A Generic Scheme for Progressive Point Cloud Coding

Yan Huang, Jingliang Peng, C.-C. Jay Kuo, *Fellow*, *IEEE*, and M. Gopi

Abstract—In this paper, we propose a generic point cloud encoder that provides a unified framework for compressing different attributes of point samples corresponding to 3D objects with an arbitrary topology. In the proposed scheme, the coding process is led by an iterative octree cell subdivision of the object space. At each level of subdivision, the positions of point samples are approximated by the geometry centers of all tree-front cells, whereas normals and colors are approximated by their statistical average within each of the tree-front cells. With this framework, we employ attribute-dependent encoding techniques to exploit the different characteristics of various attributes. All of these have led to a significant improvement in the rate-distortion (R-D) performance and a computational advantage over the state of the art. Furthermore, given sufficient levels of octree expansion, normal space partitioning, and resolution of color quantization, the proposed point cloud encoder can be potentially used for lossless coding of 3D point clouds.

Index Terms—Progressive coding, LOD, compression, octree, 3D point cloud.

1 Introduction

Three-dimensional models find applications in many fields such as gaming, animation, and scientific visualization. With the increasing capability of 3D data acquisition devices and computing machines, it is relatively easy to produce digitized 3D models with millions of points. The increase in both availability and complexity of 3D digital models makes it critical to efficiently compress the data so that they can be stored, transmitted, processed, and rendered efficiently.

Traditional polygonal mesh representations of 3D objects require both the geometry and the topology to be specified. In contrast, in point-based 3D model representation, the triangulation overhead is saved, processing and rendering are facilitated without the connectivity constraint, and objects of complex topology can be more easily represented. They make point-based representation an ideal choice in many applications that use high-quality 3D models consisting of millions of points. With such a huge amount of data, efficient compression becomes very important.

The technique of 3D model coding has been studied for more than a decade. When various coding schemes are compared, the compression ratio is the most widely used performance metric. However, in the algorithmic design space of 3D model coding, besides the compression ratio, other parameters such as the following are also important.

One main objective of 3D model compression is to compress models of all different types with various geometry and topological features and various point attributes. Thus, whether a coding scheme can be applied to a large class of models provides a metric for generality measure. Furthermore, end users evaluate a coding scheme based on the decoding efficiency in order to assure the timely reconstruction of the compressed models. This also requires that the decoders are simple to implement. Finally, compression becomes imminent for models with millions of points, whereas memory usage also increases proportionally with such large models. Thus, efficiency in memory usage of the codec becomes another important parameter, based on which the compression scheme has to be evaluated. With these requirements in mind, we propose a 3D point cloud coding scheme that is generic, is time and memory efficient, and achieves a high compression ratio.

1.1 Main Contributions

In this work, we propose a novel scheme for progressive coding of positions, normals, and colors of point samples from 3D objects with arbitrary topology. The major contributions include the following:

- **Generic coder**. It can compress point data for objects with arbitrary topology.
- **Full-range progressive coder.** At the decoder side, a model is progressively reconstructed from a single point to the complete complexity of the original model.
- **Time and space efficiency**. The decoder only needs to maintain partial octree layers and is able to reconstruct/update a model in a time-efficient manner.
- **Efficient attribute coders**. The simple and effective prediction technique in position coding, the progressive quantization and local data reorganization in normal coding, and the adaptive and nonuniform quantization in color coding lead to the superior performance of the proposed scheme.

- *Y. Huang and M. Gopi are with the Department of Computer Science, University of California, Irvine, Irvine, CA 92697. E-mail: {yanh, gopi}@ics.uci.edu.*
- *J. Peng is with the Department of Computer Science, Sun Yat-sen University, Guangzhou 510275, P.R. China. E-mail: jingliap@gmail.com.*
- *C.-C.J. Kuo is with the Ming Hsieh Department of Electrical Engineering and Signal and Image Processing Institute, University of Southern California, Los Angeles, CA 90089-2564. E-mail: cckuo@sipi.usc.edu.*

Manuscript received 14 Nov. 2006; revised 21 June 2007; accepted 23 Sept. 2007; published online 11 Oct. 2007.
Recommended for acceptance by V. Pascucci.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-0203-1106.
Digital Object Identifier no. 10.1109/TVCG.2007.70441.

- **Suitability for lossless coding**.[1] Since no resampling of the input model is done, lossless coding can be potentially achieved.

1.2 Related Work

1.2.1 Mesh Compression

The problem of 3D mesh compression has been extensively studied for more than a decade. For a comprehensive survey of 3D mesh coding techniques, we refer to Peng et al.'s work [1]. Existing 3D mesh coders can be classified into two general categories: single-rate mesh coders [2], [3], [4], [5], [6], [7] and progressive mesh coders [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18]. As compared to single-rate mesh coders, progressive coders allow a mesh to be transmitted and reconstructed in multiple levels of detail (LODs), which is suitable for streaming in networked applications. Most 3D mesh coders handle manifold meshes only, with the exception of those in [13], [14], and [15], which process meshes of arbitrary topology.

1.2.2 Point-Based Model Compression

Similar to mesh coding techniques, most point-based model coders can be classified into single-rate coders [19] and progressive coders [20], [21], [22], [23], [24], [25], [26], [27], [28]. In Krüger et al.'s work [29], although the input model is encoded into multiple LODs, the bitstream of a coarser LOD is not embedded in that of a finer one. Hence, we do not classify it as a progressive coder. Furthermore, some point-based model coders are good for samples from manifold objects only [21], [22], whereas others can handle samples from arbitrary 3D objects [19], [20], [23], [24], [25], [26], [27], [28], [29].

In Gumhold et al.'s work [19], a prediction tree is built up for each input model to facilitate prediction and entropy coding; however, it is not suitable for progressive coding. A bounding-sphere hierarchy is used by the QSplat rendering system developed by Rusinkiewicz and Levoy [20] for interactive rendering of large point-based models. Although not strictly a compression algorithm, QSplat offers a compact representation of the hierarchy structure where 48 bits are used to quantize the position, normal, and color attributes of each node. A multilevel point-based representation is adopted by Fleishman et al. [21], where the coefficient dimension is reduced from 3D to 1D for higher coding efficiency. Techniques of 3D model partitioning and height field conversion are introduced by Ochotta and Saupe [22] so that the 2D wavelet technique can be used to encode the 3D data. Multiple Hexagonal Close-Packing (HCP) grids with decreasing resolutions are constructed by Krüger et al. [29] where sequences of filled cells are extracted and encoded for each HCP grid. An extended edge collapse operator merges two end points of a virtual edge into one point in Wu et al.'s work [23]. The cluster-based hierarchical Principal Component Analysis (PCA) is used by Kalaiah and Varshney [24] to derive an efficient statistical geometry representation. Since the research in [20], [23], [29], and [24] focus on efficient rendering, no rate-distortion (R-D) data of point cloud compression are reported therein.

Among the previous works on point-based model coding, [25], [26], [27], and [28] are the most related to our current work. Waschbüsch et al. [25] used iterative point pair contraction for LOD construction, and the reverse process is encoded. It encodes all point attributes under a common framework. Although this technique is applicable to samples from nonmanifold objects in principle, no such results were presented. Besides, there is a limit on the number of LODs that should be encoded, beyond which the method might show a significant degradation in coding efficiency.

All the coders in [26], [27], and [28] are based on octree-based partitioning of the object space. With a major focus on efficient rendering, Botsch et al. [26] encode only the position data through the coding of bytecodes associated with octree cell subdivisions. Similar to that in Peng and Kuo's work [15], the coder by Schnabel and Klein [27] encodes the number of nonempty child cells and the index of the child cell configuration for each octree cell subdivision. If color attributes are to be coded, it first encodes a color octree and then encodes a color index for each nonempty cell in the position octree. Despite its good R-D performance, Schnabel and Klein's [27] coder may not be generally applicable to real-time decoding due to its computational complexity.

The rest of this paper is organized as follows: Section 2 provides an overview of the proposed coding scheme. The position, the normal, and the color coders are detailed in Sections 3, 4, and 5, respectively. The evaluation of the algorithm on computational and memory efficiency is made in Section 6. A bit allocation strategy is proposed in Section 7. Experimental results are presented in Section 8, and concluding remarks are drawn in Section 9.

2 Overview of the Proposed Coding Scheme

2.1 Constructing the Levels of Detail of the Model

The proposed encoder recursively and uniformly subdivides the smallest axis-aligned bounding box of a given model into eight children in an octree data structure. Only the nonempty child cells will be subdivided further. The part of the model within each cell is represented by its cell's attributes—the position of each cell is represented by the geometric center of the cell, and the normal/color of each cell is set to the average of the normals/colors of contained points. The attributes of nonempty cells in each level in the octree structure yield an LOD of the original 3D model. We call each point in an LOD as a *representative*.

2.2 Coding of Levels of Detail

The efficiency of the proposed coding scheme lies in the effective coding of LODs of the model represented by the octree data structure. In association with each octree cell subdivision, we encode the position, normal, and color attributes of each nonempty child cell.

The position of each cell is implicit as the subdivision of a cell is uniform, and the center of the cell can be computed from the position of the parent cell. Nevertheless, the

1. In the field of model compression, "lossless coding" refers to the coding process that, for a specific quantization of (a floating-point) data, represents and recovers the quantized data in a lossless manner. In essence, the reconstructed data is within a tolerance range from the original input, and there is no resampling of the input data. Specifically, it does not mean that the decoded data is exactly the same as the floating-point input.

