A Generic Scheme for Progressive Point Cloud Coding

Yan Huang, Jingliang Peng, C.-C. Jay Kuo 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 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, positions of point samples are approximated by the geometry centers of all tree-front cells while normals and colors are approximated by their statistical average within each of tree-front cells. With this framework, we employ attribute-dependent encoding techniques to exploit different characteristics of various attributes. All of these have led to 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.**

I. Introduction

3D 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 representation of 3D objects require both geometry and 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 the 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 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, while 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, time and memory efficient, and achieves a high compression ratio.

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

Yan Huang and M. Gopi are with Department of Computer Science, University of California, Irvine, CA 92697, USA. E-mails: yanh@ics.uci.edu and gopi@ics.uci.edu

Jingliang Peng is with Department of Computer Science, Sun Yat-sen University, Guangzhou 510275, P. R. China. E-mail: jingliap@gmail.com

C.-C. Jay Kuo is with Ming Hsieh Department of Electrical Engineering and Signal and Image Processing Institute, University of Southern California, Los Angeles, CA 90089-2564, USA. E-mail: cckuo@sipi.usc.edu

[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.

B. Related Work

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]. The existing 3D mesh coders can be classified into two general categories: single-rate mesh coders [2]–[7] and progressive mesh coders [8]–[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 of 3D mesh coders handle manifold meshes only, with exceptions of [13]–[15] which process meshes of arbitrary topology.

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]–[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]) while others can handle samples from arbitrary 3D objects ([19], [20], [23]–[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 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]–[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 non-manifold 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]–[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 byte codes 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 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. Sec. II provides an overview of the proposed coding scheme. The position, the normal and the color coders are detailed in Secs. III, IV and V, respectively. Evaluation of the algorithm on computational and memory efficiency is made in Sec. VI. A bit allocation strategy is proposed in Sec. VII. Experimental results are presented in Sec. VIII, and concluding remarks are drawn in Sec. IX.

II. Overview of Proposed Coding Scheme

Constructing LODs of the Model: The proposed encoder recursively and uniformly subdivides the smallest axis-aligned bounding box of a given model into eight children in a 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 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*.

Coding of LODs: The efficiency of the proposed coding scheme lies in effective coding of LODs of the model represented by the octree data structure. In association with each octree cell subdivision, we encode 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 sequence of nonempty child cells has to be coded efficiently. The position coder is described in Sec. III.

The normal attribute is first quantized based on uniform subdivision of the unit sphere. When the normal information needs to be refined for an octree cell subdivision, normals of 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 Sec. IV.

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 (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 Sec. V.

III. 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.

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' 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 2-D 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. 1(a) and (b), respectively.

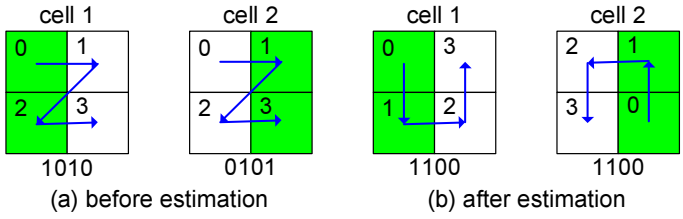


Fig. 1. Examples of occupancy code formation (a) before and (b) after estimation of each child cell's relative probability of being nonempty, where nonempty and empty child cells are colored green and white, respectively. The traversal orders are denoted by the blue arrows.

To reduce the entropy of occupancy codes, we "push" '1'-bits 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 [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.

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.

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 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 (*i.e.*, cells) could be neighbors, and the range of local neighborhood is controlled by parameter δ. Interestingly, a similar condition was used by Gopi *et al.* [30] for 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 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 parent's siblings that have been compressed until now. We then prune cells from this list that do not satisfy the above two distance criteria.

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 Sec. IV, gives an approximate local tangent plane denoted by p.
- On either side of p, we sum up 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.

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. The 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 red arrows. Accordingly, the associated occupancy code is changed from 0011 to 1100, with 1's 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.

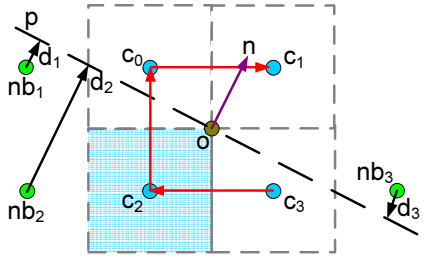


Fig. 2. Determination of the new child cell traversal order based on estimated relative probabilities, where $C_0 \ldots C_3$ are child cells, with C_2 and C_3 nonempty (filled). The approximate tangent plane p is determined by the normal n at the parent representative o and nb_i is a neighbor representative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

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 Figures 3(a) and (b). 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 the reordering, leading to a greatly reduced entropy value of 4.58 from the entropy of 6.95 before reordering. This method achieves similar entropy reductions in other models also.

