Geometry-guided Progressive Lossless 3D Mesh Coding with Octree (OT) Decomposition

Jingliang Peng* C.-C. Jay Kuo†
University of Southern California

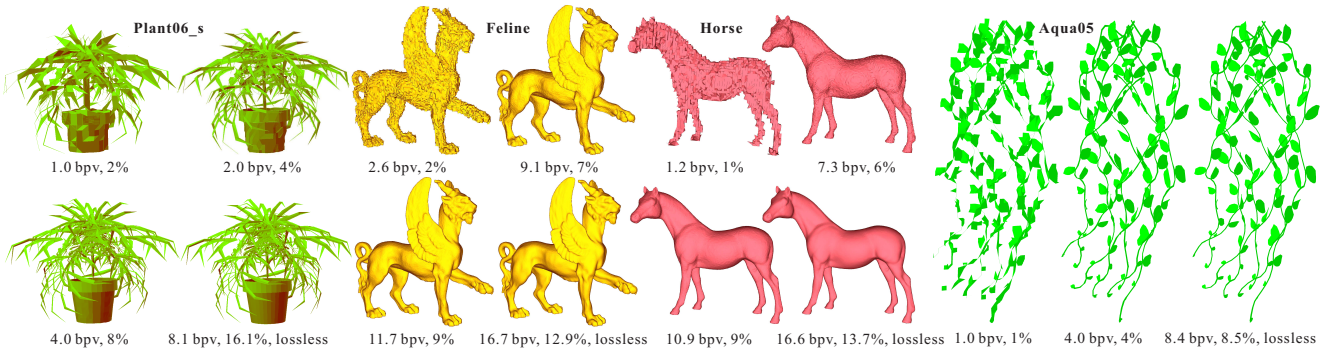


Figure 1: Examples of progressive mesh reconstruction.

Abstract

A new progressive lossless 3D triangular mesh encoder is proposed in this work, which can encode any 3D triangular mesh with an arbitrary topological structure. Given a mesh, the quantized 3D vertices are first partitioned into an octree (OT) structure, which is then traversed from the root and gradually to the leaves. During the traversal, each 3D cell in the tree front is subdivided into eight child-cells. For each cell subdivision, both local geometry and connectivity changes are encoded, where the connectivity coding is guided by the geometry coding. Furthermore, prioritized cell subdivision is performed in the tree front to provide better rate-distortion (R-D) performance. Experiments show that the proposed mesh coder outperforms the kd-tree algorithm in both geometry and connectivity coding efficiency. For the geometry coding part, the range of improvement is typically around 10%∼20%, but may go up to 50%∼60% for meshes with highly regular geometry data and/or tight clustering of vertices.

Keywords: 3D geometry compression, mesh compression, progressive lossless coding, non-manifold mesh, triangle soup

1 Introduction

3D graphics data are widely used in multimedia applications such as video gaming, engineering design, virtual reality, e-commerce and scientific visualization. With increasing popularity and complexity of 3D graphics data but limited network bandwidth and processing power, it is critical to compress 3D mesh data efficiently.

A typical mesh codec encodes three types of information: connectivity, geometry and attributes. The connectivity data describe the adjacency relationship between vertices; the geometry data provide the positions of vertices; and the attribute data give surface normals, material reflectance, texture coordinates, etc. Most of current mesh encoders only deal with the connectivity data and the geometry data under the argument that the attribute data can be encoded similarly to the geometry data. Most earlier research in 3D mesh coding is connectivity-centric in that the compact representation of connectivity data is given a higher priority, while the geometry coding is driven, and also restrained at the same time, by the connectivity coding. However, since the geometry data are dominant in the compressed file size in most cases, a good geometry coder is essential to the high coding efficiency of a 3D graphic codec. Thus, geometry-centric algorithms focusing on the geometry coding, which even guided the connectivity coding, have emerged in recent years.

1.1 Historical Review

Early research on 3D mesh compression focused on single-rate compression techniques to save the bandwidth between the CPU and the graphics card. Codecs of this category include [Taubin and Rossignac 1998; Bajaj et al. 1999b; Touma and Gotsman 1998; Alliez and Desbrun 2001b; Gumhold and Straßer 1998; Rossignac 1999; Coors and Rossignac 2004], all of which are lossless codecs, only allowing for negligible quantization error. Among those, the best achievable geometry coding bit rate is 6∼10 bpv at a quantization resolution of 8 bits per coordinate, and the best achievable connectivity coding bit rate is typically less than 3 bpv, using the state-of-the-art codecs [Touma and Gotsman 1998; Alliez and Desbrun 2001b; Coors and Rossignac 2004]. Recently, lossy single-rate mesh codecs were proposed in [Szymczak et al. 2002; Attene et al. 2003], which achieves much higher coding efficiency by combining compression with remeshing.

Later on, with the increasing popularity of networked applications, progressive compression and transmission has been intensively studied, which enables the progressive reconstruction of a 3D mesh in different levels of detail (LODs). Algorithms of this category include progressive meshes [Hoppe 1996] and its extension in [Popovic and Hoppe 1997; Taubin et al. 1998; Pajarola and Rossignac 2000], the patch coloring approach [Cohen-Or et al. 1999], the valence-driven conquest approach [Alliez and Desbrun 2001a], the embedded coding approach [Li and Kuo 1998], and the layered decomposition approach [Bajaj et al. 1999a], all of which

*e-mail: jingliap@usc.edu
†e-mail:cckuo@sipi.usc.edu

are connectivity-centric. Based on the observation that geometry data are often dominant in the compressed file size, geometry-centric algorithms have emerged in recent years, including the kd-tree mesh codec [Gandoin and Devillers 2002; Devillers and Gandoin 2000], the spectral coding geometry codec [Karni and Gotsman 2000], and the wavelet-based mesh codecs [Khodakovsky et al. 2000; Khodakovsky and Guskov 2000]. Among all the above-mentioned progressive mesh codecs, the spectral coding geometry codec and the wavelet-based mesh codecs are lossy codecs that either truncate high-frequency geometric information or start with a complete remeshing of the input manifold model and have very high coding efficiency. However, it is not straightforward about how to remesh/transform complex non-manifold meshes, and some applications require the original data to be faithfully preserved. That context calls for the progressive lossless mesh codecs that faithfully preserve both connectivity and geometry data in the full resolution, among which the kd-tree codec produced the best results and is most related to our work.

For a more complete survey of techniques in 3D mesh compression, readers are referred to [Peng et al. ; Alliez and Gotsman 2003; Gotsman et al. 2002].