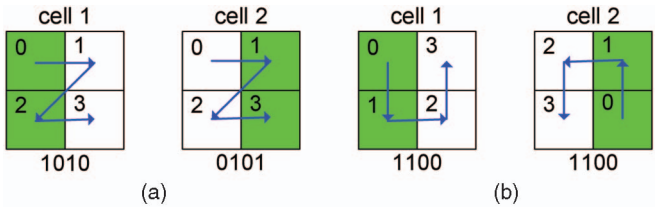

Fig. 1. Examples of occupancy code formation (a) before and (b) after the estimation of each child cell's relative probability of being nonempty, where nonempty and empty child cells are colored green and white, respectively. The traversal orders are denoted by the blue arrows.

sequence of nonempty child cells has to be coded efficiently. The position coder is described in Section 3.

The normal attribute is first quantized based on the uniform subdivision of the unit sphere. When the normal information needs to be refined for an octree cell subdivision, the normals of the children are predicted by the normal of their parent, and their residuals are coded. On the unit sphere, quantized normals around the predicted normal are locally sorted and indexed, resulting in a reduced entropy of normal residual indices. The normal coder is discussed in Section 4.

Before color coding, PCA is first performed on the color data of the model to determine a new color frame where an oriented bounding box of color samples is calculated. Then, the generalized Lloyd algorithm (GLA) is used to calculate the quantization ranges/representatives along each dimension of the oriented bounding box. This adaptive quantization reduces the number of quantization bins (and, thus, the number of representational bits) for a given quantization error threshold. When the color information needs to be refined for an octree cell subdivision, each child color is predicted to be the same as its parent color, and the residual is encoded. The color coder is detailed in Section 5.

3 Position Coder

For each octree cell subdivision, the point representing the parent cell is replaced by points representing nonempty child cells. The decoder needs to know which child cells are nonempty so that a representative can be placed at the geometry center of each nonempty child cell, leading to a finer approximation to the original point cloud model. Our main contribution in position coding is to propose a technique to lower the entropy of codes representing nonempty children using a neighborhood-based predictor.

3.1 Occupancy Code

In the proposed position coder, a 1-bit flag is used to signify whether a child cell is nonempty, with "1" indicating a nonempty child cell and "0" indicating an empty child cell. For each octree cell subdivision, if we traverse all child cells according to a fixed order and collect the flag bits of all child cells, we will obtain an 8-bit code called the *occupancy code*, which has to be coded. For the ease of illustration, we consider a 2D example and show the quadtree subdivision and its occupancy code in Fig. 1. If we traverse child cells according to the fixed order, we will obtain two occupancy codes, 1010 and 0101, for the two cell subdivisions in Figs. 1a and 1b, respectively.

To reduce the entropy of occupancy codes, we "push" "1" bit toward an end by reordering the bits in each occupancy code, as shown in Fig. 1. It is worthwhile to point out that the technique of octree cell subdivision was also used by Peng and Kuo in [14] and [15] for mesh compression. Peng and Kuo [15] encode the index of each nonempty-child-cell tuple instead of the occupancy code. Despite its high coding efficiency, the process of pseudo-probability estimation and tuple sorting is computationally intensive. As compared to the bit reordering technique used by Peng and Kuo [14] for entropy reduction in coding triangular meshes, the occupancy code reordering method described below differs extensively in local neighborhood identification and probability assignment.

3.2 Occupancy Code Reordering

For each cell subdivision, we first estimate each child cell's probability of being nonempty based on the parent cell's local neighborhood. Then, we determine the new traversal order of child cells and reorder bits in the corresponding occupancy code according to the relative magnitude of the estimated probability. The key in occupancy code reordering is probability estimation, which consists of two steps: neighborhood identification and probability assignment.

3.2.1 Neighborhood Identification

In a 3D mesh, an edge indicates the neighbor relationship between two vertices, which was utilized by Peng and Kuo [14] for bit reordering. Since we do not have edges in a point-based 3D model, we call two representatives c_1 and c_2 in the current LOD (and the corresponding octree cells C_1 and C_2) neighbors if and only if the following conditions are satisfied:

- The difference of the level numbers of C_1 and C_2 is less than a predetermined threshold α.
- The distance between c_1 and c_2 is less than $\delta \times \min(diag(C_1), diag(C_2))$, where δ is a constant, and $diag(C_i)$ is the diagonal length of cell C_i.

The first condition requires at most α continuous octree levels, instead of the whole octree, to be maintained during the process of compression. This allows a memory-efficient implementation of the encoder and the decoder. The second condition guarantees that only nearby representatives (that is, cells) could be neighbors, and the range of the local neighborhood is controlled by parameter δ. Interestingly, a similar condition was used by Gopi et al. [30] for the neighborhood identification of points for surface reconstruction. We set $\alpha = 3$ and $\delta = 1.5$ in our experiments. Note that there are data structures and computational geometry algorithms [31] to determine the immediate neighbors of a cell in both complete and incomplete octrees. However, these algorithms are not directly applicable to the current case since we would like to control the extent of the neighborhood using the spatial relationship.

Neighborhood identification can be performed efficiently with the help of the octree structure. To determine the neighbors of a cell after subdivision, we first construct a list of candidate neighbors of the target cell by inheriting the neighborhood relationship from its parent and including all children of the parent's siblings that have been compressed

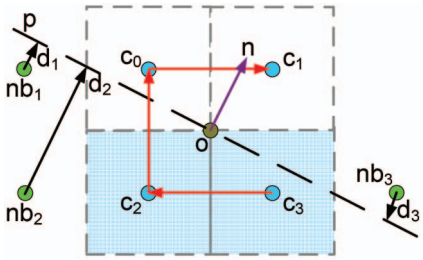

Fig. 2. Determination of the new child cell traversal order based on estimated relative probabilities, where $C_0 \ldots C_3$ are child cells, with C_2 and C_3 being nonempty (filled). The approximate tangent plane p is determined by the normal n at the parent representative o, and nb_i is a neighbor representative whose distance to p is $d_i (i = 1, 2, 3)$. The final order of child cell traversal is shown by the red arrows.

until now. We then prune cells from this list that do not satisfy the above two distance criteria.

3.2.2 Probability Assignment

In general, the local surface around a point in a model lies on one side of its local tangent plane except for saddle and other complex surfaces, and point samples contained in a cell tend to locate closely to the local tangent plane. Based on these observations, we estimate the probability of child cells being nonempty with the following steps:

- At the center of the parent cell, the normal, whose coding will be detailed in Section 4, gives an approximate local tangent plane denoted by p.
- On either side of p, we sum up the distances of neighbor representatives to p and assign higher probability values to child cells whose centers are on the side of p with a higher sum of distances.
- For child cells with centers on the same side of p, higher probability values are assigned to those whose centers are closer to p.

For computational efficiency, we use a plane instead of a higher order surface patch to approximate the local surface.

Being different from the probability assignment in Peng and Kuo's work [14], which orders child cells purely based on their distances to a local approximating surface, our algorithm prioritizes all points on one side of plane p over those on the other. In general, the plane-side-based priority assignment has led to an additional coding gain in our experiments.

3.2.3 Bit Reordering

It is not the exact probability values but their relative magnitudes that matter in the proposed occupancy code reordering algorithm. They guide the child cell traversal and the order of corresponding bits in the occupancy code. Consider the example of a quadtree cell subdivision shown in Fig. 2. The parent representative is shown as o, and its neighbors are given by nb_i. The distances of nb_i to the tangent plane at o given by the normal vector n at o are represented by d_i. The children of o are represented by C_i. Child cells C_2 and C_3 are assigned higher probability values than C_0 and C_1 since $d_1 + d_2 > d_3$. Child cell C_3 is assigned a higher probability value than C_2 since C_3 is closer to the tangent plane than C_2. For the same reason, C_0 has higher probability assignment than C_1. Based on this probability assignment, the order of child cell traversal is changed from $C_0 \rightarrow C_1 \rightarrow C_2 \rightarrow C_3$ to $C_3 \rightarrow C_2 \rightarrow C_0 \rightarrow C_1$, as illustrated by the red arrows. Accordingly, the associated occupancy code is changed from 0011 to 1100, with the 1s being shifted to the left side. Note that this probability estimation algorithm takes into account the local geometry of point-based 3D models implicitly, and it works well for different local curvatures including regions of maxima and minima.

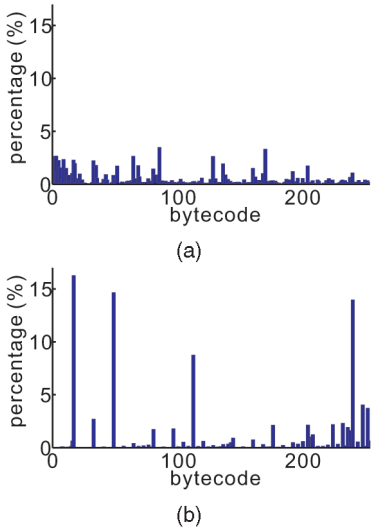

Fig. 3. The distribution of occupancy codes (a) before and (b) after bit reordering.

3.2.4 Effect of Bit Reordering

The effectiveness of the probability estimation and the occupancy code reordering techniques in entropy reduction of occupancy codes is shown in Figs. 3a and 3b. These figures show the histograms of occupancy codes before and after the reordering based on the accumulative statistics for the Octopus model with eight-level octree subdivision. High peaks show up at a few values after reordering, leading to a greatly reduced entropy value of 4.58 from an entropy value of 6.95 before reordering. This method achieves similar entropy reductions in other models also.