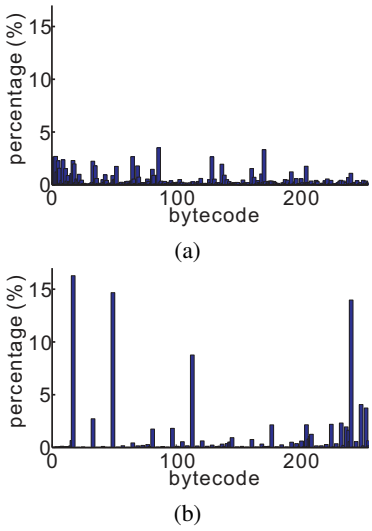


Fig. 3. The distribution of occupancy codes (a) before and (b) after bit reordering.

IV. 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 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.

Normal Quantization: Before compression, normals need to be quantized. This is achieved by iterative subdivision of the normal space (*i.e.* the unit Gauss sphere) to a pre-determined resolution as done by Taubin *et al.* [32] and Botsch *et al.* [26]. Each representative normal can then be identified by an index into a table of quantized unit normals determined by the above subdivision. We use the same subdivision and indexing approaches of Botsch *et al.* [26]. The iterative process of normal space subdivision and indexing is illustrated in Fig. 4.

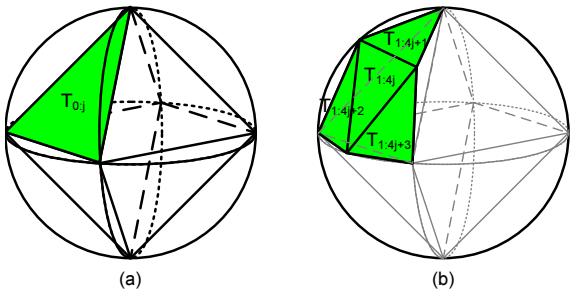


Fig. 4. Normal quantization: (a) an octahedron is inscribed into the unit sphere and its eight facets, $T_{0:j}$ $(j = 0, 1, \ldots, 7)$ form the first level of subdivision; and (b) triangle $T_{0:j}$ $(j \in \{0, 1, \ldots, 7\})$ is subdivided into four sub-triangles, $T_{1:4j} \ldots T_{1:4j+3}$ with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

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, 6 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.

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 i-th level of normal space subdivision, we re-index 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 encoder or 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 arithmetic encoded to indicate whether this local search is successful. If it is, the local index of the matching normal is arithmetic coded; otherwise, a global search is conducted and the global normal index is arithmetic coded.

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. 6(a) and 6(b), we see a much more concentrated distribution around a small number of local normal indices in Fig. 6(b).

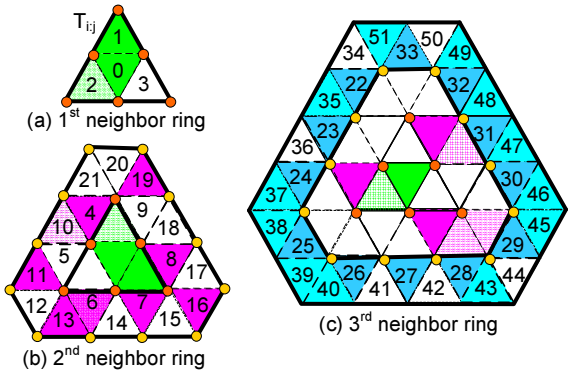


Fig. 5. Local normal indexing: (a) $T_{i:4j} \ldots T_{i:4j+3}$ are assigned the smallest four indices since they have the smallest difference in normal from $T_{i:4j}$. $T_{i:4j+1} \ldots T_{i:4j+3}$ form the 1^{st} neighbor ring of $T_{i:4j}$; (b) the 1^{st} neighbor ring of $T_{i:4j}$ is expanded and the 2^{nd} neighbor ring (in purple and pink) is formed by the triangular facets around the 1^{st} 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 facets; (c) the local neighborhood is expanded to the 3^{rd} neighbor ring similarly.