1.2 kd-Tree Mesh Coder

The kd-tree coding algorithm employs an iterative kd-tree decomposition based on 3D cell subdivisions, inspired by [Schmalstieg and Schaufler 1997; Rossignac and Borrel 1993] which spatially group vertices into clusters to construct different LODs. When it subdivides a cell into two child-cells, the number of vertices in one of the two child-cells is arithmetic coded, and the associated connectivity change is encoded using one of two operations: the vertex split [Hoppe 1996] or the generalized vertex split [Popovic and Hoppe 1997]. As reported, it outperforms prior work in terms of coding efficiency. Furthermore, it can compress triangular meshes of any topology, even triangle soups. In spite of the excellent performance of the kd-tree algorithm, some drawbacks are observed and summarized below.

- Redundancy in the vertex number information. For the purpose of progressive mesh reconstruction, not the exact number of vertices in each cell, but whether each cell is empty or not is necessary since each cell is represented with its centroid in any LOD. The coding overhead is higher at higher levels (*i.e.*, levels closer to the root), since higher level cells generally have more vertices.
- Geometry-connectivity correlation not fully utilized. Local geometry data are exploited in the connectivity coding to predict the updated connectivity. However, local connectivity data are not utilized in the geometry coding which, instead, predicts from a weakly defined "neighborhood" based on spatial closeness of cells. The search for local "neighborhood" costs computation and memory, and the spatial-closeness-based "neighborhood" may provide inaccurate information.
- No preferential treatment of cell subdivisions. Cells in the tree front of the same level are subdivided one by one without any discrimination. However, the coding of these cells may have a different R-D contribution.

1.3 Overview of Proposed Algorithm

In this work, we propose a new progressive lossless mesh encoder using the octree (OT) decomposition. By using the OT decomposition, we subdivide a 3D cell into eight at each step. The OT decomposition is used in [Saupe and Kuska 2001; Laney et al. 2002; Lee et al. 2003] to compress isosurfaces, and in [Botsch et al. 2002] to represent point sampled geometry data. The motivation of choosing the OT decomposition is that an OT cell subdivision leads to richer information that will assist our geometry and connectivity encoders a lot. Based on the OT decomposition, our proposed algorithm has the following distinguished features.

- The geometry coding algorithm does not encode the vertex number in each cell, but encode the information whether each cell is empty or not, which is often more concise.
- A uniform connectivity coding approach is adopted, which is efficient and can be potentially applied to the coding of arbitrary polygonal meshes.
- Either geometry data or connectivity data are exploited for local prediction of the other.
- Prioritized cell subdivision is performed in the tree front to achieve better R-D performance.

It is worthwhile to point out that Botsch *et al.* [2002] also encoded the information whether each child-cell is empty or not after each octree cell subdivision. But coding efficiency was not optimized and there was no need to encode connectivity data in their work.

Given a 3D mesh, we first calculate its bounding box, quantize the vertex coordinates, and build up an OT structure through recursive 3D space partitioning with each node in the OT representing a 3D cell. After that, the proposed mesh encoder traverses the OT from the root to the lowest level in a top-down fashion. During the traversal, each 3D cell in the tree front is subdivided into eight child-cells with three cell bi-partitionings in three directions along the X, Y and Z axes, respectively. Nonempty child-cells are recursively subdivided until the finest resolution allowed by the quantization scheme is reached. On the other hand, empty child-cells will not be subdivided any more. Whenever a cell is subdivided into eight child-cells, we need to encode the associated local change in both geometry and connectivity.

Generally speaking, for the local geometry change, we have to specify which child-cells are nonempty. As to the local connectivity change, we should encode the connectivity between nonempty child cells, and the connectivity between nonempty child-cells and the parent-cell's neighbor cells. The detail of the geometry coding and the connectivity coding algorithms will be discussed in Secs. 2 and 3, respectively.

The proposed mesh coder significantly outperforms the kd-tree algorithm in both geometry and connectivity coding efficiency. For the geometry coding part, the range of improvement is highly dependent on the characteristics of the mesh to encode. Typically, it is around 10%$\sim$20%, but may go up to 50%$\sim$ 60% for meshes with highly regular geometry data and/or tight clustering of vertices.

2 Geometry Coder

2.1 Nonempty-Child-Cell Coding

In contrast with the kd-tree geometry encoder, we do not encode the exact vertex numbers. Instead, we only encode the information of nonempty child-cells after each cell subdivision. To achieve this purpose, we propose the nonempty-child-cell coder to encode the indices of nonempty child-cells.

Here, two cells are called neighbors if there is at least one edge in the original mesh connecting the vertex of one cell to that of the other. In the corresponding LOD of the mesh, the neighbor relationship of two cells is represented by an edge between their representative vertices. The valence of a cell is defined as the number of its neighbor cells. Each child-cell is labelled with a 3-bit index, $b_1b_2b_3$, according to its location relative to each bi-partitioning. For example, b_1 is the bit index with respect to the X-axis bi-partitioning. It is assigned 0 (1) if its x-value is in the lower (upper) half of the parent-cell. Consider the case that there are T nonempty child-cells after a cell subdivision. The indices of nonempty child-cells form a tuple, $(t_1, t_2, \ldots, t_T)$, where $t_i \in \{0, 1, \ldots, 7\}$, $1 \le i \le T$, and T is the tuple dimension. Typically, T values are concentrated

in $4 \sim 8$ in higher tree levels (closer to the root) and $1 \sim 3$ in lower tree levels (closer to the leaves). T values are more widely spread between 1 and 8 in the middle tree levels. We call a tuple with dimension T a T-tuple.

To encode a cell subdivision, we first encode the number of $1 \leq T \leq 8$. Note that T cannot be 0 since we only subdivide nonempty cells. In general, the bigger the cell valence is, the more nonempty child-cells that cell will have. Furthermore, cells of the same OT level tend to have a similar number of nonempty child-cells. Based on these observations, T is arithmetic coded using both the cell's OT level and its valence as the context, leading to 30%$\sim$50% improvement compared with coding without context.

The nonempty-child-cell tuple of the target cell is also arithmetic coded, using the T value of the target cell as the context. For a given T, the number of possible tuples can be computed as a combinatorial number, $K_T = C_T^8$. By encoding the nonempty-child-cell tuple in a straightforward manner using the arithmetic coder under context T, we need $\log_2 K_T$ bits per nonempty-child-cell tuple on the average. One simple way to implement this coder is to have a look-up table that links the codes and the tuples. To further improve coding efficiency, we can estimate the pseudo-probability of each T-tuple's being the nonempty-child-cell tuple, sort all the possible T-tuples in a descending order of their pseudo-probability values, and encode the nonempty-child-cell tuple's index in the sorted array with an arithmetic coder under context T.

Before calculating the tuples' pseudo-probability values, we first calculate a priority value for each child-cell, which estimates its possibility of being nonempty. The prediction is based on the observation that nonempty child-cells tend to be close to the centroid of the parent-cell's neighbor cells, if the original 3D surface is locally sampled with a high regularity. Thus, we can calculate a priority value for each child-cell by taking into account the number of the parent's neighbor cells in its vicinity and their distances to the centroid of the parent cell.

