling, we start with a uniform dense surface data set of an object. The first step is to construct the octree representation by recursively subdividing the

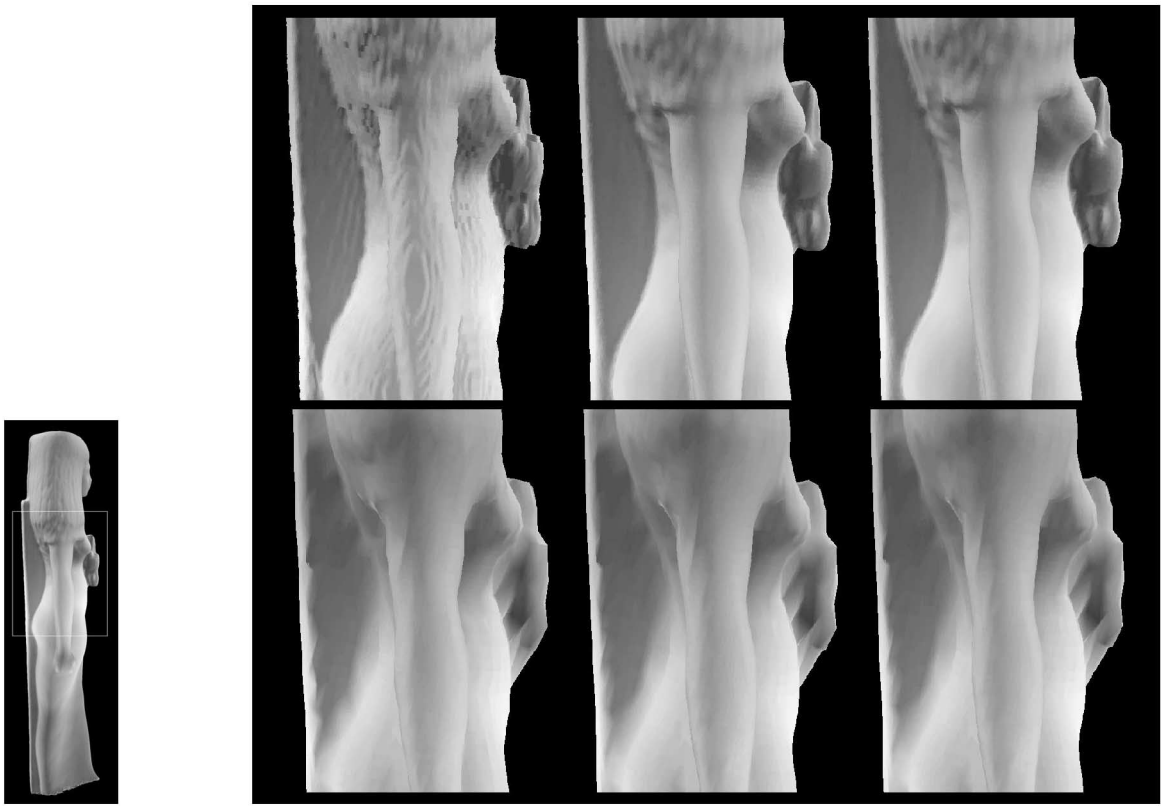


Fig. 10. Zoom on the right arm of the Isis statuette with different precision and resolution levels. From left to right: $N = 8$, $N = 12$, and $N = 16$ at level $R = 8$ (top row), and at the intermediate level $r = 6$ (bottom row).

bounding box so as to identify the ON cubes. The surface points are traversed one by one along the octree paths and the nodes occupied at least by one point are labeled ON. The choice of the finest octree resolution depends on the original resolution of the data or the requirements of the application. Once the octree construction is accomplished up to the target resolution, the normal attribute for each octree cube is determined in a similar way as the shape from silhouette application, whereas the other particle attributes can be computed, for example, by averaging the corresponding attributes of the surface points occupying that cube. As described in Section 4.3, at the octree surface thinning step, we need to know whether a cube that is adjacent to an ON cube is IN or OUT. This information is directly available with volumetric construction methods such as shape from silhouette or when the data is already volumetric such as MRI data. Its availability is more problematic when the data is acquired by a range scanner since, in this case, various difficulties arise such as integration of multiple range images, surface gaps, or range uncertainty. Fortunately, a volumetric approach can handle these problems in a very robust manner [7]. In this work, we consider the representation of sufficiently dense surface data which is free of such problems and which already corresponds to a close surface. The outside-inside information in the vicinity of the surface can then be extracted by tracking the outer or inner boundary of each octree surface. Another possibility to differentiate the outside and the inside in the vicinity of the surface is to use the normal attributes of surface particles if available.