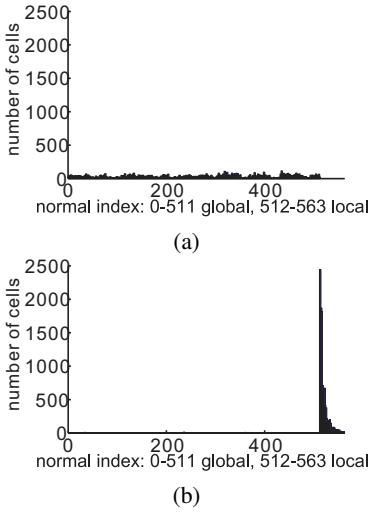


Fig. 6. The distribution of normal indices: (a) global indices at 9-bit quantization for the Igea model and (b) the corresponding local normal indexing, where the local normal indices are offset by 512 for the purpose of plotting.

V. 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 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, the 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.

Adaptive Color Frame Determination: A high degree of color correlation exists in a wide range of 3D models and 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. 7(a)–(c). 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 eigen vectors $\mathbf{V_1}$, $\mathbf{V_2}$ 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 that which 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% that of B. In general, such a compact bounding box leads to 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'.

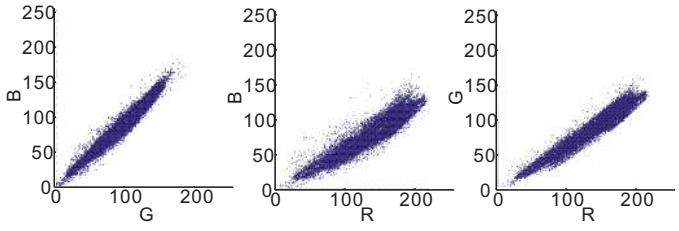


Fig. 7. The distribution of color samples for the Face model viewed from (a) the R-axis (b) the G-axis, and (c) the B-axis.

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 the generalized Lloyd algorithm (GLA), which is a clustering algorithm widely used in the context of vector quantization [34] and pattern recognition [35]. (See 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 to its representative is within the tolerance. In our experiment, we use $1/32$ of 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.
- The following data along each of the above three axes:
  - the value of the first quantization representative.
  - 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 1-D ranges of size k each is about three orders of magnitude faster than running it once for 3-D cubes of size k^3 that partition the entire 3D bounding box. Furthermore, the separable GLA scheme demands a table consisting of $3k$ 1-D representatives, rather than a table consisting of k^3 3-D representatives as demanded by the joint GLA scheme.

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 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, and estimate the distortion per input sample by the average distance between each input color sample and its quantized representative. We see from 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 the octree-based quantization scheme. For $O(k^3)$ 3D color quantization representatives, the octree-based color quantization scheme requires $O(k^3)$ of space to store the color quantization table, while the adaptive scheme requires just $O(k)$ of space since the quantization representatives are stored independently on each of the three axes of the bounding box.

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 arithmetic 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

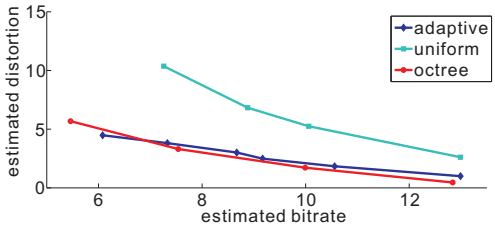


Fig. 8. Comparison of R-D curves of three color quantization schemes for the Face model.

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 1-D luminance-subspace and the 2-D chrominance-subspace to prioritize the luminance component over the chrominance components.

VI. Evaluation on Time and Space Efficiency

A. Asymptotic Performance Analysis

In this subsection, 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 level of detail; namely, $O(N)$ cell subdivisions in total.

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 candidate neighbor set. The probability assignment entails $b + 8$ point-plane distance calculations. The bit re-ordering process demands at most $8 \times \log_2 8 = 24$ comparisons. Since b is a bounded constant by the definition of 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 the 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 as analyzed above, we need to construct the octree and the color quantization table once at the encoder as a pre-processing 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 encoding scheme. For more detailed timing data, we refer to Table IV in Sec. VIII

Memory Cost: Both the encoder and the decoder have to store several tree-front layers of the octree, i.e. 3 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 the 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 $O(N)$.

B. Comparison with the Prior Art

In this subsection, we compare the asymptotic performance of our scheme with the prior art [25]–[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-square local plane approximation in the position coding and the local coordinate transformaion/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 re-ordering 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 prioritization of nonempty-child-cell configurations and re-ordering 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].

VII. 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 the normal and color coding, we have the flexibility in specifying the octree level at which the points would inherit those attributes from 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.