Associated with the parent-cell, there are three cell bi-partitionings along three orthogonal axes, denoted by $bp_i (i \in \{1,2,3\})$, where the subscript i means the axis number, and the 1st, the 2nd and the 3rd axis refers to the X, Y, and Z axis, respectively. Associated with each cell bi-partitioning, the partitioning plane also partitions the neighbor cells into two subsets: $S_{i,1} = \{C_{i,1,1}, C_{i,1,2}, \ldots, C_{i,1,n_{i,1}}\}$ and $S_{i,2} = \{C_{i,2,1}, C_{i,2,2}, \ldots, C_{i,2,n_{i,2}}\}$. They contain $n_{i,1}$ and $n_{i,2}$ neighbor cells, respectively. For each neighbor cell $C_{i,j,k} (i \in \{1,2,3\}, j \in \{1,2\}, k \in \{1,2,\ldots,n_{i,j}\})$, we can calculate the distance along the ith axis, $d_{i,j,k}$, between the centroid of the cell to be subdivided and the centroid of the neighbor cell $C_{i,j,k}$, resulting in two distance subsets, $L_{i,1} = \{d_{i,1,1}, d_{i,1,2}, \ldots, d_{i,1,n_{i,1}}\}$ and $L_{i,2} = \{d_{i,2,1}, d_{i,2,2}, \ldots, d_{i,2,n_{i,2}}\}$, corresponding to $S_{i,1}$ and $S_{i,2}$. Next, we sum up the distances in $L_{i,j} (i \in \{1,2,3\}, j \in \{1,2\})$ and get

$$D_{i,j} = \sum_{k=1}^{n_{i,j}} w_{i,j,k} \times d_{i,j,k},$$

where $w_{i,j,k}$ is the weight assigned to neighbor cell $C_{i,j,k}$. Since cells at lower OT levels provide more accurate geometry information, we assign the OT level number of cell $C_{i,j,k}$ to $w_{i,j,k}$. Thus, it is called the weight of levels. After bi-partitioning bp_i, $i \in \{1,2,3\}$, if child-cell $c_k (k \in \{1,2,\ldots,8\})$ and the cells in $S_{i,k_i} (k_i \in \{1,2\})$ are located at the same side of the bi-partitioning plane, its priority p_k is calculated as

$$p_k = \sum_{i=1}^{3} \left(w_i \times D_{i,k_i} \right),$$

where w_i is the weight of unbalance associated with the bi-partitioning bp_i. For each $bp_i (i \in \{1,2,3\})$, we calculate its extent of "unbalancing" as

$$u_i = \left| \frac{D_{i,1}}{D_{i,1} + D_{i,2}} - 0.5 \right|.$$

Sorting u_i, $i = 1,2,3$, in a descending order, we obtain u_{i_j}, $j = 1,2,3$ and $i_j \in \{1,2,3\}$, such that $u_{i_1} \geq u_{i_2} \geq u_{i_3}$. Observing that a more unbalanced bi-partitioning is usually more helpful in nonempty-child-cell prediction, we assign to each bi-partitioning, bp_i, a weight of unbalance as $w_{i_1} = 3$, $w_{i_2} = 2$, $w_{i_3} = 1$.

To illustrate the idea described above, let us consider a 2D example as shown in Fig. 2, where the dotted lines represent part of the quantization grid, the solid squares represent the cells in the current tree front, a solid line between two cells represent the neighbor relationship, the blue dashed lines represent the two bi-partitionings, bp_1 and bp_2, respectively, and a black dot means that the associated cell is nonempty. Cells are of different sizes because they are located in different OT levels. In Fig. 2, C_0 is the cell to be subdivided, $C_1 \sim C_5$ are the neighbor cells of C_0. Cell C_0 will be subdivided into four child-cells, $c_1 \sim c_4$, of which c_2 and c_4 are nonempty. Associated with the bi-partitioning bp_1, we have $S_{1,1} = \{C_2, C_3, C_4\}$, $S_{1,2} = \{C_1, C_5\}$, $L_{1,1} = \{5,5,1\}$, $L_{1,2} = \{8,7\}$. Assuming that C_0 and C_1 are located at the 5th OT level, and $C_2 \sim C_5$ are located at the 6th OT level, we have weights of the level as $w_{1,1,1} = w_{1,1,2} = w_{1,1,3} = w_{1,2,2} = 6$, and $w_{1,2,1} = 5$. Thus, $D_{1,1} = \sum_{k=1}^{3} w_{1,1,k} \times d_{1,1,k} = 66$, $D_{1,2} = \sum_{k=1}^{2} w_{1,2,k} \times d_{1,2,k} = 82$. Similarly, we obtain $D_{2,1} = 50$, and $D_{2,2} = 78$. Since bp_2 is more unbalanced than bp_1, we assign more weight to bp_2. That is, we have weights $w_1 = 1$, $w_2 = 2$. Finally, we calculate the priority value for each child-cell. For example, the priority value for child-cell c_1 is $p_1 = w_1 \times D_{1,1} + w_2 \times D_{2,1} = 166$. The priority values for other child-cells can be similarly obtained. They are $p_2 = 182$, $p_3 = 222$, and $p_4 = 238$.

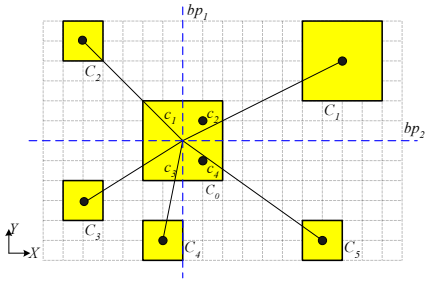
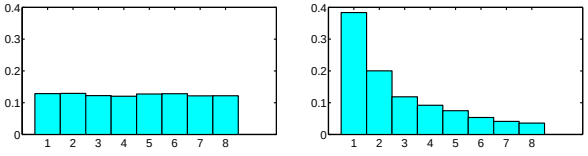


Figure 2: A 2D example of priority value calculation.

To demonstrate the effectiveness of priority calculation, for the manifold meshes used in our experiments, we count the number of nonempty child-cells at each of the eight child-cell locations before and after the priority calculation, and plot the histograms in Fig. 3(a) and Fig. 3(b) respectively. Note that, in Fig. 3(b), the child-cell locations are ordered with respect to the priority values within each cell subdivision. From this figure, we see that after priority calculation, the nonempty child-cells are concentrated in high-priority locations.



(a) before (b) after

Figure 3: The nonempty-child-cell number histograms before and after priority calculation for the tested manifold meshes.