8.2 Results

For range data modeling, three surface data sets without color information have been utilized: the Isis statuette and the hip bone objects,[2] and the Happy Buddha.[3] These data sets are actually surface point clouds that we have obtained by oversampling the vertices of their triangular models. As for the shape from silhouette construction technique, two objects, the Coignard and the Anyi statuettes, have been captured in rotation by a geometrically calibrated camera.

The progressive object modeling scheme is demonstrated in Figs. 11, 12, and 13 for the Coignard and the Isis statuettes, the Anyi statuette and the hip bone, and the Happy Buddha, respectively. The maximum resolution level for each object varies depending on the resolution of the initial data. The choice of the level where the thinning process is relaxed depends mainly on the object complexity and has been determined experimentally.

We verify in Table 1, that the number of triangles at each octree level is about twice the number of ON nodes for the Anyi statuette and the Happy Buddha. The assumptions that we make on the number of eliminated nodes and phantom nodes in the appendix are also verified. These observations are especially valid at the resolutions where the thinning process is strictly applied.

In Table 1, we also present, for the direct triangulation process, the experimental distributions of the different configurations illustrated in Fig. 7. First, we observe that the

2. Cyberware Inc., http://www.cyberware.com.

3. The Stanford 3D Scanning Repository, http://www-graphics.stanford.edu/data/3Dscanrep.