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 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 bitrate while the vertical axis is the corresponding PSNR values for the position/normal coding. We see from Fig. 9 that the 'EveryOther' strategy leads to significantly better position quality at any fixed total bitrate. Interestingly, the 'EveryOther' strategy achieves not only higher position quality, but also higher normal quality at relatively high bitrates. This could be explained by the fact that the normal quantization reaches its maximum resolution quickly with the 'Every' strategy while the position resolution is still relatively low. After that, the normal coding is not efficient since normals are already encoded in full resolution while the relatively low position resolution does not help much in providing good normal prediction accuracy.

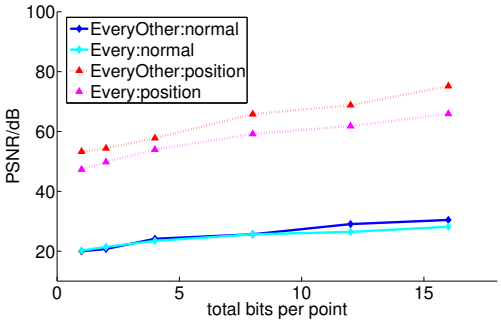
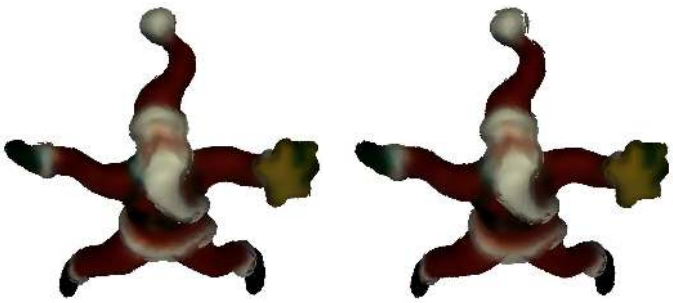


Fig. 9. R-D curves of normal coding for the Dragon model with different bit allocation strategies.

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 2.5bpp with two bit allocation strategies is shown in Fig. 10. Fig. 10(a) adopts color coding at every level and normal resolution refinement and coding at every other level. Fig. 10(b) uses color coding at every other level and normal resolution refinement and coding at every level. We see clearly that the reconstructed model in Fig. 10(a) has higher visual quality than that in Fig. 10(b) especially in regions around Santa's hat, face, beard and waist. This could be due to the higher contribution to visual quality of position's and color's accuracy than 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 higher resolution for position and color data than 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].



(a) (b)

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

VIII. Experimental Results

Nine point-based models are used in our experiments as shown in Fig. 11. They are: Face, Igea, Dragon and Octopus by the 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 experiments for fair comparison with previous work, although only one dragon model is rendered in Fig. 11. Except for models provided by Pointshop 3D, all other models were transferred to the Surfel format by ourselves.

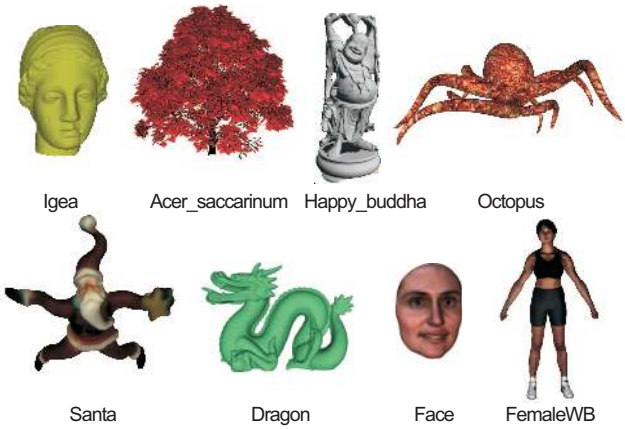


Fig. 11. Models used in our experiments.

A. Rate-Distortion Performance Comparison

The coding performance is measured in bits per point (bpp), 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 the peak signal of 255 for an 8-bit pre-quantization 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]–[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% bitrate 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 Fig. 13(a) and Fig. 13(b), respectively. The horizontal axis is the coding bitrates while the vertical axis gives the position/normalPSNR values. Note that the R-D data of the position coder of Waschbüsch *et al.* [25] are taken from the progressive encoding curves in Fig. 10 of [25] and the R-D data of normal encoding for [25] are taken from the encoding results in Table 2 of [25]. Due to the lack of data, we are not able to make full-range comparison especially for normal encoding as shown in Fig. 13(b). As shown in Fig. 13(a), the PSNR improvement in position coding is around 10dB and 15dB for Igea and Dragon, respectively, at all bitrates. Further, the proposed normal coder can reduce the bitrate by about 50% at certain high PSNR values.