4 Normal Coder

The main contribution in normal coding is to rearrange the normal data using a novel local normal indexing scheme that significantly reduces the entropy. The normal of a representative is the normalized average of the normals of all data points contained in the corresponding octree cell. For each cell subdivision, all nonempty child cells are predicted to have the same normal as their parent, and prediction residuals are coded using a local normal indexing scheme that organizes similar normals around the predicted one on the unit sphere (Gauss sphere) into a 1D list.

4.1 Normal Quantization

Before compression, normals need to be quantized. This is achieved by the iterative subdivision of the normal space (that is, the unit Gauss sphere) to a predetermined resolution as done by Taubin et al. [32] and Botsch et al. [26]. Each representative normal can then be identified by an index into

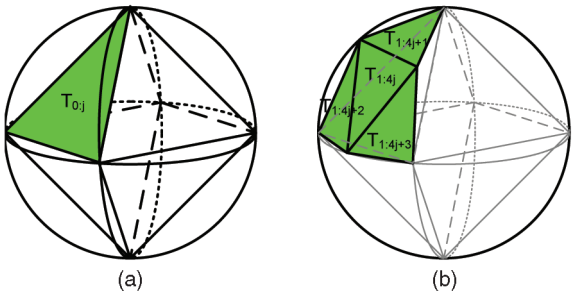

Fig. 4. Normal quantization. (a) An octahedron is inscribed into the unit sphere and its eight facets $T_{0:j}$ $(j = 0, 1, \ldots, 7)$ form the first level of subdivision. (b) Triangle $T_{0:j}$ $(j \in \{0, 1, \ldots, 7\})$ is subdivided into four subtriangles $T_{1:4j} \ldots T_{1:4j+3}$, with index $(1 : 4j)$ assigned to the central subtriangle whose normal is equal to that of $T_{0:j}$.

a table of quantized unit normals determined by the above subdivision. We use the same subdivision and indexing approaches as those used by Botsch et al. [26]. The iterative process of normal space subdivision and indexing is illustrated in Fig. 4.

In terms of R-D performance, it is not meaningful to encode the normal value in high resolution when the positional resolution is still low. Hence, we build up multiple normal tables, one for each level of normal space subdivision, and associate an appropriate resolution-level normal table with each level of the octree in the position coder. In our experiments, a maximum of 13 bits (that is, six levels of sphere subdivision) are used for normal quantization. When an octree cell is subdivided with increased resolution of normal quantization, we need to encode the normal of each child representative. Since in most cases, the normal of a child representative is close to that of the parent, we predict the normal of a child representative to be the same as that of the parent and encode the residual.

4.2 Local Normal Indexing

The proposed normal coder is based on a local normal indexing scheme with an objective to reduce the entropy of normal residuals. For each triangular facet $T_{i:4j}$ at the ith level of normal space subdivision, we reindex the same-level facets in its local neighborhood on the sphere based on the differences in their normal from $T_{i:4j}$. We maintain an array, $A_{i,4j}$, of pointers to these facets in the neighborhood, as shown in Fig. 5. Although we can further expand the local neighborhood, we have already seen very good performance with just three rings in our experiments, and the advantage of having more rings with additional coding bit complexity is negligible. Note that at lower quantization resolutions, the neighborhood may not have enough triangles to have three rings and hence will have fewer rings. The normal space subdivision scheme and the local normal indexing scheme are computed only once and stored as a table for use by both the encoder and the decoder.

Initially, the normal of the root octree cell is represented with a 3-bit global normal index. When an octree cell is subdivided and the associated normal data need to be refined, the indexed local neighborhood facet of the Gauss sphere around the facet of the parent normal in which the normal of the child representative falls is searched. A 1-bit flag is arithmetically encoded to indicate whether this local search is successful. If it is, the local index of the matching normal is arithmetically coded; otherwise, a global search is conducted, and the global normal index is arithmetically coded.

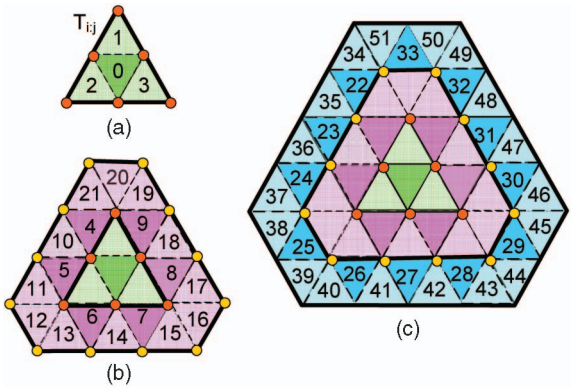

Fig. 5. Local normal indexing. (a) $T_{i:4j} \ldots T_{i:4j+3}$ are assigned the smallest four indices since they have the smallest difference in normal from $T_{i:4j}$. $T_{i:4j+1} \ldots T_{i:4j+3}$ form the first neighbor ring of $T_{i:4j}$. (b) The first neighbor ring of $T_{i:4j}$ is expanded, and the second neighbor ring (in purple and pink) is formed by the triangular facets around the first neighbor ring. Note that the purple facets are assigned smaller indices than the pink ones since their normals are closer to that of $T_{i:4j}$ than those of the pink facet. (c) The local neighborhood is expanded to the third neighbor ring similarly.

In essence, the proposed local normal indexing scheme increases the occurrence frequencies of local normal indices $(0 \ldots 51)$, resulting in a reduced entropy of the normal data. Fig. 6 demonstrates the effectiveness of the local normal indexing scheme. By comparing Figs. 6a and 6b, we see a much more concentrated distribution around a small number of local normal indices in Fig. 6b.

5 Color Coder

Our approach to color data coding is adaptive quantization followed by delta coding. To reduce the resultant data entropy at the same distortion tolerance, the proposed

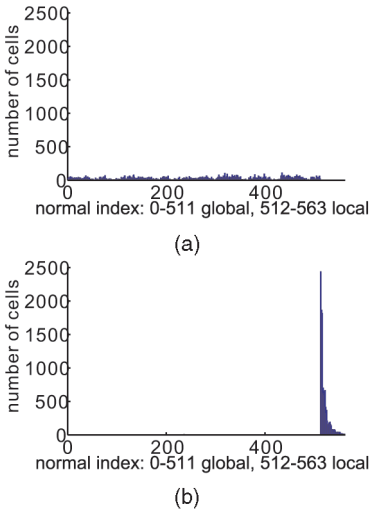

Fig. 6. The distribution of normal indices. (a) Global indices at 9-bit quantization for the Igea model. (b) The corresponding local normal indexing, where the local normal indices are offset by 512 for the purpose of plotting.

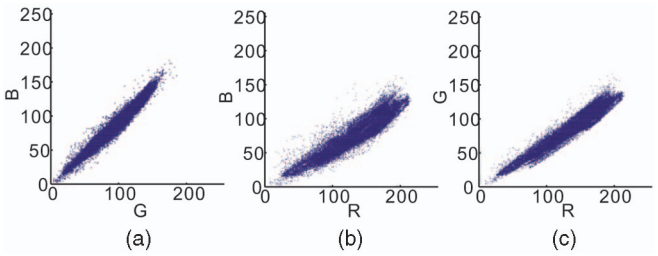

Fig. 7. The distribution of color samples for the Face model viewed from (a) the R-axis, (b) the G-axis, and (c) the B-axis.

quantization scheme utilizes the probabilistic distribution of color samples specific to an input model. As a result, the proposed adaptive color quantization scheme leads to optimized R-D performance when compared with the uniform quantization scheme and the well-known octree-based color quantization scheme by Gervautz and Purgathofer [33]. Finally, delta coding is employed to further reduce the entropy of color data. The proposed adaptive quantization scheme consists of two major components: adaptive color frame determination and adaptive quantization representative/range computation.

5.1 Adaptive Color Frame Determination

A high degree of color correlation exists in a wide range of 3D models, and the color samples of a model tend to cluster in only a small portion of the color space. This often leads to high redundancy in color representation when the uniform RGB space quantization is used. For example, there is a high degree of color sample clustering in the RGB color space for the Face model, as shown in Figs. 7a, 7b, and 7c. This observation generally holds in most models. To exploit this color coherency, we derive a new Cartesian color frame based on the probabilistic distribution of input color data so as to achieve higher representational efficiency. Specifically, PCA is applied to the set of input color samples in the RGB color space. The three orthogonal eigenvectors $\mathbf{V}_1$, $\mathbf{V}_2$, and $\mathbf{V}_3$ identified by PCA and the centroid, $\mathbf{C}$, of the input color samples determine a new Cartesian color frame and is denoted by F'. The oriented bounding box that tightly encloses the color samples in the frame F' is denoted by B', and the one that is defined in F is denoted by B. Typically, the volume of B' is significantly smaller than that of B. For the Face model, whose color data distribution is illustrated in Fig. 7, the volume of B' is only around 15 percent of that of B. In general, such a compact bounding box leads to the reduction of redundancy in the representation.

After the new color frame F' is determined, the coordinates of each color sample in the old RGB color frame F are transformed to the new color frame F'. These transformed coordinates are used to compute B'.

5.2 Adaptive Quantization Range and Representative Calculation

In the new color frame F', we subdivide each dimension of B' into quantization ranges and select a representative for each range. To utilize the probabilistic distribution of color samples, instead of equally subdividing B' along each dimension, we adaptively determine the extent of individual ranges along each dimension of B' such that the average quantization error can be minimized for a given number of quantization ranges. This is done using GLA, which is a clustering algorithm widely used in the context of vector quantization [34] and pattern recognition [35]. (See the Appendix for more information.) After the application of GLA, a sequence of optimal representatives is obtained along each dimension of B', and the set of midpoints between adjacent pairs of representatives delimits the individual quantization ranges.