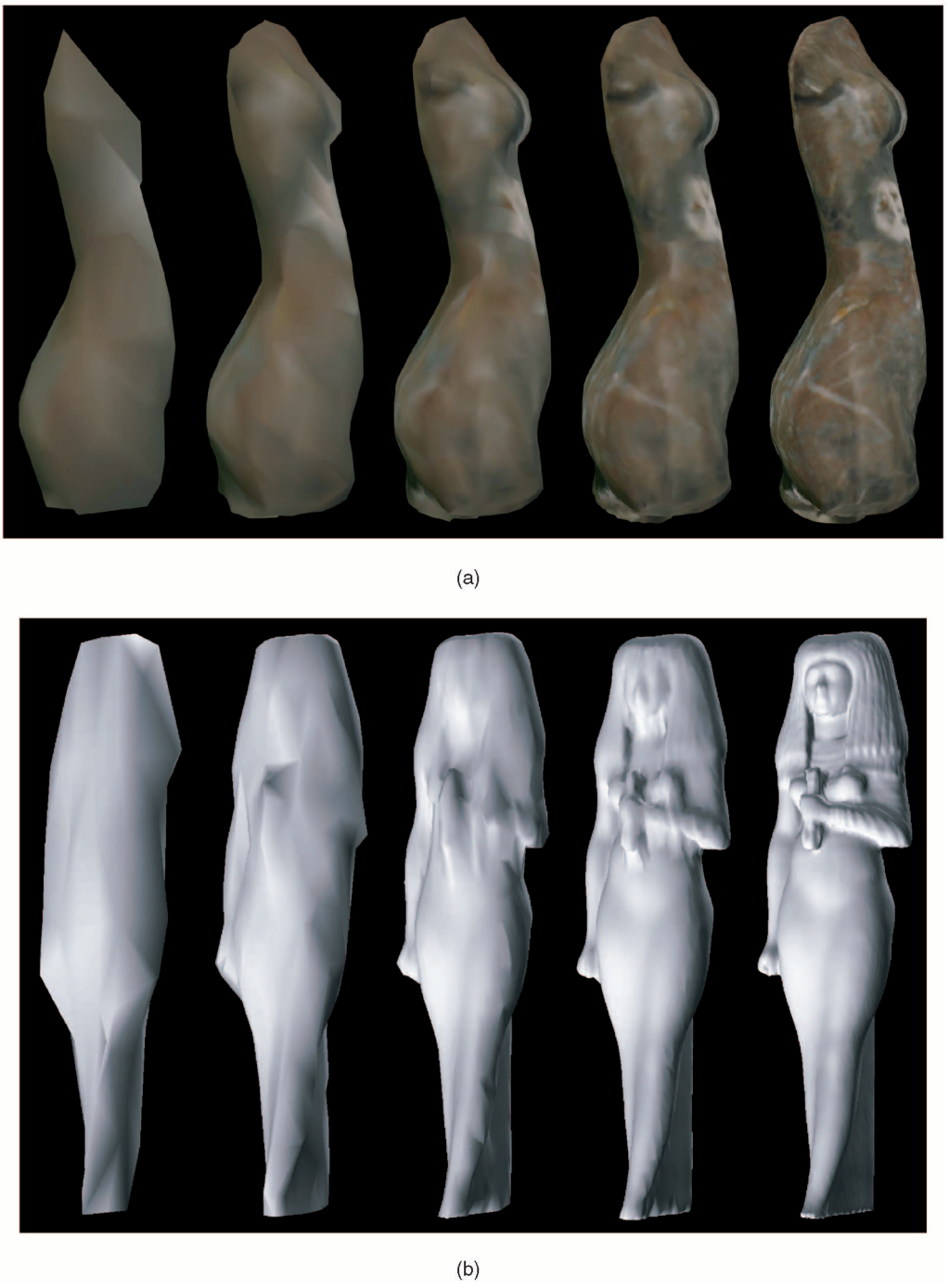


Fig. 11. Progressive representation of (a) the Coignard statuette with color and (b) the Isis statuette surface at increasing levels of detail $r = 4, 5, 6, 7, 9$ and $r = 3, 4, 5, 6, 8$, respectively. The bitload for the whole 12-bit quantized geometry is 13.8 bits per particle (6.90 bits per triangle) and 16.8 bits per particle(8.40 bits per triangle), respectively. At the finest level of detail, the Coignard ($R = 9$) contains 95,683 particles and the Isis ($R = 8$) 100,214 particles.

configurations $C_i, i \leq 5,$ having no ambiguity for triangulation are those encountered most frequently. On the other hand, the configuration C_6 occurs with a rate of about 10 percent and the remaining configurations $C_i, i > 6,$ very rarely depending on the complexity of the object and on the current resolution level. The configuration C_6 is disambiguated in almost all cases, whereas the configurations $C_i, i > 6,$ can be disambiguated with a rate up to 50 percent depending on the object complexity. The final number of resulting nonmanifold local triangulations are also given in Table 1. The nonmanifold triangulations produced are mostly internal triangles except for those that might originate from the topological test that we employ during the thinning process in order to avoid artificial holes (see Section 4.3). The surface normal discontinuities for instance, which can be observed on the reconstructed hip bone at $r = 6$ in Fig. 12, are mainly due to these external nonmanifold triangles.

The implementation of the proposed scheme has been realized by using C++/OpenGL programming languages