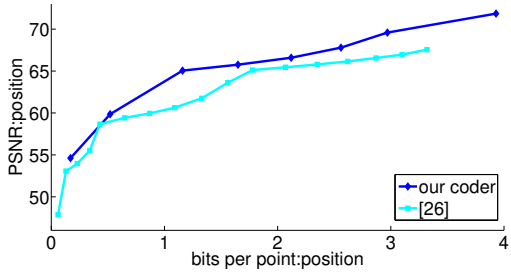
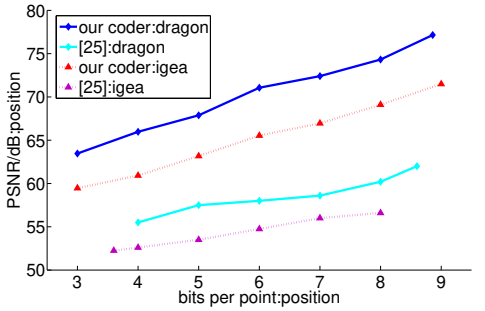
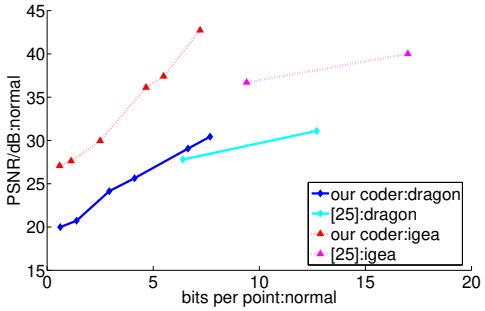


Fig. 12. R-D performance comparison of the proposed coder and that in Botsch *et al.*'s work [26] for Octopus (466k).

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. 14(a)-(c), where the horizontal axes are the total coding bitrates (*i.e.,* the sum of position, normal and color encoding bitrates) in bpp 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 bitrates with an improvement of up to 9dB, which roughly corresponds to 65% 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. 14(b). For lower bit rates, performance of the proposed normal coder is dictated by the coarse normal



(a)



(b)

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

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 bitrates.

Since Schnabel and Klein [27] do not compress normal data, we compare this work with only the proposed position and color coders in Tables I and II. Table II 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's [27] achieves higher PSNR values by 4dB at most bitrates for Dragon-vrip and Igea. In contrast, the proposed color coder outperforms [27] by 10dB or higher at lower bitrates and around 2–6dB at higher bitrates for Santa and FemaleWB. The R-D data of [27] shown in Table I and Table II were not reported in the original paper yet kindly provided by the authors for this comparison.

TABLE I

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) Dragon-vrip

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) Igea

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 III, where the bitrate and distortion are reported in bpp and PSNR, respectively. 'N/A' signifies unavailable data because the fully expanded octree in our experiments has 12 levels only, and the final total bitrate of Happy Buddha is less than 16.0 bpp. Due to high randomness in position and normal data and non-manifoldness, Acer_saccarinum requires more coding bits than other models. Similarly, due to high variation in the color of Octopus, it requires more bits than other models to represent the color information. To the best 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.

TABLE II

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) Santa

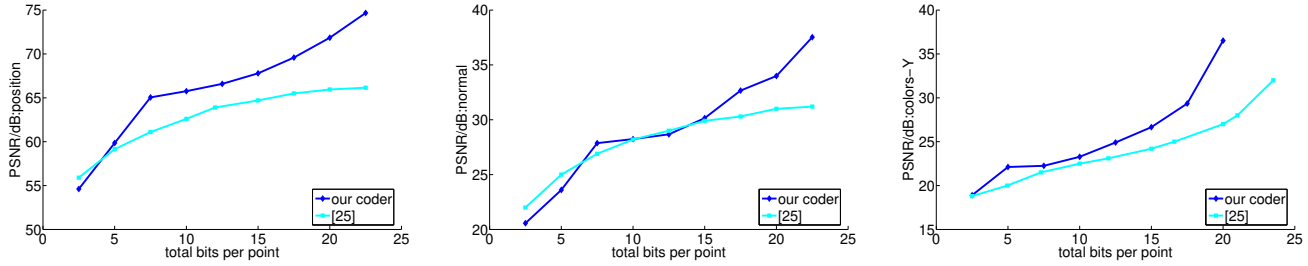
Bitrate(bpp)		1.022	1.31	2.2	5.36	6.62
PSNR	our coder	26.41	26.91	28.83	38.32	43.72
	[27]	13.96	24.95	28.55	32.63	N/A

(b) FemaleFB

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 bitrates of 2bpp, 4bpp, 8bpp and 16bpp, respectively, while the last row shows the uncompressed original models. As shown in this figure, a reasonable profile already appears at 2bpp. We can achieve very decent model quality at 8bpp. The reconstructed models are almost indistinguishable from the original ones at 16bpp.