After calculating the child-cells' priority values, we need to calculate the pseudo-probability values of all possible T-tuples for given T. More specifically, for each T-tuple $TP_i = (i_1, i_2, \ldots, i_T)$, its pseudo-probability PP_i is calculated by $PP_i = \sum_{j=1}^{T} p_{i_j}$. According to our experiments on the tested manifold meshes, the above tuple sorting technique improves the nonempty-child-cell-tuple coding by 23% on the average.

2.2 Prioritized Cell Subdivision

In contrast with the original kd-tree algorithm that treats all cells in the tree front equally, we rank cells in the tree front according to their importance and subdivide more important cells earlier to provide better mesh quality at lower bit rates. All cells in the tree front are efficiently maintained with a heap structure, which costs $O(\log(N))$ time, where N is the number of elements in the heap, for the most important cell search/removal or new cell insertion. The key issue lies in the definition of cell importance. Here, we identify the following three rules.

1. A higher cell valence implies more vertices contained in a cell.
2. A bigger cell size implies more mesh quality improvement when the cell is subdivided.
3. A larger distance from neighbor cells implies more impact of cell subdivision on local 3D volume refinement.

Based on the above observation, we define the importance value I for each cell c as

$$I_c = vsl,$$

where v is the cell valance, s the cell size and l the average distance of the target cell's centroid to its neighbor cells' centroids. On one hand, the R-D performance is improved by dividing important cells first. On the other, the coding bit rate is reduced by about 0.2 bpv on the average for the tested manifold meshes, since earlier subdivision of more important cells often leads to better improvement of the mesh, which will in turn benefit the nonempty-child-cell prediction in later subdivision of less important cells.

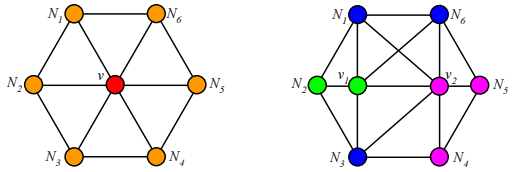
3 Proposed Connectivity Coder

For each cell subdivision, the change of local connectivity has to be encoded by the connectivity encoder. Under the kd-tree framework, Devillers and Gandoin [2000] proposed a purely edge-based connectivity encoder, which can encode arbitrary edge-based connectivity but its coding efficiency is not yet optimized and the facet information are missing. Later on, Gandoin and Devillers [2002] proposed another connectivity encoder that encodes the connectivity change associated with each cell subdivision using one of two operations: the vertex split [Hoppe 1996] or the generalized vertex split [Popovic and Hoppe 1997]. As a result, it can encode the connectivity of any simplicial complex with improved coding efficiency. However, it is not suitable for an arbitrary polygonal mesh since it is generally not a simplicial complex. In this work, we propose an efficient connectivity encoder that can encode the connectivity data of arbitrary triangular meshes and can be easily extended to polygonal mesh coding. In the following, we first concentrate on the coding of triangular mesh connectivity.

For each OT cell subdivision that subdivides a cell C into K nonempty child-cells, we use $K-1$ kd-tree cell subdivisions to simulate it, where each subdivision partitions a set of nonempty child OT cells into two subsets. In each of these kd-tree cell subdivisions, with the positions of all nonempty child OT cells known before the kd-tree simulation, the representative point of each nonempty kd-tree cell is no more its centroid as in the kd-tree algorithm [Gandoin and Devillers 2002], but the average position of the centroids of all nonempty OT cells contained, which generally provides a better approximation and helps increase the accuracy of the prediction technique used in our connectivity coder.

We say that there is a vertex split whenever the corresponding kd-tree cell subdivision leads to two nonempty child-cells, without differentiating between the vertex split and the generalized vertex split as done in [Gandoin and Devillers 2002]. If two cells are neighbors, we say that their representative vertices are neighbors in the current LOD of mesh. For each vertex split, let us denote the vertex to split by v, the neighbor vertices before the vertex split by N_i, $i = 1, 2, \ldots, M$, where M is the number of neighbor vertices, and the two new vertices resulted from the vertex split by v_1 and v_2. Then, we need to encode the following information associated with this vertex split:

- vertices in N_i that are connected to both v_1 and v_2 (called the pivot vertices);
- whether each nonpivot vertex in N_i is connected to v_1 or v_2;
- whether v_1 and v_2 are adjacent in the refined mesh.



(a) (b)

Figure 4: Illustration of the vertex split.

An example of the vertex split is illustrated in Fig. 4. The configurations before and after the vertex split are shown in Figs. 4(a) and (b), respectively. We see that, of the six neighbor vertices, N_1, N_3 and N_6 are pivot vertices that are connected to both v_1 and v_2 in the refined mesh. In the connectivity coding of this example, we need to specify the pivot neighbor vertices, N_1, N_3 and N_6, to assign each of the rest neighbor vertices to either v_1 or v_2 and to specify that v_1 and v_2 are adjacent in the refined mesh.

Note that only the edge information is encoded in above. For the purpose of mesh reconstruction, we also need to keep the facet information, which can be done automatically with no extra coding cost, as described in Subsection 3.4.

3.1 Coding of Pivot-Vertex-Tuple

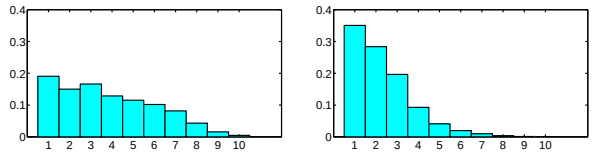
To encode the pivot vertices among all neighbor vertices N_i, $i = 1, 2, \ldots, M$, we employ a method similar to that used in the proposed geometry coder. Assuming that there are P pivot vertices, the number P is arithmetic coded using M as the context. Next, we need to encode the pivot-vertex-tuple, which is the P-tuple of the pivot vertices' indices. For each possible P-tuple of the neighbor vertex indices, we estimate its probability of being the pivot-vertex-tuple. The probability values of all possible P-tuples form a probability table which is utilized by the arithmetic coder that encodes the pivot-vertex-tuple using both M and P as contexts. The remaining issue is how to estimate each P-tuple's probability of being the pivot-vertex-tuple.

First, we estimate the priority p_i for each neighbor vertex N_i, where $1 \le i \le M$. To estimate p_i, we make three virtual edges: between N_i and v_1, between v_1 and v_2, and between v_2 and N_i. Generally speaking, the more regular the triangle $\triangle N_i v_1 v_2$ is, the more probable that N_i will be a pivot vertex. For a given perimeter, the bigger the area, the more regular a triangle will be. Thus, we calculate the regularity r_i of $\triangle N_i v_1 v_2$ and the priority p_i of N_i as

$$p_i = r_i = \frac{\sigma_i}{2s} = \frac{\sqrt{s(s-a)(s-b)(s-c)}}{2s},$$

where a, b and c are the lengths of the three edges in $\triangle N_i v_1 v_2$, respectively, σ_i is the area of $\triangle N_i v_1 v_2$ and $s = (a+b+c)/2$.