The number of required ranges in each dimension is determined by the tolerance to the quantization error. In each dimension, the GLA algorithm can be repeatedly applied by adding additional seed representatives at every iteration until the algorithm yields a partition of B' such that the maximum difference between any sample and its representative is within the tolerance. In our experiment, we use 1/32 of the RGB cube's side length as the tolerance level, which has yielded a final color quality that is perceptually indistinguishable from the original in all our test models.

Having determined quantization representatives and ranges along each dimension of B', we construct a color quantization table, which consists of the following data items:

- the origin $\mathbf{C}$ and axes $\mathbf{V}_1$, $\mathbf{V}_2$, and $\mathbf{V}_3$ of the new color frame and
- the following data along each of the above three axes:
  - the value of the first quantization representative and
  - the number of quantization ranges and intervals between every two consecutive quantization representatives.

For time and space efficiency, GLA is conducted along each dimension separately. Please note that running GLA three times, once for each dimension, for 1D ranges of size k each is about three orders of magnitude faster than running it once for 3D cubes of size k^3 that partition the entire 3D bounding box. Furthermore, the separable GLA scheme demands a table consisting of $3k$ 1D representatives, rather than a table consisting of k^3 3D representatives as demanded by the joint GLA scheme.

5.3 Effectiveness of Adaptive Color Quantization

To illustrate the effectiveness of the proposed adaptive color quantization scheme, we use two other color quantization schemes as benchmarks: the uniform quantization scheme that subdivides each dimension of the original RGB color cube into equal-sized ranges and the octree-based color quantization scheme by Gervautz and Purgathofer [33] that adaptively constructs an octree in the original RGB color cube for the quantization purpose.

We plot the estimated R-D curves of three quantization schemes in Fig. 8 with the Face model. The schemes are denoted as "uniform," "octree," and "adaptive" in the figure. The quantization resolution is controlled by the number of quantization ranges along each dimension in the uniform quantization scheme and the proposed adaptive quantization scheme and by the number of quantization representatives in the octree-based scheme. For each quantization resolution, we estimate the corresponding coding bits per input sample based on the entropy of quantized color indices

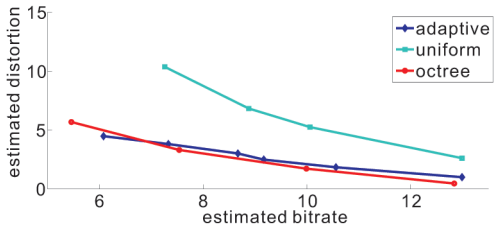

Fig. 8. Comparison of the R-D curves of the three color quantization schemes for the Face model.

and estimate the distortion per input sample by the average distance between each input color sample and its quantized representative. We see in Fig. 8 that the proposed adaptive quantization scheme yields a significant R-D advantage over the uniform quantization scheme.

Although the octree-based color quantization scheme yields almost the same R-D performance as the adaptive color quantization scheme, the memory efficiency of the adaptive scheme can be two orders of magnitude superior to that of the octree-based quantization scheme. For $O(k^3)$ 3D color quantization representatives, the octree-based color quantization scheme requires $O(k^3)$ space to store the color quantization table, whereas the adaptive scheme requires just $O(k)$ space since the quantization representatives are stored independently on each of the three axes of the bounding box.

5.4 Entropy Coding

Since the color decoder requires the color quantization table, the encoder has to encode the table before encoding the colors of representatives in any LOD. In our implementation, the color quantization table is arithmetically coded.

In an intermediate LOD, the color of a representative is quantized in the new color frame F' according to the obtained quantization table, and the quantized color coordinates are to be encoded. Motivated by the observation that there is usually high correlation between colors of a child representative and its parent, a child representative is predicted to have the same color as its parent, and only the residual is coded with an arithmetic coder, leading to further entropy reduction.

Please note that the RGB color representation instead of the luminance-chrominance representation (such as the YUV color space) is used here. Since the luminance-chrominance color representation models human perception of color more closely, it would be interesting to study the quantization scheme in the luminance-chrominance color representation as a future extension. For example, we may apply the adaptive quantization scheme separately to the 1D luminance subspace and the 2D chrominance subspace to prioritize the luminance component over the chrominance components.

6 Evaluation on Time and Space Efficiency

6.1 Asymptotic Performance Analysis

In this section, we conduct an asymptotic performance analysis on the computational and memory costs of the proposed point cloud coder. In order to provide a big-O analysis, we focus on the cost associated with major algorithmic steps and the most expensive operations in each step. Consider the case where there are N points in the input 3D model, which is decoded to a sufficient LOD, namely, $O(N)$ cell subdivisions in total.

6.1.1 Computational Cost

The computational cost for each cell subdivision can be analyzed as follows:

- *Position encoding/decoding*. The neighbor search requires b point-point distance calculations, where b is the size of the candidate neighbor set. The probability assignment entails $b+8$ point-plane distance calculations. The bit reordering process demands at most $8 \times \log_2 8 = 24$ comparisons. Since b is a bounded constant by the definition of a local neighborhood, the computational cost for each position encoding/decoding is $O(1)$.
- *Normal encoding/decoding*. The major cost of normal encoding resides in the normal table search, which is dominated by local normal indexing. For each nonempty child cell, the encoder performs at most 52 (mostly less than 10) calculations and comparisons of the normal difference, and the decoder retrieves the normal vector through a table lookup whose cost is negligible. Thus, the computational cost for each normal encoding/decoding is $O(1)$. Note that the normal quantization table can be built up once and stored for use by any encoder/decoder.
- *Color encoding/decoding*. The encoder performs $3O(\log k)$ searches to quantize each color if the quantization ranges and representatives of each dimension are organized into a binary search tree, where k is the average number of ranges along each dimension. Typically, we have $\log k \leq 8$ for $k \leq 256$. The decoder simply retrieves the quantized color through three table lookups. In addition, both the encoder and the decoder need to perform a color coordinate transformation through a matrix-vector production. The total computational cost of color encoding/decoding is thus $O(1)$.

Besides the computational cost per cell subdivision analyzed above, we need to construct the octree and the color quantization table once at the encoder as a preprocessing step. For a 3D model of N points, building an $(r+1)$-layer octree costs $O(rN)$ in the position encoding, where r is the number of quantization bits along each 1D dimension. For the color encoding, one PCA step and one GLA iteration cost $O(N)$ and $O(kN)$, respectively, where k is the average number of quantization ranges along each dimension in the adaptively determined color frame.

Based on the above analysis, we conclude that the proposed encoding scheme has a computational complexity of $O(\max(r,k)N)$. Based on our experiments, r and k typically take their values from the range of 10-30 in order to produce an approximation to the original 3D model with perceptually indistinguishable quality. We also conclude that the proposed decoding scheme has a computational complexity of $O(N)$, which is much faster than the encoding

scheme. For more detailed timing data, we refer to Table 4 in Section 8.

6.1.2 Memory Cost

Both the encoder and the decoder have to store several tree-front layers of the octree, that is, three tree-front layers of the octree, which takes $O(N)$ space. In addition, the encoder has to store the position, normal, and color attributes associated with each point in the input model, whose memory cost is again $O(N)$. Furthermore, both the encoder and the decoder need to store the normal quantization table and the color quantization table. The normal quantization table takes $O(sM)$ space, where s is the neighborhood size in local normal indexing, which is a constant, and M is the number of distinct normals in the maximum quantization resolution. The color quantization table takes $O(k)$ space. Since $M < N$ and $k < N$, the overall memory cost of the encoder/decoder is $O(N)$.

6.2 Comparison with the Prior Art

In this section, we compare the asymptotic performance of our scheme with the prior art [25], [26], [27]. Since all these works and our work has $O(N)$ space complexity, we just focus on the comparison of computational complexity below.

Botsch et al. [26] only compress the position data with a computational cost of $O(N)$. As compared to our coder, its computational efficiency is slightly better since no prediction is made in the encoding process (at the cost of poorer R-D performance).

Waschbüsch et al. [25] compress all position, normal, and color data. A computational cost of $O(N)$ is needed for the actual encoding/decoding of each attribute. Before the actual coding, the encoder needs to build up a multi-resolution hierarchy through a minimum-weight perfect matching process [36], which demands an extra computational cost of $O(N^2 \log N)$. In addition, both the least squares local plane approximation in position coding and the local coordinate transformation/conversion in attribute coding demand higher complexity than ours.

Schnabel and Klein [27] encode position and color data. The overall coding time is between $O(N \log N)$ and $O(N^2 \log N)$ due to the expensive reordering of tree-front cells in both the position and the color octrees. The prioritization of nonempty-child-cell configurations associated with each cell subdivision step is also computationally intensive. Actually, we observe that the prioritization of nonempty-child-cell configurations and the reordering of tree-front cells are major computational bottlenecks in some octree-based 3D model coders such as those by Schnabel and Klein [27] and Peng and Kuo [15].

7 Discussion on Bit Allocation Strategy

In this section, we consider the bit budget allocation strategy among different types of point attributes for the proposed point cloud coder. Since the coding process is driven by the iterative octree cell subdivision, the positional resolution is always increased at every level of octree expansion. For normal and color coding, we have the flexibility in specifying the octree level at which the points would inherit those attributes from their parents and at which the difference between their attributes should be encoded. Further, since the resolution of normal quantization is progressively refined, we have extra flexibility in specifying the octree level at which the resolution of normal quantization should increase.