B. 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 Sec. VI, we report the measured timing data for selected models in Table IV. The experiment was conducted on a PC with Intel® Pentium®-4 3.20GHz CPU and 1GB RAM. The reported timing data in this table refer to full-resolution encoding/decoding with our unoptimized prototype research code. We see from Table IV that models with less than a half million points (*e.g.* Igea, Dragon and Octopus) can be decoded within 10 seconds while models with around one million points (*e.g.* Acer_saccarinum and Happy Buddha) may take about 30–40 seconds. For very large models such as Acer_saccarinum and Happy Buddha, the increase in decoding time seems to be super-linear 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 original point attribute data and the quantization of normal and color data.



(a) (b) (c)

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

TABLE III

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

The reader may have noticed that, for some models other than Acer_saccarinum and Happy Buddha, the relative magnitudes of decoding and/or the encoding time may not be commensurate with the relative magnitudes of the point numbers. Although the number of points in Santa is only about one half 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 significant amount of computation is demanded by the adaptive color quantization in encoding and the color coordinate transformation in decoding. Although there are 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 every other (every) octree level while color coding is performed every (every other) octree level. Reconstruction of a normal vector requires decoding of one integral index and looking up a normal table once, while reconstruction of a color vector requires decoding of 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 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 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 while 1,022,947 cell subdivisions are encoded for Dragon.

IX. 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 re-sample the input model. Thus, it can be potentially used for lossless encoding, if the levels of