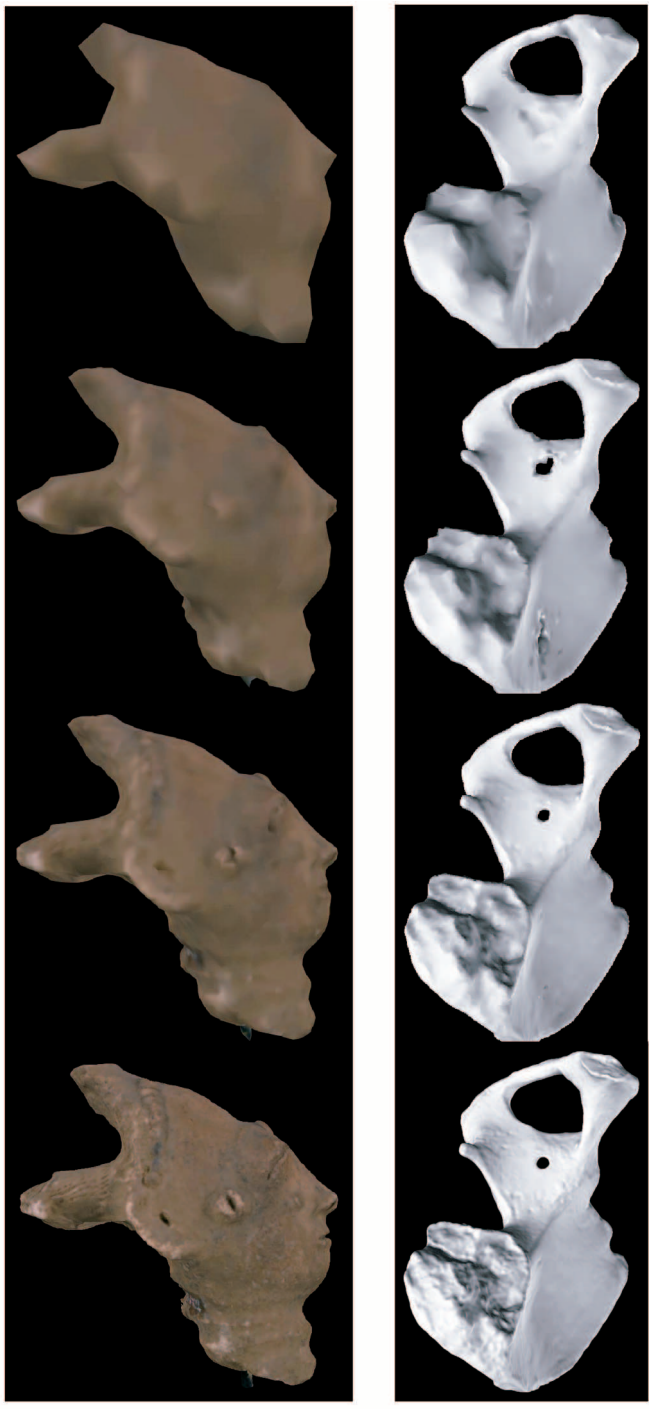


Fig. 12. Progressive representation of (a) the Anyi statuette with color and (b) the hip bone surface at increasing levels of detail $r = 5, 6, 7, 9$ and $r = 5, 6, 7, 8$, respectively. The bitload for the whole 12-bit quantized geometry is 13.9 bits per particle (6.95 bits per triangle) and 17.0 bits per particle (8.50 bits per triangle), respectively. At the finest level of detail, the Anyi ($R = 9$) contains 87,322 particles and the hip bone ($R = 8$) contains 93,611 particles.

and the experiments have been performed on a SUN Ultra-10 (450 MHz) workstation. The resulting computational load of the decoding process is also presented in Table 1 for the Anyi statuette and the Happy Buddha. The given CPU times correspond to the seconds spent at each level of detail during progressive transmission to transform a particle sequence into a mesh representation. The processes of octree construction, surface scanning, and direct triangulation are all included in these values. Although the decoder has not been optimized for speed, the achieved CPU times seem quite reasonable for interactive applications. The extra computational load of the decimation task is very low, necessitating, for instance, at the finest level of detail of the Anyi statuette, 0.5 seconds of CPU time in addition to the value (12 seconds) given in Table 1. All the other results presented in this section have been obtained without decimation.

In our range data modeling applications, once the original mesh is oversampled and the corresponding point clouds are obtained, the processing time needed for the whole construction process is a question of minutes, for example, 1.5 minutes for the Isis and 10.5 minutes for Happy Buddha.

The experimental bitloads per particle necessary to encode the octree structure, i.e., for the case $N = R$, are 4.8, 4.9, 4.8, 5.0, and 4.6 bits for the Coignard, the Anyi, the Isis, the hip bone, and the Happy Buddha, respectively. These values are very coherent with the estimated value given in Section 7.1, the small discrepancies being mainly due to the varying complexities of the objects, more specifically to the varying number of resulting phantom nodes. $3(N - R)$ bits must be added to these values to achieve a higher precision $N > R$. The corresponding bitloads per triangle are almost half of these values.

9 Trade Offs

The comparison of the proposed technique with previous triangle-based progressive approaches should take into account several trade offs between preprocessing time, low resolution approximation quality, storage efficiency, and decoding/rendering performance.

9.1 Preprocessing Time

The efficiency in preprocessing time is one of the main benefits of the proposed scheme, which makes large data sets easy to handle. This is also validated by our experiments. The processing time for large data sets in our case is at the order of minutes, whereas low-speed triangle-based techniques that produce very accurate approximations, require hours of processing time. This is mainly due to the effort that they spend in order to optimize the placement of individual mesh vertices. The runtime performance of our representation scheme is, in fact, at the same level as the fastest triangle-based schemes that sacrifice some of the approximation quality.

9.2 Approximation Quality

Due to the octree surface thinning step, no theoretical error bound is guaranteed for low resolution approximations produced by our technique. This step seems inevitable since we render the encoded surface particles by using triangle meshes. At high resolutions, our technique produces reliable approximations, whereas depending on the object complexity the approximation quality degrades as the resolution gets lower. Triangle-based schemes, on the other hand, result in good quality low resolution models, in proportion to the specific effort spent per primitive.

(a)

(b)

Fig. 13. Progressive representation of the Happy Buda surface, from left to right at increasing levels of detail: (a) The whole object at $r = 7, 8, 9$ and (b) a detail from the zoomed surface at $r = 9$ and $r = 10$. The bitload for the whole 12-bit quantized geometry is 10.6 bits per particle (5.3 bits per triangle).