To demonstrate the effectiveness of the proposed priority calculation, for the manifold meshes used in our experiments, we count the number of pivots at each neighbor locations before and after the priority calculation, and plot the histograms in Figs. 5(a) and (b), respectively. The neighbor locations are indexed based on their order in the target cell's neighbor vertex list in Fig. 5(a) while they are indexed based on the relative magnitude of priority values within each vertex split in Fig. 5(b). Note that we only plot for the vertex splits with at most 10 neighbor vertices, since they constitute the majority of vertex splits. From Fig. 5, we see that the pivot vertices are concentrated in high-priority locations after priority calculation.



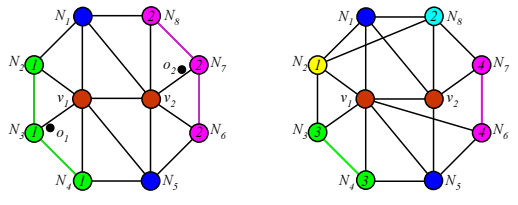
(a) before (b) after

Figure 5: The pivot number histograms before and after priority calculation for the tested manifold meshes.

Once the priority value for each neighbor vertex is calculated, the probability of each P-tuple's being the pivot-vertex-tuple is estimated by summing up the priority values of the neighbor vertices contained in that tuple. After normalization, the estimated probability table is obtained, which is used by the arithmetic coder to encode the pivot-vertex-tuple. According to our experiments on the tested manifold meshes, the above probability estimation technique improves the pivot-vertex-tuple coding by 45% on the average.

3.2 Nonpivot Neighbor Assignment

After identifying pivot vertices, we have to assign the nonpivot neighbor vertices to v_1 or v_2. To do so, we first partition the nonpivot vertices in N_i, $1 \le i \le M$, into different segments. Each nonpivot vertex that is adjacent to more than two other vertices in N_i forms a separate segment. Then, the remaining nonpivot vertices are partitioned into maximum connected segments. The segment partitioning is illustrated in Fig. 6. For the configuration in Fig. 6(a), N_1 and N_5 are pivot vertices, and the nonpivot neighbor vertices are partitioned into two maximum connected segments: $\{N_2, N_3, N_4\}$ and $\{N_6, N_7, N_8\}$. Each nonpivot neighbor vertex is labeled with its segment number and each segment is colored uniquely. For the configuration in Fig. 6(b), again N_1 and N_5 are pivot vertices, and the nonpivot neighbor vertices are partitioned into four segments: $\{N_2\}$, $\{N_8\}$, $\{N_3, N_4\}$, and $\{N_6, N_7\}$. Note that either N_2 or N_8 forms a segment by itself since it is adjacent to three other vertices in N_i, $1 \le i \le 8$.



(a) (b)

Figure 6: Illustration of nonpivot neighbor vertex segmentation.

For each segment, we use one flag bit to indicate whether all the vertices in it are connected to the same one of v_1 and v_2. If not, we treat each vertex in that segment as a separate segment. Sometimes, a nonpivot vertex segment may be connected to both v_1 and v_2. For instance, in the 4th segment in Fig. 6(b), N_6 is connected to v_1, and N_7 is connected to v_2. However, in a manifold or "almost manifold" mesh, almost every nonpivot segment is connected to only one of v_1 and v_2, such as the two nonpivot segments that are shown in Fig. 6(a). Since the flag bits can be efficiently coded with an arithmetic coder, the segmentation of nonpivot vertices provides an effective way to group vertices connected to the same new vertex.

Next, for each segment S_i, we calculate its centroid, o_i, and calculate the distances $d_{i,j}$, $j = 1,2$, between o_i and v_j, $i = 1,2$, respectively. We predict that the segment is adjacent to the one of v_1 and v_2 with the smaller distance. For instance, in Fig. 6(a), the centroids, o_1 and o_2 are calculated for the 1st and the 2nd segments, respectively. For the 1st segment, since o_1 is closer to v_1 than to v_2, we predict that it is adjacent to v_1, which is accurate. For each segment, a 1-bit flag is used to indicate whether the prediction is accurate. This flag bit is again arithmetic coded.

3.3 Adjacency between New Vertices

One bit is used to indicate whether the new vertices, v_1 and v_2, are connected in the refined mesh, which is arithmetic coded, too. According to our experiments, the pivot-vertex-tuple selection dominates the connectivity coding cost, while the coding of the nonpivot neighbor assignment and the adjacency between the new vertices can be done very efficiently, whose total cost is in general less than 0.5 bpv for manifold or "almost manifold" meshes.

3.4 Facet Construction

In this work, we assume that facets in a triangular mesh are double-sided, *i.e.*, a facet has the same set of material properties in its two opposite sides and can be rendered from either side when it comes into view. In other words, for the purpose of rendering, we do not differentiate between the two orientations of a facet. Actually, this should be the same underlying assumption in the kd-tree algorithm [Gandoin and Devillers 2002] since it does not encode the orientation information that cannot be simply inferred from the local context of each vertex split or generalized vertex split, especially for non-manifold meshes.

We can construct the facets for each vertex split among the updated local neighborhood as follows.

1. For each facet existing before the vertex split, denoted by $\triangle A_1 A_2 A_3$ with $A_i \in \{v, N_1, \ldots, N_M\}$, $i = 1,2,3$, we consider the following scenarios. If $A_i \neq v$, $i = 1,2,3$, do nothing. Otherwise, without loss of generality, we assume $A_1 = v$ and do the following.
   (a) If both A_2 and A_3 are connected to v_1, add $\triangle v_1 A_2 A_3$.
   (b) If both A_2 and A_3 are connected to v_2, add $\triangle v_2 A_2 A_3$.
   (c) Delete $\triangle A_1 A_2 A_3$.
2. If v_1 and v_2 are connected, for each pivot vertex P_i, $i = 1, \ldots, P$, add $\triangle v_1 v_2 P_i$.

To give an example, let us examine the vertex split as shown in Fig. 4. Prior to the vertex split, there are six local facets as shown in Fig. 4(a). After the vertex split, $\triangle N_1 N_2 v$ is replaced by $\triangle N_1 N_2 v_1$ since N_1 and N_2 are both connected to v_1, $\triangle N_6 N_1 v$ is replaced by $\triangle N_6 N_1 v_1$ and $\triangle N_6 N_1 v_2$ since N_6 and N_1 are connected to both v_1 and v_2 as shown in Fig. 4(b). Other facets existing before the vertex split are similarly updated after the vertex split. Furthermore, three new facets are added, which are $\triangle v_1 v_2 N_1$, $\triangle v_1 v_2 N_3$ and $\triangle v_1 v_2 N_6$, since v_1 and v_2 are connected and N_1, N_3 and N_6 are pivot vertices.