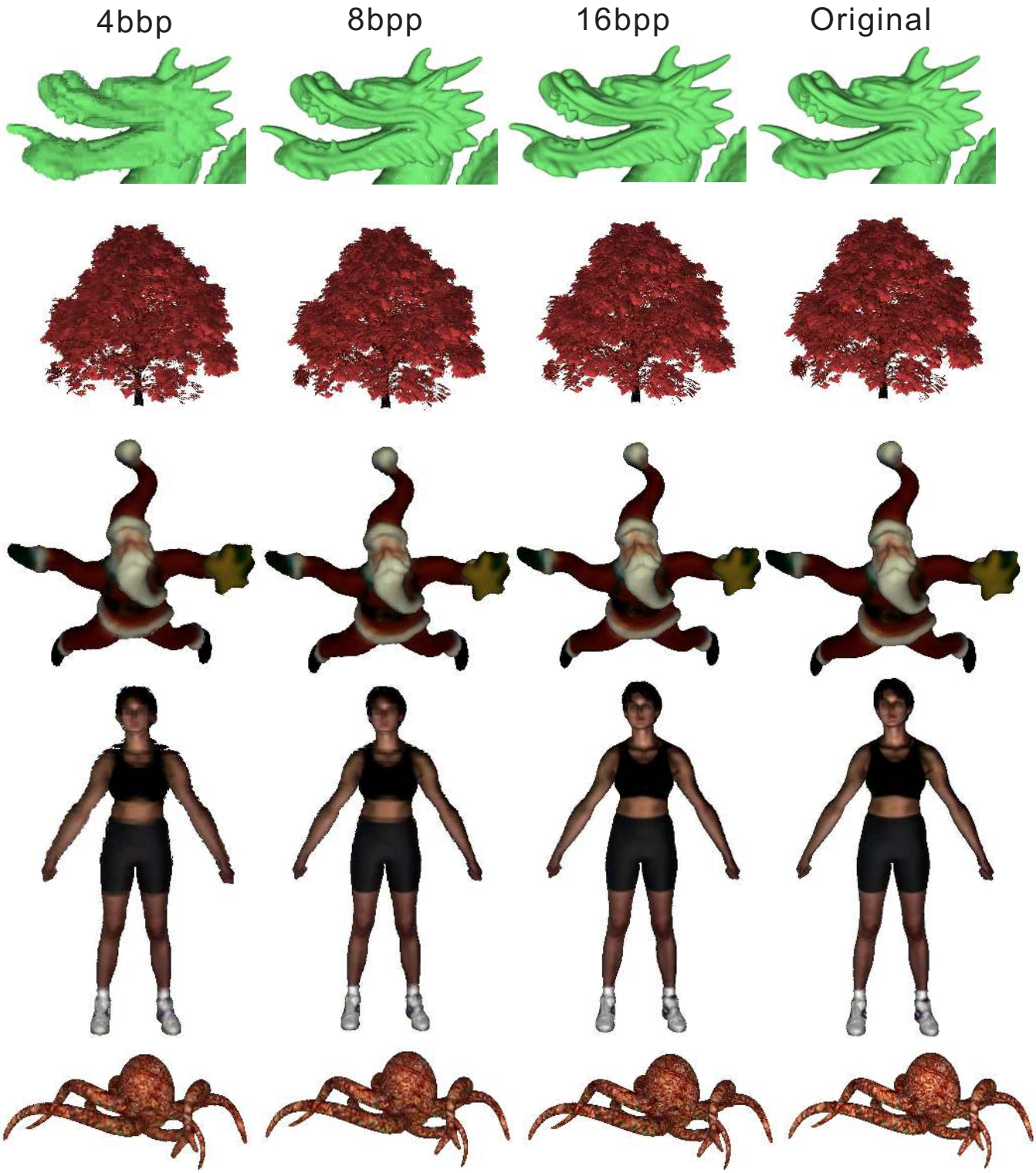


Fig. 15. Models reconstructed at different bitrates.

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

TABLE IV

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

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.

ACKNOWLEDGMENT

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

This research was partially supported by the NSF grants IIS-0712253, CCF- 0738401, and CCF- 0811809.

REFERENCES

[1] J. Peng, C.-S. Kim, and C.-C. J. Kuo, "Technologies for 3D mesh compression: A survey," *Journal of Visual Communication 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," in *Proceedings of Graphics Interface*, 1998, pp. 26–34.

[5] P. Alliez and M. Desbrun, "Valence-driven connectivity encoding for 3D meshes," in *EUROGRAPHICS*, 2001, pp. 480–489.

[6] S. Gumhold and W. Straßer, "Real time compression of triangle mesh connectivity," in *ACM SIGGRAPH*, 1998, pp. 133–140.

[7] J. Rossignac, "Edgebreaker: Connectivity compression for triangle meshes," *IEEE Trans. Visualization and Computer Graphics*, vol. 5, no. 1, pp. 47–61, 1999.

[8] H. Hoppe, "Progressive meshes," in *ACM SIGGRAPH*, 1996, pp. 99–108.

[9] D. Cohen-Or, D. Levin, and O. Remez, "Progressive compression of arbitrary triangular meshes," in *IEEE Visualization*, 1999, pp. 67–72.

[10] P. Alliez and M. Desbrun, "Progressive encoding for lossless transmission of triangle meshes," in *ACM SIGGRAPH*, 2001, pp. 198–205.

[11] J. Li and C.-C. J. Kuo, "Progressive coding of 3-D graphic models," *Proceeding of the IEEE*, vol. 86, no. 6, pp. 1052–1063, Jun 1998.

[12] C. Bajaj, V. Pascucci, and G. Zhuang, "Progressive compression and transmission of arbitrary triangular meshes," in *IEEE Visualization*, 1999, pp. 307–316.

[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," in *Proc. of the 2004 IEEE International Conference on Multimedia and Expo, ICME 2004*, 2004, pp. 1–4.

[15] J. Peng and C.-C. J. Kuo, "Geometry-guided progressive lossless 3D mesh coding with octree (OT) decomposition," in *ACM SIGGRAPH*, 2005, pp. 609–616.

[16] Z. Karni and C. Gotsman, "Spectral compression of mesh geometry," in *ACM SIGGRAPH*, 2000, pp. 279–286.

[17] A. Khodakovsky, P. Schröder, and W. Sweldens, "Progressive geometry compression," in *ACM SIGGRAPH*, 2000, pp. 271–278.

[18] A. Khodakovsky and I. Guskov, "Compression of normal meshes," *Geometric Modeling for Scientific Visualization*, G. Brunnett, B. Hamann, H. Müller, L. Linsen (Eds.). Springer Verlag pp. 189–206, 2002.

[19] S. Gumhold, Z. Karni, M. Isenburg, and H.-P. Seidel, "Predictive point-cloud compression," in *Siggraph Sketches*, 2005.

[20] S. Rusinkiewicz and M. Levoy, "Qsplat: A multiresolution point rendering system for large meshes," in *ACM SIGGRAPH*, 2000, pp. 343–352.