9.3 Storage Efficiency

The proposed scheme is quite space efficient, resulting in 5 to 9 bits per triangle for representing the whole 12-bit quantized geometry. This is first, due to the fact that connectivity or topology information is not explicitly stored and, second, to the exploitation of the octree hierarchy. Note that no entropy coding has yet been employed for geometry compression. These figures seem to compare favorably with triangle-based techniques. For instance, Pajarola and Rossignac [30] report 3.6 bits per triangle for encoding connectivity and 8.8 bits per triangle for vertex positions with a 12-bit quantized mesh of approximately 100,000 triangles. However, there is once again a trade off here to be taken into account. It is indeed possible that an optimized triangle-based technique may yield a much less number of triangles to express the same (probably simplified) geometry than the proposed technique does. In this case, triangle-based techniques are favorable. However, if the problem is rather representing a dense high resolution geometry with as little information loss as possible, our particle-based representation may be preferable in terms of storage efficiency, even though its low resolution approximations, which are in fact obtained with much less effort, are not as reliable as those resulting from triangle-based techniques.

9.4 Decoding/Rendering Performance

Direct triangulation of encoded octree particles introduces an additional computing overhead to the decoding process. This overhead seems affordable for interactive applications. In cases where this overhead is not tolerable, to accelerate the rendering process, it is also possible to directly visualize the surface particles via splatting techniques [36] without any triangulation, sacrifying, however, from rendering quality. If one wants to avoid the visual artifacts of

TABLE 1
Quantitative Evaluation of the Proposed Scheme for the Anyi Statuette and Happy Buda at Different Resolution Levels r

r	ON nodes k_r	Eliminated nodes	Phantoms m_r	Triangles (number)	Non-manifold (number)	$C_{1,2,3}$ (%)	$C_{4,5}$ (%)	C_6 (%)	$C_{i>6}$ (%)	Time (sec)
Anyi statuette										
4	100	51	50	240	11	57.09	27.64	8.00	7.27	0.01
5	333	247	236	673	7	61.86	27.94	9.79	0.41	0.03
6	1359	1006	874	2714	0	56.65	33.11	10.22	0.03	0.19
7	5434	4184	3069	10868	2	55.50	33.71	10.77	0.02	0.71
8	21724	17289	8337	43464	3	55.18	34.05	10.76	0.02	2.65
9	87322	70974	-	174750	40	54.68	34.72	10.58	0.02	12.08
Happy Buda										
4	114	51	51	295	18	49.47	32.98	3.51	14.04	0.01
5	382	314	313	830	20	54.82	33.27	10.11	1.80	0.04
6	1698	1410	1378	3519	51	54.41	34.59	9.95	1.05	0.18
7	7508	5580	5235	15598	174	52.14	37.33	9.51	1.03	0.83
8	32215	21026	17618	65607	369	50.55	39.39	9.59	0.46	4.06
9	133660	73211	45211	268192	297	49.06	41.22	9.60	0.12	18.06
10	542251	225628	-	1085521	462	48.32	41.76	9.87	0.04	75.70

The first three columns give the number of ON nodes (k_r, after thinning), the number of eliminated nodes, and the number of phantom nodes (m_r). The thinning process has been relaxed at level $r = 4$ for both objects. The total number of triangles resulting from direct triangulation and the number of nonmanifold triangulations are displayed at the fourth and fifth columns. The following four columns give the distribution (in percentage) of the various configuration types $\{C_i\}$. CPU times spent at each level of detail to transform a particle set into a mesh are given at the last column.

splatting, octree particles should be transformed into a triangle mesh and so rendered.

10 Conclusions

The modeling scheme that we have presented encodes the object geometry and the related information with progressive quality. It is especially useful for representing a dense surface data set at its original resolution. The full object information is encoded in such a way that, without increasing the data size and with little effort, the object can progressively be transmitted and viewed at different levels of detail. A user who then has access to the full information, can select the required level of detail in order to find the best compromise between the quality and the time-space efficiency during interactive visualization. Surface particles being rather uniformly distributed over the surface, the number of triangles at the highest level of detail can be large, but in turn, the accuracy of the representation is guaranteed and does not suffer from polygonization artifacts especially on the boundaries. When compared to other progressive modeling techniques, the presented scheme is a robust, simple, and fast technique that can easily be implemented for any complex real object.

The proposed scheme seems quite space efficient for geometry encoding when we consider the bitloads achieved by exploiting only the structural redundancies inherent in the hierarchical representation. Moreover, the resulting scheme is scalable in the sense that the higher geometry precision bits can be encoded independently from the basic octree structure and incorporated at any phase of the visualization or transmission process if required.

The proposed encoding strategy is open to further compression, and we are currently considering the incorporation of entropy coding techniques. When compared to, for example [36], one drawback of our scheme is that it is not view-dependent in its current form. However, our future work will involve adaptation of the proposed scheme to local surface complexity via octree pruning and also to the viewpoint for developing a progressive variable-resolution view-dependent representation. In conjunction with this, we are considering the transmission of triangle textures instead of particle colors. This would increase efficiency, especially when the geometry is already accurate enough at a given level of detail.