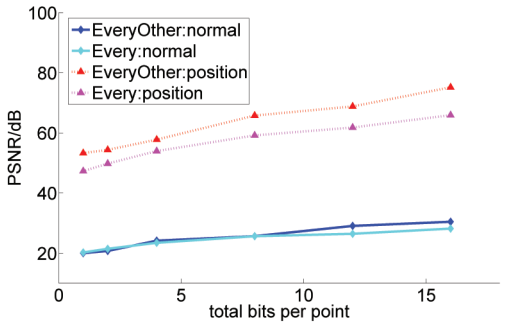

Fig. 9. R-D curves of normal coding for the Dragon model with different bit allocation strategies.

For models without color attributes, better approximation quality of intermediate models has been observed for most of our test models when we encode the normal updates and increment the level of normal space partitioning at every other octree level (rather than at every octree level). This may be due to the fact that the positional accuracy contributes more to the model quality than the normal accuracy when points are densely sampled.

We plot the R-D curves for the Dragon model in Fig. 9 using different bit allocation strategies. Two sets of R-D curves are obtained with two bit allocation strategies: to increase the normal resolution and conduct the normal coding at every level and at every other level, denoted by "Every" and "EveryOther," respectively. The horizontal axis is the total coding bit rate, whereas the vertical axis is the corresponding PSNR values for the position/normal coding. We see in Fig. 9 that the "EveryOther" strategy leads to significantly better position quality at any fixed total bit rate. Interestingly, the "EveryOther" strategy achieves not only higher position quality, but also higher normal quality at relatively high bit rates. This could be explained by the fact that normal quantization reaches its maximum resolution quickly with the "Every" strategy, whereas the position resolution is still relatively low. After that, normal coding is not efficient since normals are already encoded in full resolution, whereas the relatively low position resolution does not help much in providing good normal prediction accuracy.

For models with color attributes, we have an additional flexibility in specifying the bit allocation priority of color coding. It is still an open problem to find the optimal bit allocation among different attributes. Instead of providing an optimal or a suboptimal solution, we only illustrate the effect of different bit allocation strategies on the model quality here.

The reconstructed Santa model at 2.5 bits per point (bpp) with two bit allocation strategies is shown in Fig. 10. Fig. 10a adopts color coding at every level and normal resolution refinement and coding at every other level. Fig. 10b uses color coding at every other level and normal resolution refinement and coding at every level. We see clearly that the

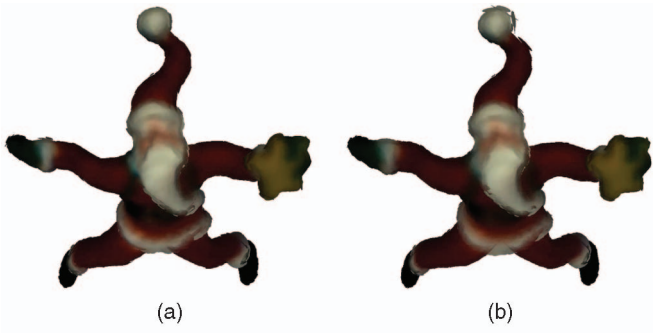

Fig. 10. Visual comparison of the Santa model at 2.5 bpp with two bit allocation strategies.

reconstructed model in Fig. 10a has higher visual quality than that in Fig. 10b, especially in regions around Santa's hat, face, beard, and waist. This could be due to the higher contribution to visual quality of the position's and color's accuracy than that of the normal's accuracy. For example, densely sampled models usually have smooth normal variation among adjacent points over the surface.

Although our experiments suggest that we need a higher resolution for position and color data than that for the normal data during bit allocation, it is worthwhile to study the bit allocation problem and its optimality conditions more thoroughly. In general, we need to estimate the R-D curves for different attributes and measure the relative importance of different attributes accordingly. A preliminary study along this direction was conducted by Li and Kuo in [11].

8 Experimental Results

Nine point-based models are used in our experiments, as shown in Fig. 11. They are Face, Igea, Dragon, and Octopus, which are courtesy of Pointshop 3D; Acer_saccarinum from Xfrog public plants (http://web.inf.tu-dresden.de/ST2/cg/downloads/publicplants/); Dragon-vrip, Santa, Happy Buddha (vripped reconstruction) from the Stanford 3D scanning repository (http://graphics.stanford.edu/data/3Dscanrep/); and FemaleWB from Cyberware (http://www.cyberware.com/). Two versions of dragon models (namely, Dragon and Dragon-vrip) are used in our

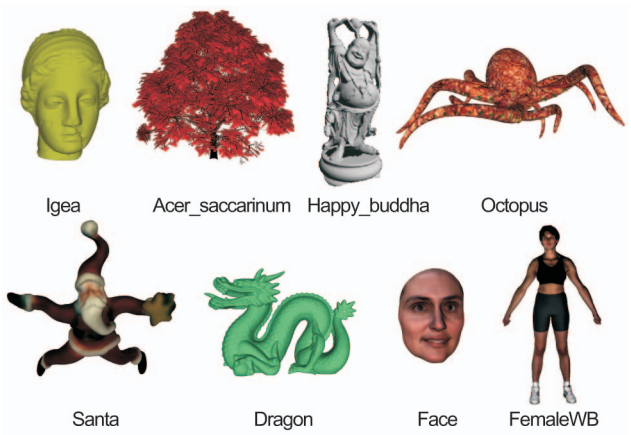

Fig. 11. Models used in our experiments.

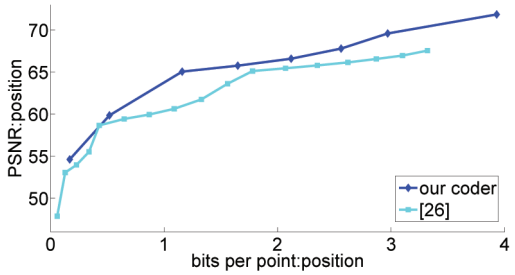

Fig. 12. R-D performance comparison of the proposed coder and that in Botsch et al.'s work [26] for Octopus (466k).

experiments for fair comparison with previous work, although only one dragon model is rendered in Fig. 11. Except for the models provided by Pointshop 3D, all other models were transferred to the Surfel format by ourselves.

8.1 Rate-Distortion Performance Comparison

The coding performance is measured in bits per point, which is the ratio of the total coding bit rate and the number of points in the original model. Although the MLS surface that compares the difference between two point-based models was adopted as the distortion measure by Pauly et al. [37] and Fleishman et al. [21], we do not use the MLS-surface-based distortion metric since it is not suitable for measuring normal and color distortions [25]. Here, we use the peak signal-to-noise ratio (PSNR) to measure position, normal, and color distortions as done by Waschbüsch et al. [25]. The position PSNR is calculated using the euclidean distance between corresponding points in the original and the reconstructed models with the peak signal given by the diagonal length of the tightly fit axis-aligned bounding box of the original model. The normal PSNR is calculated using angles between the original and the reconstructed normals with a peak signal of 180 degrees. The color PSNR is measured separately for each color channel, using the difference between the original and the reconstructed color coordinates with a peak signal of 255 for an 8-bit prequantization of each color channel. We measure the color PSNR in the RGB space except that when the color encoding performance is compared with that of Waschbüsch et al.'s work [25], the color PSNR is measured in the YUV space for fair comparison.

We compare the R-D performance of the proposed point cloud coder with those in [25], [26], and [27], which serve as benchmarks since they too can encode point samples of 3D objects with arbitrary topology progressively. Please note that the coder of Botsch et al. [26] encodes only position data, the coder of Schnabel and Klein [27] encodes both position and color data, and the coder of Waschbüsch et al. [25] encodes all position, normal, and color data.

The R-D performance of the proposed position coder and that of our own implementation of Botsch et al.'s [26] algorithm is compared in Fig. 12. We see that the proposed position coder has 33-50 percent bit rate reduction for PSNR values below 65.

The R-D performance of the proposed progressive coder and that of Waschbüsch et al. [25] for position and normal coding is compared in Figs. 13a and 13b, respectively. The horizontal axis is the coding bit rates, whereas the vertical

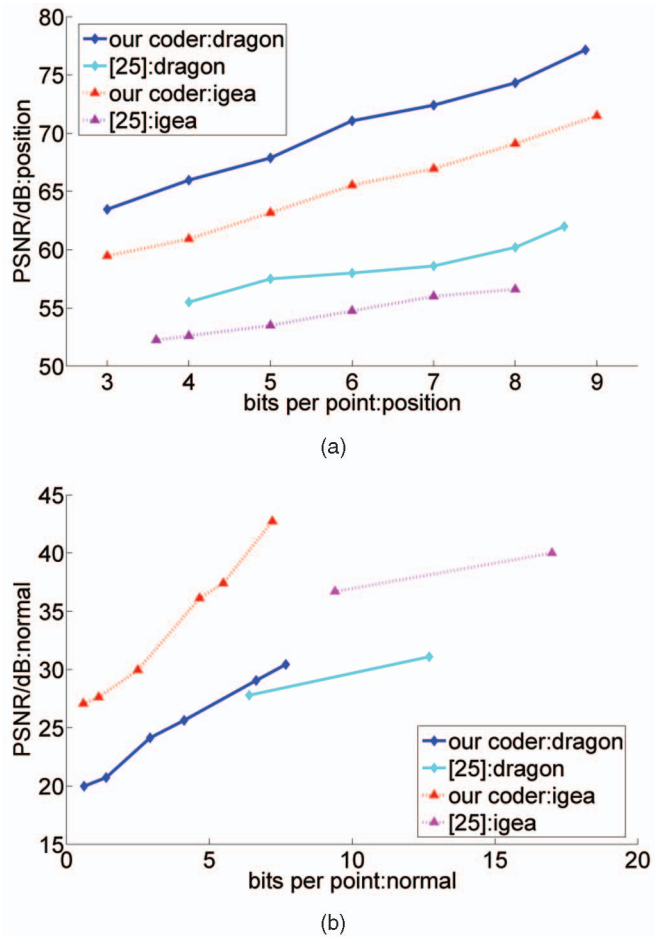

Fig. 13. R-D performance comparison of the proposed coder and that of Waschbüsch et al. [25] for Dragon (436k) and Igea (134k). (a) Position coding. (b) Normal coding.

axis gives the position/normal PSNR values. Note that the R-D data of the position coder of Waschbüsch et al. [25] are taken from the progressive encoding curves in [25, Fig. 10], and the R-D data of normal encoding for [25] are taken from the encoding results in [25, Table 2]. Due to the lack of data, we are not able to make a full-range comparison, especially for normal encoding, as shown in Fig. 13b. As shown in Fig. 13a, the PSNR improvement in position coding is around 10 dB and 15 dB for Igea and Dragon, respectively, at all bit rates. Further, the proposed normal coder can reduce the bit rate by about 50 percent at certain high PSNR values.