The above algorithm constructs all possible triangular facets from the edge-based connectivity. On one hand, when the mesh is fully restored, all the original facets have been constructed. On the other, the reconstructed mesh may contain facets that do not exist in the original mesh. This may happen only when there are boundary loops with exactly three distinct vertices after quantization, which will be treated as valid triangular facets in our algorithm. In typical

meshes, this problem rarely occurs since most boundary loops (if there is any) contain more than three vertices after quantization.

To solve the invalid facet problem, we can apply a pre-processing step by adding extra vertices to each problematic boundary loop so that it has more than three vertices after quantization. Note that when the quantization resolution is not sufficiently high, vertices on a problematic boundary loop may be very close to each other after quantization, adding extra vertices may introduce significant distortion to that boundary loop. This phenomenon is however more a problem of the quantization resolution than one of our algorithm.

3.5 Discussion

The proposed connectivity encoder is general in the sense that, independent of facet construction, the underlying edge-based connectivity coding can be used to compress aibitrary connectivity among a set of 3D points. Thus, we can extend it to encode the connectivity data of arbitrary polygonal meshes by modifying the facet construction procedure. That is, for each vertex split, we have to take care of the update of existing facets and the generation of new facets.

If the orientation information of each facet is required (*e.g.*, the facet is no longer double-sided), we need extra coding bits. Then, we may use a flag bit for each newly generated facet to specify its orientation. To improve the efficiency of flag bit coding, a local prediction scheme could be performed, which is expected to be highly accurate for manifold or "almost manifold" meshes.

4 Coding Performance Analysis

4.1 Geometry Coder

For the ease of analysis, we focus on the case that the OT of the vertex data is expanded level by level without using prioritized cell subdivision. Similar to [Devillers and Gandoin 2000], we divide the whole cell subdivision process into the following two stages.

1. Vertex separation. Cells are recursively subdivided until the OT level is reached where there is at most 1 vertex in each tree-front cell.
2. Position finalization. The position of each vertex is further refined with recursive cell subdivision, until the finest resolution allowed by the coordinate quantization is reached.

They are examined in detail below.

Vertex Separation. For an arbitrary mesh, the vertex distribution can be quite random. Therefore, it is difficult, if not impossible, to conduct an analysis that is applicable to all kinds of meshes. For regularly sampled manifold or "almost manifold" meshes, it is however often true that a cell subdivision leads to four nonempty child-cells on the average, if the vertex density is sufficiently high. A similar observation was also made in [Botsch et al. 2002]. To facilitate the analysis, we concentrate on manifold or "almost manifold" meshes. Furthermore, we have the following two assumptions.

1. The total vertex number n is an integral power of four, *i.e.*, $n = 4^K$, where K is a positive integer.
2. Each cell subdivision in the vertex separation stage leads to exactly four nonempty child-cells.

In the analysis, we label the OT levels increasingly from the root to the leaves, where the root is labeled as the 0th level. Since there are $N_i = 4^i$ cells in the ith level ($0 \le i \le K$), there are $I_i = 4^{K-i}$ vertices in each cell of the ith level on the average. Since each cell subdivision leads to four nonempty child-cells, the entropy coding of T values costs 0 bit. The nonempty-child-cell tuple coding costs $\log_2 K_4 = 6.13$ bits per cell subdivision, when entropy coded with no prediction. Thus, the number of coding bits is calculated as

$$C_O = 6.13 \sum_{i=0}^{K-1} 4^i \approx \frac{6.13}{3} \times 4^K = 2.04n. \quad (1)$$

To compare the performance of the proposed geometry coder and the kd-tree geometry coder, let us estimate the lower bound of coding bits with the kd-tree geometry encoder under the same assumptions. Corresponding to each OT cell subdivision, there exist a sequence of kd-tree cell subdivisions that achieve the same effect. Of all the possible OT cell subdivisions that lead to four nonempty child-cells, the corresponding kd-tree coding is most efficient when the first kd-tree subdivision allocates all vertices into one kd-tree child-cell, the second allocates two vertices into one child-cell and the remaining into the other, the third and the fourth subdivide the associated vertex sets into two nonempty subsets, respectively. Thus, if there are I vertices in an OT cell that has four nonempty child-cells, with the arithmetic coder, the number of kd-tree coding bits is, at the best,

$$B(I) = 2log_2(I+1) + \log_2(I-1) + \log_2 3 > 3\log_2 I + \log_2 3. \quad (2)$$

Thus, the lower bound number for the coding bits required by the kd-tree geometry encoder can be found by

$$C_K = \sum_{i=0}^{K-1} 4^i B(4^{K-i}) > (\frac{\log_2 3}{3} + \frac{8}{3})(4^K - 1) - 2K \approx 3.19n, \quad (3)$$

where $B(\cdots)$ is computed based on Eq. (2). Comparing Eqs. (1) and (3), we see a substantial performance gain of the proposed geometry coder over the kd-tree geometry coder.

Position Finalization. Consider the case where the quantization resolution is L bits per coordinate. That corresponds to $L+1$ OT levels in total, including the root level. After the first stage, in which the OT is fully expanded to the Kth level, we still need to subdivide the tree-front cells through $L-K$ iterations to characterize the final position of each vertex. Since there is only one vertex in each tree front cell, the proposed geometry coder and the kd-tree geometry encoder need 3 bits per cell subdivision. Therefore, they cost the same number of bits

$$C_F = (L-K) \times n \times 3 = 3(L - \log_4 n)n \quad (4)$$

to specify the final position of each vertex. Let the total geometry coding costs for the proposed geometry coder and the kd-tree geometry coder be $C_{O,T}$ and $C_{K,T}$, respectively. Then, we have

$$C_{O,T} = C_O + C_F = (3(L-K) + 2.04)n, \quad (5)$$

$$C_{K,T} \ge C_K + C_F > (3(L-K) + 3.19)n. \quad (6)$$

From Eqs. (5) and (6), we see that the proposed geometry coder is more efficient than the kd-tree geometry coder. Furthermore, it is easy to see that if finer resolutions (*i.e.*, larger values of $L-K$) are needed for mesh reconstruction, the coding gains of both geometry coders become less, and the coding gain of the proposed geometry coder over the kd-tree geometry coder becomes less significant due to the fixed overhead of the $3(L-K)$ term.

4.2 Connectivity Coder

Let us assume that the average valence of all vertices generated during the whole coding process is V, and the average number of pivot vertices in all vertex splits is P. Using the proposed connectivity coder, in each vertex split, by rough estimation, we need

1. $\log_2(V+1)$ bits for the coding of pivot vertex number,
2. $\log_2 C_P^V$ bits for the coding of pivot-vertex-tuple,
3. 1 bit for the coding of new vertices' adjacency, and
4. at most $V-P$ bits for the nonpivot neighbor assignment.