[21] S. Fleishman, D. Cohen-Or, M. Alexa, and C. T. Silva, "Progressive point set surfaces," *ACM Trans. Graph.*, 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," in *Eurographics Symposium on Point-Based Graphics*, 2004, pp. 103–112.

[23] J. Wu, Z. Zhang, and L. Kobbelt, "Progressive splatting," in *Eurographics Symposium on Point-Based Graphics*, 2005, pp. 25–32.

[24] A. Kalaiah and A. Varshney, "Statistical geometry representation for efficient transmission and rendering," *ACM Transactions on 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," in *Eurographics Symposium on Point-Based Graphics*, 2004.

[26] M. Botsch, A. Wiratanaya, and L. Kobbelt, "Efficient high quality rendering of point sampled geometry," in *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2002, pp. 53–64.

[27] R. Schnabel and R. Klein, "Octree-based point cloud compression," in *Eurographics Symposium on Point-Based Graphics*, 2006, pp. 111–120.

[28] Y. Huang, J. Peng, C.-C. J. Kuo, and M. Gopi, "Octree-based progressive geometry coding of point clouds," in *Eurographics Symposium on Point-Based Graphics*, 2006, pp. 103–110.

[29] J. Krüger, J. Schneider, and R. Westermann, "Duodecim - a structure for point scan compression and rendering," in *Eurographics Symposium on Point-Based Graphics*, 2005, pp. 99–107.

[30] M. Gopi, S. Krishnan, and C. Silva, "Surface reconstruction using lower dimensional localized delaunay triangulation," *Eurographics*, 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," *Proceeding of the IEEE*, vol. 96, no. 6, pp. 1228–1243, Jun 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. Communications*, 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 Ltd, 1986.

[37] M. Pauly, M. Gross, M., and L. KOBBELT, "Efficient simplification of point-sampled surfaces," in *Proc. VIS*, 2002, pp. 163–170.

APPENDIX

The generalized Lloyd algorithm (GLA): 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 1-D 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.
   a) 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.
   b) 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.

Yan Huang Yan Huang is a Ph. D. student in the Department of Computer Science at the University of California, Irvine. She got her M.S. at the University of California, Irvine in 2003 and her B.S. in Computer Science at the Beijing University, People's Republic of China in 1997. Her current research interests include 3D data processing, interactive walkthrough applications and realtime rendering.

Jingliang Peng Dr. Jingliang Peng received the Ph.D. degree in electrical engineering from University of Southern California in 2006, the B.S. and the M.S. 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 in 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 Dr. C.-C. Jay Kuo received the B.S. degree from the National Taiwan University, Taipei, in 1980 and the M.S. and Ph.D. degrees from the Massachusetts Institute of Technology, Cambridge, in 1985 and 1987, respectively, all in Electrical Engineering. He is Director of the Signal and Image Processing Institute (SIPI) and Professor of Electrical Engineering, Computer Science and Mathematics at the 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 co-author of about 140 journal papers, 730 conference papers and 9 books.

Dr. Kuo is a Fellow of IEEE and SPIE and a member of ACM. He is Editor-in-Chief for the Journal of Visual Communication and Image Representation, and Editor for the Journal of Information Science and Engineering, LNCS Transactions on Data Hiding and Multimedia Security and the EURASIP Journal of Applied Signal Processing. He was on the Editorial Board of the IEEE Signal Processing Magazine in 2003-2004. He served as Associate Editor for IEEE Transactions on Image Processing in 1995-98, IEEE Transactions on Circuits and Systems for Video Technology in 1995-1997 and IEEE Transactions on Speech and Audio Processing in 2001-2003. Dr. Kuo received the National Science Foundation Young Investigator Award (NYI) 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 Communication Society in 2005. He is an IEEE Signal Processing Society Distinguished Lecturer in 2006. He is also Advisor to the SMPTE (Society of Motion Picture Television Engineers)-USC student chapter.

M. Gopi Dr. Gopi Meenakshisundaram (M. Gopi) is an Assistant Professor in the Department of Computer Science at the University of California, Irvine. He got his Ph.D from the University of North Carolina at Chapel Hill in 2001, M.S. from the Indian Institute of Science, Bangalore in 1995, and B.E from Thiagarajar College of Engineering, Madurai, India in 1992. He has worked on various geometric and topological problems in computer graphics. His current research interest focusses on graph algorithms for geometry processing, geometry and topology compression, and sketch based modelling.