Next, we compare the R-D performance with all coding schemes taken into account. The R-D performance of position, normal, and Y-color-component coders for Octopus is shown in Figs. 14a, 14b, and 14c, where the horizontal axes are the total coding bit rates (that is, the sum of position, normal, and color encoding bit rates) in bits per point, and the vertical axes give the corresponding PSNR values. The proposed position and color coders outperform those of Waschbüsch et al. [25] at almost all bit rates with an improvement of up to 9 dB, which roughly corresponds to 65 percent distortion reduction. As compared with the normal coder of Waschbüsch et al. [25], the proposed coder has comparable or better R-D performance for higher bit rates, as shown in Fig. 14b. For lower bit rates, the performance of the proposed normal coder is dictated by the coarse normal quantization resolutions that are adopted in initial octree levels. As the resolution of normal quantization is refined, the normal encoding performance is improved at higher bit rates.

Since Schnabel and Klein [27] do not compress normal data, we compare this work with only the proposed position and color coders in Tables 1 and 2. Table 2 shows only the PSNR value of the B-color-component coding; similar trends are observed for other color components as well. The position coder of Schnabel and Klein [27] achieves higher PSNR values by 4 dB at most bit rates for Dragon-vrip and Igea. In contrast, the proposed color coder outperforms that in [27] by 10 dB or higher at lower bit rates and around 2-6 dB at higher bit rates for Santa and FemaleWB. The R-D data of the coders in [27], shown in Tables 1 and 2, were not reported in the original paper yet kindly provided by the authors for this comparison.

A comprehensive list of R-D data for position, normal, and color coding with the proposed point cloud coder for six test models is given in Table 3, where the bit rate and distortion are reported in bits per point and PSNR, respectively. "N/A" signifies unavailable data because the fully expanded octree in our experiments has 12 levels only, and the final total bit rate of Happy Buddha is less than 16.0 bpp. Due to high randomness in position and normal data and nonmanifoldness, Acer_saccarinum requires more coding bits than other models. Similarly, due to the high variation in the color of Octopus, it requires more bits than other models to represent the color information. To the best

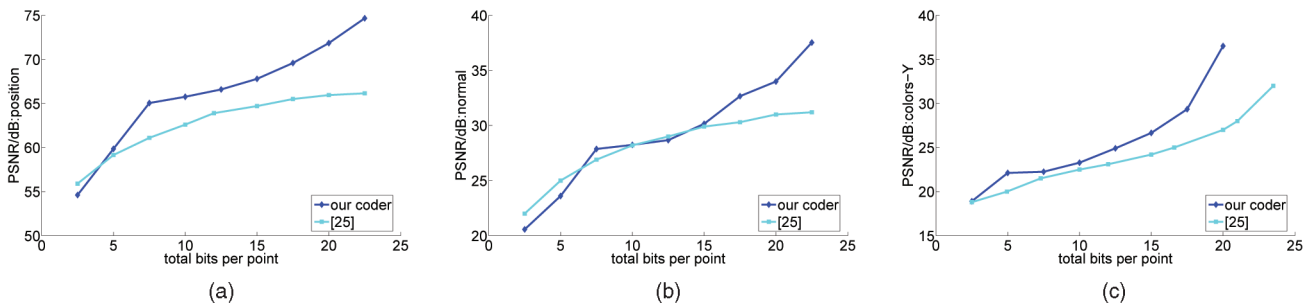

Fig. 14. R-D performance comparison of the proposed progressive coders and those in Waschbüsch et al.'s work [25] for Octopus (466k). (a) Position coding. (b) Normal coding. (c) Y-color-component coding.

TABLE 1
Comparison with Schnabel and Klein's Work [27]: Position Coding

Bitrate(bpp)		0.04	0.15	0.53	1.82	4.15	5.34	8.41
PSNR	our coder	42.16	48.15	53.89	60.10	66.23	68.82	75.59
	[27]	45.86	52.07	58	63.84	68.52	N/A	N/A

(a)

Bitrate(bpp)		0.04	0.43	1.77	4.35	6.45	8.71	11.28
PSNR	our coder	38.59	49.91	55.44	61.59	66.10	71.15	75.15
	[27]	47.79	53.6	59.39	65.4	70.17	76.65	82.21

(b)

(a) Dragon-vrip. (b) Igea.

TABLE 2
Comparison with Schnabel and Klein's Work [27]: Color Coding

Bitrate(bpp)		1.865	2.09	2.75	4.82	5.82
PSNR	our coder	34.88	35.59	38.18	42.08	44.36
	[27]	17.52	26.7	32.35	37.61	N/A

(a)

Bitrate(bpp)		1.022	1.31	2.2	5.36	6.62
PSNR	our coder	26.41	26.91	28.83	38.32	43.72
	[27]	13.96	24.95	28.55	32.63	N/A

(b)

(a) Santa. (b) FemaleFB.

of our knowledge, no other work except that of Huang et al. [28] has ever reported R-D data on highly complex models like Happy Buddha.

Intermediate LODs of Dragon, Octopus, and Acer_saccarinum are shown in Fig. 15 for visual comparison. The same Dragon and Octopus models were considered by Waschbüsch et al. [25] also. The first four rows show the models reconstructed at total bit rates of 2 bpp, 4 bpp, 8 bpp, and 16 bpp, respectively, whereas the last row shows the uncompressed original models. As shown in this figure, a reasonable profile already appears at 2 bpp. We can achieve very decent model quality at 8 bpp. The reconstructed models are almost indistinguishable from the original ones at 16 bpp.

8.2 Computational Complexity Comparison

Another important advantage of the proposed coding scheme is its low computational complexity. In addition to the asymptotic performance analysis given in Section 6, we report the measured timing data for selected models in Table 4. The experiment was conducted on a PC with an Intel Pentium IV 3.20-GHz CPU and 1 Gbyte of RAM. The reported timing data in this table refer to full-resolution encoding/decoding with our unoptimized prototype research code. We see in Table 4 that models with less than a half million points (for example, Igea, Dragon, and Octopus) can be decoded within 10 sec, whereas models with around one million points (for example, Acer_saccarinum and Happy Buddha) may take about 30-40 sec. For very large models such as Acer_saccarinum and Happy Buddha, the increase in decoding time seems to be superlinear with respect to the number of points in the input model, which could be due to the large memory requirement. Typically, the encoding time is several times higher than the decoding time due to the extra cost associated with the maintenance of the original point attribute data and the quantization of normal and color data.

The reader may have noticed that for some models other than Acer_saccarinum and Happy Buddha, the relative magnitudes of the decoding and/or the encoding time may

TABLE 3
R-D Data for Position, Normal, and Color Coding Using the Proposed Point Cloud Coder

Total bitrate	Data type	Position				Normal				Color		
		Dragon (436k)	Acer_saccarinum (889k)	Octopus (466k)	Happy Buddha (1,088k)	Dragon (436k)	Acer_saccarinum (889k)	Octopus (466k)	Happy Buddha (1,088k)	Octopus (466k)	Santa (76k)	FemaleWB (148k)
1.0	R	0.38	0.46	0.08	0.29	0.62	0.54	0.61	0.71	0.30	0.63	0.63
	D	53.28	53.02	53.15	53.88	19.99	12.95	20.08	18.91	17.45	28.28	24.73
2.0	R	0.61	0.69	0.14	0.47	1.39	1.31	0.88	1.53	0.98	1.08	1.1
	D	54.35	53.46	53.99	56.07	20.73	13.23	20.32	21.22	18.34	31.96	26.38
4.0	R	1.07	1.17	0.24	1.86	2.93	2.83	1.55	2.14	2.21	2.66	2.69
	D	57.76	54.51	58.33	63.35	24.13	13.70	22.83	24.52	21.69	36.67	30.38
8.0	R	3.88	2.11	1.24	3.39	4.12	5.89	3.96	4.61	2.79	4.30	4.49
	D	65.78	57.98	65.21	66.84	25.64	14.86	28.09	26.07	22.40	39.95	34.33
12.0	R	5.36	5.45	2.03	4.72	6.64	6.55	4.52	7.28	5.46	5.66	5.88
	D	68.81	65.06	66.40	70.38	29.06	15.07	28.63	28.64	24.55	42.59	36.88
16.0	R	8.32	6.78	2.73	N/A	7.68	9.22	5.43	N/A	7.84	6.84	7.39
	D	75.17	66.78	68.40	N/A	30.44	16.43	30.80	N/A	27.49	46.52	43.83