Appendix

Based on the description of our hierarchical particle encoder, we now explain how the bitload estimate given in Section 7.1 is analytically obtained. Recall that a flag of 1 bit must be added to each encoded node to specify whether it corresponds to a phantom or not. The total number of flag bits at level r, $r < R$ is the sum of the number of phantom nodes and the number of particles at that level. Let n_r denote the number of particles in set P^r, m_r the number of phantom nodes at level r, and k_r the number of nodes in set S^r. We know that $m_0 = m_R = 0$ and $k_0 = n_0 = 1$. Then, the bitload B_r required to encode the additional geometry with N bits of precision for each level r is given by

$$B_r = \begin{cases} 3(N-r)n_r & \text{if } r = 0, \\ 7k_{r-1} + 8m_{r-1} + n_r + m_r + 3(N-r)n_r & \text{if } 0 < r < R, \\ 7k_{r-1} + 8m_{r-1} + 3(N-r)n_r & \text{if } r = R. \end{cases} \tag{5}$$

The first two terms for the case $0 < r < R$ in (5) correspond to the number of bits necessary to encode the octree structure, i.e., the child nodes at level r with respect to their parents at level $r-1$. The next two terms are the number of flag bits, and the last term gives the number of additional precision bits. Flag bits are not needed at level R. If we sum up the bitloads of all octree levels given in (5), we obtain the total bitload $B = \sum_{r=0}^{R} B_r$:

$$B = 7\sum_{r=0}^{R-1} k_r + 9\sum_{r=1}^{R-1} m_r + \sum_{r=1}^{R-1} n_r + 3\sum_{r=0}^{R}(N-r)n_r. \tag{6}$$

The bitload expressed by (6) can be approximated through some assumptions, which are valid in practice. The basic assumption is on the following relation between the number of nodes at consecutive octree levels: $k_r \approx 4k_{r-1}$, for r sufficiently large and assuming that the object surface is nonfractal. Since, by construction, $n_r = k_r - k_{r-1}$, $n = \sum_r n_r$, and $n = k_R$, i.e., the total number of particles is equal to the number of ON leaf nodes, this assumption yields, furthermore, the following approximations expressing, in terms of n, the number k_r of ON nodes and the number n_r of particles at level r, respectively:

$$k_r \approx \left(\frac{1}{4}\right)^{R-r} n, \tag{7}$$

$$n_r \approx \left(\frac{3}{4}\right)\left(\frac{1}{4}\right)^{R-r} n. \tag{8}$$

Another observation is that the octree surface thinning process eliminates, at most, half of the ON nodes at each level. Since the m_r phantom nodes are only a part of the eliminated nodes, we deduce: $m_r \leq k_r$. For the bitload, the number of phantom nodes is, in the worst case, equal to the number of nodes in set S^r, i.e., $m_r = k_r$. Substituting this together with (8) and (7) into (6), we obtain a good estimate of the bitload for encoding the whole geometry:

$$B \approx \left(\frac{61}{12} + 3(N-R)\right)n. \tag{9}$$

Dividing (9) by the number n of particles, we obtain the following estimate of the bitload per particle: $5.1 + 3(N-R)$, where N is the required number of precision bits and R is the octree resolution.

Acknowledgments

This work has been supported by the European ESPRIT project ACOHIR, and by TUBITAK (Technology and Research Council of Turkey). The authors would like to thank warmly, the artist R. Coignard and the Louvre (Paris) for authorizing us to use the images of the statuettes. They would also like to thank the anonymous reviewers for their constructive and valuable comments.

References

[1] C. Andújar, D. Ayala, P. Brunet, R. Joan-Arinyo, and J. Solé, "Automatic Generation of Multiresolution Boundary Representations," *Proc. Eurographics,* vol. 15, no. 3, pp. 87-96, 1996.

[2] C. Andújar, D. Ayala, and P. Brunet, "Validity-Preserving Simplification of Very Complex Polyhedral Solids," *Virtual Environments,* pp. 1-10, 1999.

[3] H.H. Chen and T.S. Huang, "A Survey of Construction and Manipulation of Octrees," *Computer Vision, Graphics, and Image Processing,* vol. 43, no. 3, pp. 409-431, 1988.

[4] C.H. Chien and J.K. Aggarwal, "Volume/Surface Octrees for the Representation of Three-Dimensional Objects," *Computer Vision Graphics Image Processing,* vol. 36, pp. 256-273, 1986.

[5] A. Ciampalini, P. Cignoni, C. Montani, and R. Scopigno, "Multiresolution Decimation Based on Global Error," *The Visual Computer,* vol. 13, no. 5, pp. 228-246, 1997.