Overall, the average number of bits for each vertex split is equal to

$$C_n = \log_2(V+1) + \log_2 C_P^V + V - P + 1.$$

For manifold or "almost manifold" meshes, almost all vertex splits have two pivot vertices and almost all new vertices are connected in the refined mesh. Thus, the coding of pivot vertex number and new vertices' adjacency can be done efficiently. Furthermore, with the proposed method, the coding of nonpivot assignment is also very efficient. Generally speaking, for each vertex split, the coding cost mainly comes from the coding of the pivot-vertex-tuple, which needs $\log_2 C_P^V$ bits, while the coding of all other information takes less than 1 bit. In our experiments, V is around 7, P is about 2 for manifold or "almost manifold" meshes. Thus, the average coding cost is about 5 bits per vertex split, which can be even reduced after the use of prediction and arithmetic coding. Since each vertex split introduces a new vertex, the connectivity coding cost per vertex split is roughly equal to the connectivity coding cost per vertex.

5 Experimental Results

For the purpose of comparison, we have implemented the kd-tree geometry coder ourselves. It yields results close to those reported in [Gandoin and Devillers 2002] with an difference of about 5%. In this section, we primarily focus on the comparison of geometry coding since the geometry data dominate the compressed file size in most cases. In our experiments, almost all mesh vertices are quantized with 12 bits per coordinate, with an exception of the 'fandisk' mesh, whose vertices are quantized with 10 bits per coordinate to be consistent with [Gandoin and Devillers 2002] for fair comparison. All the triangle soups (*i.e.*, the meshes 'skeleton', 'm_tree6', 'm_tree8', 'plant06_s', 'plant12_s', and 'aqua05') are obtained from the 3DCafe website, *http://www.3dcafe.com/asp/meshes.asp*. The test meshes are organized into three classes: class A, class B and class C. This classification is based on the range of improvement of the proposed algorithm over the kd-tree algorithm in terms of geometry coding efficiency. The ranges of improvement are 5%∼15%, 15%∼25% and 25%∼100% for meshes in classes A, B and C, respectively.

Experimental results are listed in Table 1, where the mesh class, the mesh name and the number of vertices in each mesh are listed in the first three columns. Then, we compare the coding bit rates (in the unit of bpv) for two algorithms, namely, the kd-tree mesh coder [Gandoin and Devillers 2002] and the mesh coder proposed in this work. They are denoted by KT and OT, respectively. For each algorithm, we report the geometry and the connectivity coding costs separately. In other words, bit rates for the geometry coding are listed in the 4th and the 6th columns while those for the connectivity coding are listed in the 5th and the 7th columns. Among the geometry bit rates of KT, those marked with '*' are taken from the original paper [Gandoin and Devillers 2002], while others are obtained with our own implementation of the kd-tree geometry coder. For meshes that are not tested in [Gandoin and Devillers 2002], their connectivity coding bit rates are indicated by '–' in the table. For each mesh in Table 1, if its geometry coding bit rates are G_{KT} and G_{OT} for KT and OT, respectively, then the geometry coding gain (GG) of OT over KT is calculated by

$$GG = \frac{G_{KT} - G_{OT}}{G_{KT}} \times 100\%.$$

The higher geometry coding gain of the proposed OT algorithm for meshes in classes B and C is attributed to their underlying properties: (i) tighter vertex clustering and (ii) higher regularity of geometry data. To measure whether a mesh has tight vertex clustering, we can examine the number of OT decomposition levels required to reach the stage of one-vertex-per-cell. A larger number of levels implies tighter clustering of vertices. As explained in Sec. 4.1, compared with the kd-tree algorithm, the proposed geometry coder has higher efficiency in the vertex separation stage and about the same efficiency in the position finalization stage. Therefore, the tighter the vertex clustering is, the more efficient the proposed geometry coder will be. For instance, for the 'bunny' mesh in class A, it needs about 8 OT decomposition levels to reach the one-vertex-per-cell stage. For the 'rabbit' mesh in class B, it demands about 9 OT decomposition levels to reach the same stage. Finally, for the 'plant06_s' model in class C, even when the OT is fully expanded, there are still about 3.5 vertices per cell on the average. Thus, it has the tightest vertex clustering among the three. By examining mesh 'plant06_s' as rendered in Fig. 1, we see that most vertices are clustered around stems and leaves. As shown in the last column of Table 1, the geometry coding gains of OT over KT are 10.8%, 21.9% and 61.2% for the 'bunny', 'rabbit' and 'plant06_s' models, respectively.

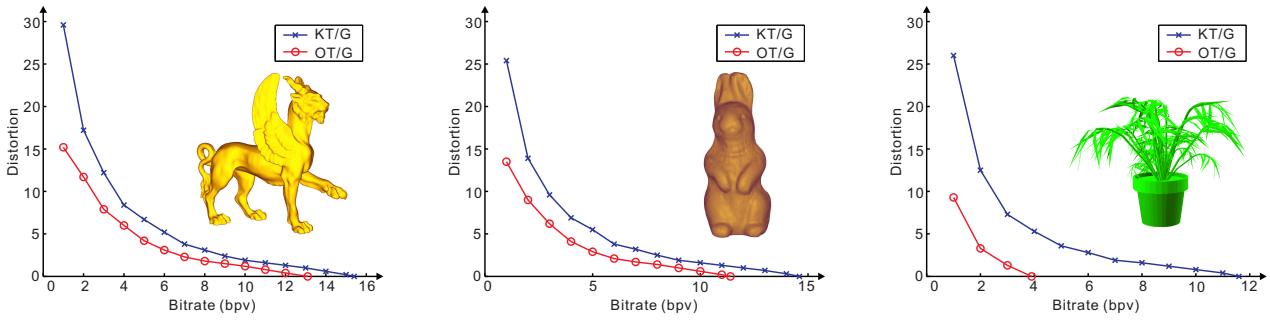
Table 1: Bit rates (in bpv) for geometry and connectivity coding.