TABLE 4
Statistics of Encoding/Decoding Time in the Unit of Seconds and the Number in the Parenthesis Refers to the Number of Points in Each Model

	Igea (123K)	Dragon (436k)	Acer_saccarinum (889k)	Happy Buddha (1,088k)	Santa (76k)	FemaleWB (148k)	Octopus (466k)
Encoding	5.11	21.93	162.66	190.58	8.26	13.94	48.76
Decoding	4.03	9.94	33.52	41.90	5.49	5.56	9.69

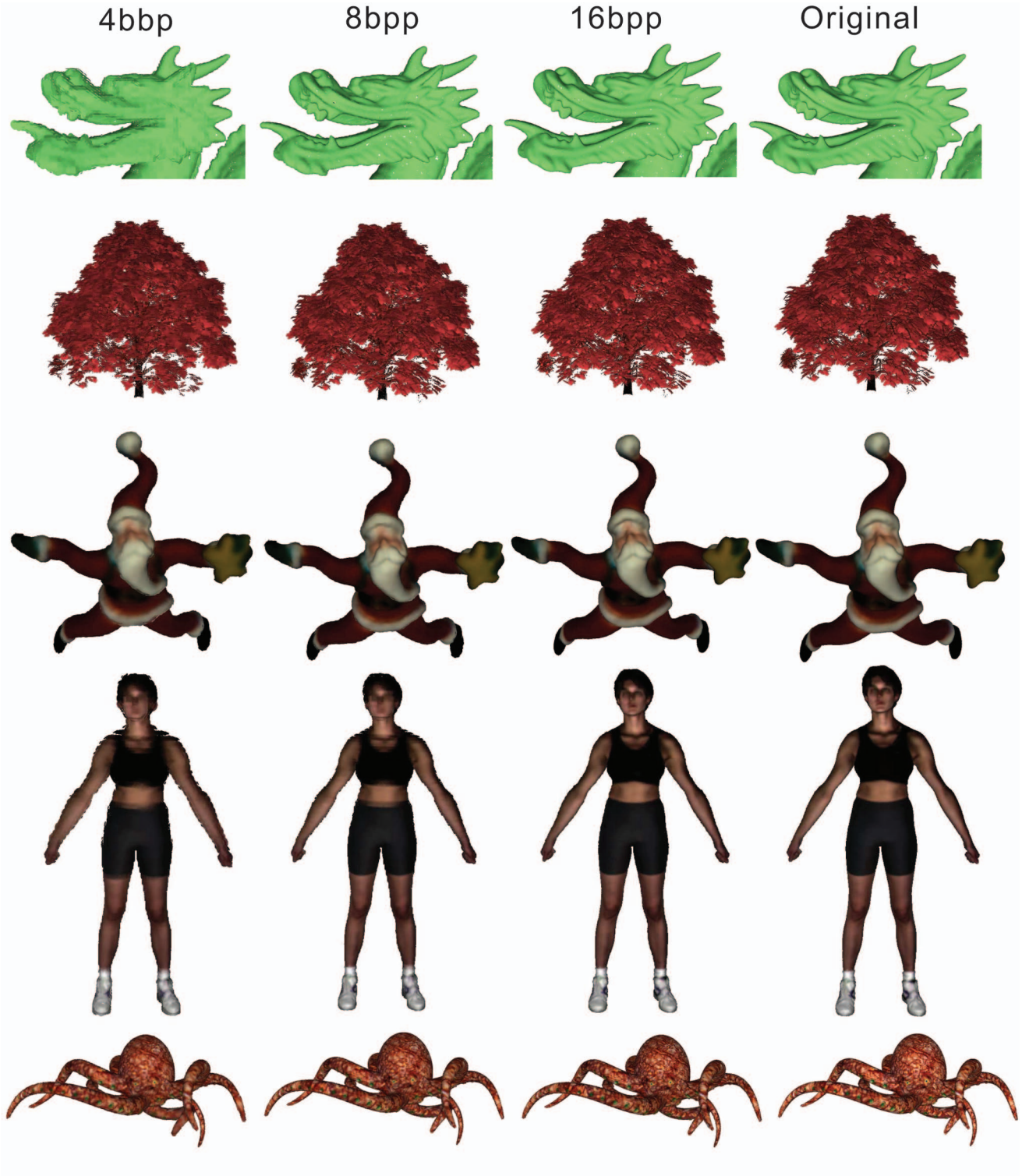

Fig. 15. Models reconstructed at different bit rates.

not be commensurate with the relative magnitudes of the point numbers. Although the number of points in Santa is only about one half of that in Igea, the encoding and the decoding times for Santa are more than those of Igea. The reason is that color data need to be encoded and decoded for Santa but not for Igea, and a significant amount of computation is demanded by the adaptive color quantization in encoding and the color coordinate transformation in decoding. Although there is almost twice the number of points in FemaleWB as in Santa, the decoding time for FemaleWB is comparable with that for Santa. This may be related to the different bit allocation strategies we employ for the two models. For Santa (FemaleWB), normal resolution refinement and coding is performed at every other (every) octree level, whereas color coding is performed at every (every other) octree level. The

reconstruction of a normal vector requires decoding one integral index and looking up a normal table once, while reconstruction of a color vector requires decoding three integral indices, searching three 1D color representative tables, and transforming the color coordinates. This difference in the decoding complexity of normal and color leads to the similar decoding performance of two models with significantly different sizes. As for the encoding time, however, FemaleWB takes significantly more time due to the complexity associated with adaptive color quantization over significantly more points in the color space, when compared with Santa. Although Octopus and Dragon have comparable numbers of points, the encoding time for Octopus is more than twice that of Dragon. This may again be explained by the complexity in adaptive color quantization, since we need to encode the color data for Octopus but not for Dragon. Interestingly, their decoding time is comparable, although more types of attributes are encoded for Octopus. This may be explained by the different point distributions or, in other words, different numbers of octree cell subdivisions to encode/decode in these two models. According to our statistics, 711,658 cell subdivisions are encoded/decoded for Octopus, whereas 1,022,947 cell subdivisions are encoded for Dragon.

9 Conclusion and Future Work

A generic point cloud coder was proposed to encode attributes, including position, normal, and color, of points sampled from 3D objects with arbitrary topology in this work. With novel and effective schemes of quantization, prediction, and data rearrangement, the proposed point cloud coder results in a significant R-D gain and offers a computational advantage over prior art. Another advantage of the proposed point cloud coder is that it does not resample the input model. Thus, it can be potentially used for lossless encoding if the levels of octree expansion and normal space partitioning are sufficiently large and the resolution of color quantization is fine enough.

There are several ways to extend the current research. First, we would like to design more effective predictors for normal and color data. Currently, we predict that each child cell has the same color and normal as its parent. However, a local-neighborhood-based predictor may further improve prediction accuracy. Second, for better color quantization, we may segment all color samples inside the RGB cube into several small clusters analytically and perform adaptive quantization separately for each small cluster for higher efficiency in data representation. Other interesting directions of future work include analytical bit allocation, view-dependent 3D rendering, out-of-core compression of gigantic point clouds, and efficient decoding and rendering implementation on GPUs.

Appendix

The Generalized Lloyd Algorithm

GLA is employed to calculate the quantization ranges and representatives along each dimension of the axis-aligned bounding box in the new color frame F' such that the overall quantization error is minimized along each dimension. For a given set of 1D color coordinates $S = \{s_1, s_2, \ldots, s_N\}$, if k quantization representatives are to be calculated, GLA can be stated as follows:

1. **Initialization**. Select randomly an initial representative set $R_0 = \{r_1, r_2, \ldots, r_k\}$ from S.
2. **Iterative partition**. For $l = 1, 2, \cdots$, we perform the following:
   - Partition S into nonoverlapping subsets $P_1, P_2, \ldots, P_k$ using the nearest neighbor rule; namely, $S = \bigcup_{i\in\{1,2,\ldots,k\}} P_i$, $P_i \cap P_j = \emptyset$ for all $i \neq j$, and $P_i = \{s | d(s, r_i) \leq d(s, r_j), \forall 1 \leq j \leq k, s \in S\}$, where $d(.)$ is a distance metric.
   - Compute the new centroid r_i from all coordinates in P_i, $1 \leq i \leq k$, update representative set R_l with the k new centroids, and calculate the distortion:

$$E_l = \frac{1}{n}\Sigma_{i=1}^{k}\Sigma_{p\in P_i} d(p, r_i).$$

3. **Stopping criterion**. The above iteration stops if either $(E_{l-1} - E_l)/E_l < \delta$ or $l = L$, where δ and L are design parameters. R_l gives the final set of representatives.

Acknowledgments

The authors would like to thank M. Waschbüsch for patiently answering their questions about the work in [25] and R. Schnabel for generously sharing the experimental data that are not available in [27].