[6] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, and W. Wright, "Simplification Envelopes," *Proc. ACM SIGGRAPH,* pp. 119-128, 1996.

[7] B. Curless and M. Levoy, "A Volumetric Method for Building Complex Models from Range Images," *Proc. ACM SIGGRAPH,* 1996.

[8] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle, "Multiresolution Analysis of Arbitrary Meshes," *Proc. ACM SIGGRAPH,* pp. 173-182, 1995.

[9] J. El-Sana and A. Varshney, "Generalized View-Dependent Simplification," *Proc. Eurographics,* vol. 18, no. 3, pp. 83-94, 1999.

[10] T.A. Funkhouser and C.H. Sequin, "Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Environment," *Proc. ACM SIGGRAPH,* pp. 247-254, 1993.

[11] M. Garland, "Multiresolution Modeling: Survey and Future Opportunities," *Proc. Eurographics,* STAR—State of The Art Reports, 1999.

[12] M. Garland and P.S. Heckbert, "Surface Simplification Using Quadric Error Metrics," *Proc. ACM SIGGRAPH,* pp. 209-224, 1997.

[13] S. Giorgi, F. Pedersini, A. Sarti, and S. Tubaro, "Volume and Surface Reconstruction from Multiple Views," technical report, Dipartimento di Elettronicae Informazione, Politecnico di Milano, http://www-dsp.elet.polimi.it/ispg, 1997.

[14] J. Grossman and W. Dally, "Point Sample Rendering," *Rendering Techniques,* pp. 181-192, July 1998.

[15] R. Grosso and T. Ertl, "Progressive Iso-Surface Extraction from Hierarchical 3D Meshes," *Proc. Eurographics,* vol. 17, no. 3, pp. 126-135, 1998.

[16] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Mesh Optimization," *Proc. ACM SIGGRAPH,* pp. 19-26, 1993.

[17] H. Hoppe, "Progressive Meshes," *Proc. ACM SIGGRAPH,* pp. 99-108, 1996.

[18] H. Hoppe, "View-Dependent Refinement of Progressive Meshes," *Proc. ACM SIGGRAPH,* pp. 189-198, 1997.

[19] A. Khodakovsky, P. Schröder, and W. Sweldens, "Progressive Geometry Compression," *Proc. ACM SIGGRAPH.* pp. 271-278, 2000.

[20] D. Laur and P. Hanrahan, "Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering," *Proc. ACM SIGGRAPH,* pp. 285-288, 1991.

[21] A. Laurentini, "The Visual Hull Concept for Silhouette Based Image Understanding," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 16, no. 2, pp. 150-162, Feb. 1994.

[22] J.M. Lavest, M. Viala, and M. Dhome, "Do We Really Need an Accurate Calibration Pattern to Achieve a Reliable Camera Calibration?" *Proc. Fifth European Conf. Computer Vision,* vol. 1, pp. 158-174, 1998.

[23] A.W.F. Lee, W. Sweldens, P. Schröder, L. Cowsar, and D.P. Dobkin, "MAPS: Multiresolution Adaptive Parameterization of Surfaces," *Proc. ACM SIGGRAPH,* pp. 95-104, 1998.

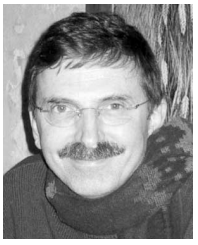
[24] M. Levoy and T. Whitted, "The Use of Points as a Display Primitive," Technical Report TR-85022, Univ. of North Carolina at Chapel Hill, 1985.

[25] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Periera, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk, "The Digital Michelangelo Project: 3D Scanning of Large Statues," *Proc. ACM SIGGRAPH,* pp. 131-144, 2000.

[26] W.E. Lorensen and H.E. Cline "Marching Cubes: A High Resolution 3D Surface Construction Algorithm," *Computer Graphics,* vol. 21, no. 4, pp. 163-169, 1987.