References

[1] J. Peng, C.-S. Kim, and C.-C.J. Kuo, "Technologies for 3D Mesh Compression: A Survey," *J. Visual Comm. and Image Representation,* vol. 16, no. 6, pp. 688-733, 2005.
[2] G. Taubin and J. Rossignac, "Geometric Compression through Topological Surgery," *ACM Trans. Graphics,* vol. 17, no. 2, pp. 84-115, 1998.
[3] C.L. Bajaj, V. Pascucci, and G. Zhuang, "Single Resolution Compression of Arbitrary Triangular Meshes with Properties," *Computational Geometry: Theory and Applications,* vol. 14, pp. 167-186, 1999.
[4] C. Touma and C. Gotsman, "Triangle Mesh Compression," *Proc. Graphics Interface Conf. (GI '98),* pp. 26-34, 1998.
[5] P. Alliez and M. Desbrun, "Valence-Driven Connectivity Encoding for 3D Meshes," *Proc. Eurographics '01,* pp. 480-489, 2001.
[6] S. Gumhold and W. Straßer, "Real Time Compression of Triangle Mesh Connectivity," *Proc. ACM SIGGRAPH '98,* pp. 133-140, 1998.
[7] J. Rossignac, "Edgebreaker: Connectivity Compression for Triangle Meshes," *IEEE Trans. Visualization and Computer Graphics,* vol. 5, no. 1, pp. 47-61, Jan.-Mar. 1999.
[8] H. Hoppe, "Progressive Meshes," *Proc. ACM SIGGRAPH '96,* pp. 99-108, 1996.
[9] D. Cohen-Or, D. Levin, and O. Remez, "Progressive Compression of Arbitrary Triangular Meshes," *Proc. IEEE Visualization Conf. (VIS '99),* pp. 67-72, 1999.
[10] P. Alliez and M. Desbrun, "Progressive Encoding for Lossless Transmission of Triangle Meshes," *Proc. ACM SIGGRAPH '01,* pp. 198-205, 2001.
[11] J. Li and C.-C.J. Kuo, "Progressive Coding of 3-D Graphic Models," *Proc. IEEE,* vol. 86, no. 6, pp. 1052-1063, June 1998.
[12] C. Bajaj, V. Pascucci, and G. Zhuang, "Progressive Compression and Transmission of Arbitrary Triangular Meshes," *Proc. IEEE Visualization Conf. (VIS '99),* pp. 307-316, 1999.

[13] P.M. Gandoin and O. Devillers, "Progressive Lossless Compression of Arbitrary Simplicial Complexes," *ACM Trans. Graphics,* vol. 21, no. 3, pp. 372-379, 2002.
[14] J. Peng and C.-C.J. Kuo, "Progressive Geometry Encoder Using Octree-Based Space Partitioning," *Proc. IEEE Int'l Conf. Multimedia and Expo (ICME '04),* pp. 1-4, 2004.
[15] J. Peng and C.-C.J. Kuo, "Geometry-Guided Progressive Lossless 3D Mesh Coding with Octree (OT) Decomposition," *Proc. ACM SIGGRAPH '05,* pp. 609-616, 2005.
[16] Z. Karni and C. Gotsman, "Spectral Compression of Mesh Geometry," *Proc. ACM SIGGRAPH '00,* pp. 279-286, 2000.
[17] A. Khodakovsky, P. Schröder, and W. Sweldens, "Progressive Geometry Compression," *Proc. ACM SIGGRAPH '00,* pp. 271-278, 2000.
[18] A. Khodakovsky and I. Guskov, "Compression of Normal Meshes," *Geometric Modeling for Scientific Visualization,* G. Brunnett, B. Hamann, H. Müller, and L. Linsen, eds., pp. 189-206, Springer, 2002.
[19] S. Gumhold, Z. Karni, M. Isenburg, and H.-P. Seidel, "Predictive Point-Cloud Compression," *SIGGRAPH Sketches,* 2005.
[20] S. Rusinkiewicz and M. Levoy, "QSplat: A Multiresolution Point Rendering System for Large Meshes," *Proc. ACM SIGGRAPH '00,* pp. 343-352, 2000.
[21] S. Fleishman, D. Cohen-Or, M. Alexa, and C.T. Silva, "Progressive Point Set Surfaces," *ACM Trans. Graphics,* vol. 22, no. 4, pp. 997-1011, 2003.
[22] T. Ochotta and D. Saupe, "Compression of Point-Based 3D Models by Shape-Adaptive Wavelet Coding of Multi-Height Fields," *Proc. IEEE/Eurographics Symp. Point-Based Graphics (PBG '04),* pp. 103-112, 2004.
[23] J. Wu, Z. Zhang, and L. Kobbelt, "Progressive Splatting," *Proc. IEEE/Eurographics Symp. Point-Based Graphics (PBG '05),* pp. 25-32, 2005.
[24] A. Kalaiah and A. Varshney, "Statistical Geometry Representation for Efficient Transmission and Rendering," *ACM Trans. Graphics,* vol. 24, no. 2, pp. 348-373, 2005.
[25] M. Waschbüsch, M. Gross, F. Eberhard, E. Lamboray, and S. Würmlin, "Progressive Compression of Point-Sampled Models," *Proc. IEEE/Eurographics Symp. Point-Based Graphics (PBG '04),* 2004.
[26] M. Botsch, A. Wiratanaya, and L. Kobbelt, "Efficient High Quality Rendering of Point Sampled Geometry," *Proc. 13th Eurographics Workshop Rendering (EGWR '02),* pp. 53-64, 2002.
[27] R. Schnabel and R. Klein, "Octree-Based Point Cloud Compression," *Proc. IEEE/Eurographics Symp. Point-Based Graphics (PBG '06),* pp. 111-120, 2006.
[28] Y. Huang, J. Peng, C.-C.J. Kuo, and M. Gopi, "Octree-Based Progressive Geometry Coding of Point Clouds," *Proc. IEEE/Eurographics Symp. Point-Based Graphics (PBG '06),* pp. 103-110, 2006.
[29] J. Krüger, J. Schneider, and R. Westermann, "Duodecim—A Structure for Point Scan Compression and Rendering," *Proc. IEEE/Eurographics Symp. Point-Based Graphics (PBG '05),* pp. 99-107, 2005.
[30] M. Gopi, S. Krishnan, and C. Silva, "Surface Reconstruction Using Lower Dimensional Localized Delaunay Triangulation," *Proc. Eurographics '00,* vol. 19, no. 3 pp. 467-478, 2000.
[31] M.D. Berg, M.V. Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications.* Springer, 1998.
[32] G. Taubin, W. Horn, F. Lazarus, and J. Rossignac, "Geometry Coding and VRML," *Proc. IEEE,* vol. 96, no. 6, pp. 1228-1243, June 1998.
[33] M. Gervautz and W. Purgathofer, "A Simple Method for Color Quantization: Octree Quantization," *Graphics Gems I,* pp. 287-293, 1990.
[34] Y. Linde, A. Buzo, and R.M. Gray, "An Algorithm for Vector Quantizer Design," *IEEE Trans. Comm.,* vol. 28, no. 1, pp. 84-95, 1980.
[35] S. Theodoridis and K. Koutroumbas, *Pattern Recognition.* Academic Press, 1999.
[36] L. Lovasz and M.D. Plummer, *Matching Theory.* Elsevier Science, 1986.
[37] M. Pauly, M. Gross, and L. Kobbelt, "Efficient Simplification of Point-Sampled Surfaces," *Proc. IEEE Visualization Conf. (VIS '02),* pp. 163-170, 2002.

Yan Huang received the BS degree in computer science from Beijing University, China, in 1997 and the MS degree from the University of California, Irvine, in 2003. She is a PhD student in the Department of Computer Science, University of California, Irvine. Her current research interests include 3D data processing, interactive walkthrough applications, and real-time rendering.

Jingliang Peng received the PhD degree in electrical engineering from the University of Southern California in 2006, the BS and MS degrees in computer science from Peking University in 1997 and 2000, respectively. Currently, he is with the Department of Computer Science, Sun Yat-sen University, China, as an associate professor. His research topics include 3D graphics data compression, digital geometry processing, 3D shape analysis, enhanced reality, and medical imaging.

C.-C. Jay Kuo received the BS degree in electrical engineering from the National Taiwan University, Taipei, in 1980 and the MS and PhD degrees in electrical engineering from the Massachusetts Institute of Technology, Cambridge, in 1985 and 1987, respectively. He is the director of the Signal and Image Processing Institute (SIPI) and a professor of electrical engineering, computer science, and mathematics in the Ming Hsieh Department of Electrical Engineering, University of Southern California (USC). His research interests are in the areas of digital image/video analysis and modeling, multimedia data compression, communication and networking, and biological signal/image processing. He is a coauthor of about 140 journal papers, 730 conference papers, and nine books. He is the editor in chief of the *Journal of Visual Communication and Image Representation* and an editor of the *Journal of Information Science and Engineering*, *LNCS Transactions on Data Hiding and Multimedia Security*, and *EURASIP Journal of Applied Signal Processing*. He was on the editorial board of the *IEEE Signal Processing Magazine* from 2003 to 2004. He served as associate editor for the *IEEE Transactions on Image Processing* from 1995 to 1998, *IEEE Transactions on Circuits and Systems for Video Technology* from 1995 to 1997, and *IEEE Transactions on Speech and Audio Processing* from 2001 to 2003. He received the National Science Foundation Young Investigator (NYI) Award and Presidential Faculty Fellow (PFF) Award in 1992 and 1993, respectively. He received the Northrop Junior Faculty Research Award from the USC Viterbi School of Engineering in 1994. He received the best paper award from the Multimedia Communication Technical Committee of the IEEE Communications Society in 2005. He was an IEEE Signal Processing Society distinguished lecturer in 2006. He is also an advisor to the Society of Motion Picture Television Engineers (SMPTE)—USC student chapter. He is a fellow of the IEEE and the SPIE and a member of the ACM.

M. Gopi received the BE degree from the Thiagarajar College of Engineering, Madurai, India, in 1992, the MS degree from the Indian Institute of Science, Bangalore, in 1995, and the PhD degree from the University of North Carolina, Chapel Hill, in 2001. He is an assistant professor in the Department of Computer Science, University of California, Irvine. He has worked on various geometric and topological problems in computer graphics. His current research interest focuses on graph algorithms for geometry processing, geometry and topology compression, and sketch-based modeling.