[27] M. Lounsbery, T. DeRose, and J. Warren "Multiresolution Surfaces of Arbitrary Topological Type," *ACM Trans. Graphics,* vol. 16, no. 1, pp. 34-73, 1997.
[28] D. Luebke and C. Erikson, "View-Dependent Simplification of Arbitrary Polygonal Environments," *Proc. ACM SIGGRAPH,* pp. 199-208, 1997.
[29] C. Montani, R. Scateni, and R. Scopigno, "A Modified Look-Up Table for Implicit Disambiguation of Marching Cubes," *The Visual Computer*, vol. 10, pp. 353-355, 1994.
[30] R. Pajarola and J. Rossignac, "Compressed Progressive Meshes," *IEEE Trans. Visualization and Computer Graphics,* vol. 6, no. 1, pp. 79-92, 2000.
[31] H. Pfister, M. Zwicker, J. van Baar, and M. Gross, "Surfels: Surface Elements as Rendering Primitives," *Proc. ACM SIGGRAPH,* pp. 359-376, 1983.
[32] N. Pla-Garcia, "Recovering a Smooth Boundary Representation from an Edge Quadtree and from a Face Octree," *Proc. Eurographics,* vol. 13, no. 4, pp. 189-198, 1994.
[33] J. Popovic and H. Hoppe, "Progressive Simplicial Complexes," *Proc. ACM SIGGRAPH,* pp. 217-224, 1997.
[34] W. Reeves, "Particle Systems—A Technique for Modeling a Class of Fuzzy Objects," *Proc. ACM SIGGRAPH,* pp. 335-342, 2000.
[35] J. Rossignac and P. Borrel, "Multi-Resolution 3D Approximations for Rendering Complex Scenes," *Geometric Modeling in Computer Graphics,* pp. 455-465, 1993.
[36] S. Rusinkiewicz and M. Levoy, "QSplat: A Multiresolution Point Rendering System for Large Meshes," *Proc. ACM SIGGRAPH,* pp. 343-352, 2000.
[37] S. Rusinkiewicz and M. Levoy, "Streaming QSplat: A Viewer for Networked Visualization of Large, Dense Models," *Proc. Symp. Interactive 3D Graphics,* pp. 63-68, 2001.
[38] H. Samet, *Applications of Spatial Data Structures.* Addison-Wesley, 1990.
[39] F. Schmitt and Y. Yemez, "3D Color Object Reconstruction from 2D Image Sequences," *Proc. IEEE Int'l Conf. Image Processing,* vol. 3, pp. 65-69, 1999.
[40] W.J. Schroeder, J.A. Zarge, and W.E. Lorensen, "Decimation of Triangle Meshes," *Proc. ACM SIGGRAPH,* vol. 26, pp. 65-70, 1992.
[41] R. Shekhar, E. Fayad, R. Yagel, and F. Cornhill, "Octree-Based Decimation of Marching Cubes Surfaces," *Proc. Visualization Conf.,* pp. 335-342, Sept. 1996.
[42] R. Szeliski, "Rapid Octree Construction from Image Sequences," *Proc. CVGIP: Image Understanding,* vol. 58, no. 1, pp. 23-32, 1993.
[43] R. Szeliski and D. Tonnesen, "Surface Modeling with Oriented Particle Systems," *Proc. ACM SIGGRAPH,* vol. 26, pp. 185-194, 1992.
[44] G. Taubin, A. Guezic, W.P. Horn, and F. Lazarus, "Progressive Forest Split Compression," *Proc. ACM SIGGRAPH,* pp. 123-132, 1998.
[45] J. Wilhelms and A. Gelder, "Topological Considerations in Isosurface Generation," *Computer Graphics,* vol. 24, no. 5, pp. 57-62, 1990.
[46] J. Wilhelms and A. V. Gelder, "Octrees for Faster Isosurface Generation," *Proc. ACM SIGGRAPH,* vol. 11, no. 3, pp. 201-227, 1992.
[47] Y. Yemez and F. Schmitt, "Progressive Multilevel Meshes from Octree Particles," *Proc. Int'l Conf. 3D Digital Imaging and Modeling,* pp. 290-299, Oct. 1999.

Yücel Yemez received the BS degree from Middle East Technical University, Ankara, Turkey, in 1989, and the MSc and PhD degrees from Bogaziçi University, Istanbul, Turkey, respectively, in 1992 and 1997, all in electrical engineering. From 1997 to 2000, he was a postdoctoral researcher in the Image and Signal Processing Department at Télécom Paris (Ecole Nationale Supérieure des Télécommunications). Currently, he is an assistant professor of the Computer Engineering Department at Koç University, Istanbul, Turkey. His research interests include 3D modeling and visualization, video and image coding, data compression, multiresolution and wavelet analysis, and automated biometrics.



Francis Schmitt received the engineering degree from Ecole Centrale de Lyon, France, in 1973, and the PhD degree in physics from Université Paris 6 in 1979. He has been at Télécom Paris (Ecole Nationale Supérieure des Télécommunications) since 1973, where he is currently a full professor in the Image and Signal Processing Department. His research interests include computer vision, 3D modeling, computational geometry, color picture analysis and synthesis, colorimetry, and multispectral imagery.