Class	Mesh	#v	KT/G	KT/C	OT/G	OT/C	GG
A	skeleton	6,308	15.9*	11.4	14.8	10.3	6.9%
	bunny	35,947	14.8*	3.1	13.2	2.7	10.8%
	fandisk	6,475	12.1*	2.9	10.7	2.6	11.6%
	feline	49,864	15.4	–	13.1	3.6	14.9%
B	horse	19,851	16.4*	3.9	13.7	2.9	16.5%
	m_tree6	19,616	15.9	–	12.9	7.9	18.9%
	m_tree8	54,223	13.3	–	10.6	9.7	20.3%
	rabbit	67,039	14.6	–	11.4	3.4	21.9%
C	tore high	36,450	16.9	–	8.9	2.9	47.3%
	plant06_s	21,582	13.9	–	5.4	2.5	61.2%
	plant12_s	36,423	11.6	–	3.9	2.1	66.4%
	aqua05	16,784	16.4*	8.5	5.3	3.1	67.7%

By examining the geometry bit rates of the proposed OT algorithm for manifold meshes in Table 1 (including 'bunny', 'feline', 'horse', and 'rabbit'), we observe that the more vertices a manifold mesh has, the more efficient the geometry coding will be with respect to a fixed quantization resolution. This is largely true since the more vertices a manifold mesh has, the more OT decomposition levels are needed to reach the one-vertex-per-cell stage, which leads to higher coding efficiency according to the analysis given in Sec. 4.1. However, the 'tore high' mesh is an exception to this general rule. The high coding efficiency of OT mainly comes from its highly regular geometry data. Note that the 'tore high' mesh is a high resolution mesh of torus with ideal geometry and connectivity regularity. The regular geometry leads to regular vertex distribution and good prediction accuracy, which both contribute to high efficiency in entropy coding.

To compare the rate-distortion (R-D) performance, we plot the R-D curves for three meshes ('feline', 'rabbit', and 'plant12_s') for the kd-tree geometry coder (KT/G) and the proposed OT geometry coder (OT/G) in Fig. 7. The distortion of any intermediate mesh is measured by the average distance between the original vertices (after 12-bit quantization) and their representative vertices. Note that the three meshes are representatives from classes A, B and C, respectively. We see from Figs. 7(a)-(c) that, as compared with the KT/G coder, the proposed OT/G coder produces significantly less distortion at all bit rates for all three meshes. The gain is particularly obvious at low bit rates. Furthermore, the R-D advantage increases from Fig. 7(a) to Fig. 7(c).

Some visual examples of progressive mesh reconstruction based on the 12-bit coordinate quantization are given in Fig. 1. For the rendering, no smoothing was done and the Phong shading was used. For each reconstructed mesh, its total bit rate (including the bit rates for both geometry and connectivity coders) and the percentage of the compressed data size over the original data size are given in subcaptions. The original data size is calculated using 3×12 bits per vertex and $3\times\log_2 n$ bits per triangle, assuming there are n vertices in the original mesh. Generally speaking, the reconstructed mesh is indistinguishable from the original mesh at a data size that is less



(a) (b) (c)

Figure 7: The R-D performance comparison between the OT/G and the KT/G coders for three meshes: (a) feline, (b) rabbit and (c) plant12_s.

than 10% of the original one.

As to connectivity coding efficiency, the OT/C algorithm also outperforms the kd-tree connectivity coder (KT/C) as shown in Table 1. The gain varies from 10% (for 'skeleton', 'bunny' and 'fandisk') to 60% (for 'aqua05').

6 Conclusion and Future Work

An octree(OT)-based progressive lossless 3D mesh encoder was proposed in this work to process 3D meshes of any topology. Instead of encoding the vertex numbers in each subdivided child-cells directly, we examine an alternative way to encode the change of the geometry information. Besides, extensive prediction schemes were developed to make various arithmetic coders more efficient. It was demonstrated that the proposed mesh coder achieves the state-of-the-art coding performance.

Our main experimental results can be summarized as follow. The proposed OT coder is superior to the KT coder in both geometry and connectivity coding efficiency. For the geometry coding, the improvement depends on the characteristics of the underlying model. Generally speaking, if the model has tighter vertex clustering and/or more regular geometry data, more improvement can be observed. The improvement of geometry coding may go as high as 50%∼60% for meshes with tight vertex clustering and/or high regularity.

In the future, we will generalize various ideas in this work to encode point-based geometry data, since the point-based representation is a popular representation of 3D graphics data and the huge amount of point-based geometry data demands efficient coding.

References

ALLIEZ, P., AND DESBRUN, M. 2001. Progressive encoding for lossless transmission of triangle meshes. In *ACM SIGGRAPH*, 198–205.

ALLIEZ, P., AND DESBRUN, M. 2001. Valence-driven connectivity encoding for 3D meshes. In *EUROGRAPHICS*, 480–489.

ALLIEZ, P., AND GOTSMAN, C. 2003. Recent advances in compression of 3d meshes. In *Proceedings of the Symposium on Multiresolution in Geometric Modeling*.

ATTENE, M., FALCIDIENO, B., SPAGNUOLO, M., AND ROSSIGNAC, J. 2003. Swingwrapper: Retiling triangle meshes for better edgebreaker compression. *ACM Transactions on Graphics 22*, 4, 982–996.

BAJAJ, C., PASCUCCI, V., AND ZHUANG, G. 1999. Progressive compression and transmission of arbitrary triangular meshes. In *IEEE Visualization*, 307–316.

BAJAJ, C. L., PASCUCCI, V., AND ZHUANG, G. 1999. Single resolution compression of arbitrary triangular meshes with properties. *Computational Geometry: Theory and Applications 14*, 167–186.

BOTSCH, M., WIRATANAYA, A., AND KOBBELT, L. 2002. Efficient high quality rendering of point sampled geometry. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*, 53–64.

COHEN-OR, D., LEVIN, D., AND REMEZ, O. 1999. Progressive compression of arbitrary triangular meshes. In *IEEE Visualization*, 67–72.

COORS, V., AND ROSSIGNAC, J. 2004. Delphi: geometry-based connectivity prediction in triangle mesh compression. *The Visual Computer 20*, 8-9, 507–520.

DEVILLERS, O., AND GANDOIN, P. 2000. Geometric compression for interactive transmission. In *IEEE Visualization*, 319–326.

GANDOIN, P. M., AND DEVILLERS, O. 2002. Progressive lossless compression of arbitrary simplicial complexes. *ACM Trans. Graphics 21*, 3, 372–379.

GOTSMAN, C., GUMHOLD, S., AND KOBBELT, L. 2002. Simplification and compression of 3D meshes. In *Tutorials on Multiresolution in Geometric Modelling*.

GUMHOLD, S., AND STRASSER, W. 1998. Real time compression of triangle mesh connectivity. In *ACM SIGGRAPH*, 133–140.

HOPPE, H. 1996. Progressive meshes. In *ACM SIGGRAPH*, 99–108.

KARNI, Z., AND GOTSMAN, C. 2000. Spectral compression of mesh geometry. In *ACM SIGGRAPH*, 279–286.

KHODAKOVSKY, A., AND GUSKOV, I. 2000. Normal mesh compression. *Preprint*.

KHODAKOVSKY, A., SCHRÖDER, P., AND SWELDENS, W. 2000. Progressive geometry compression. In *ACM SIGGRAPH*, 271–278.

LANEY, D., BERTRAM, M., DUCHAINEAU, M., AND MAX, N. 2002. Multiresolution distance volumes for progressive surface compression. In *Proceedings of the First International Symposium on 3D Data Processing, Visualization, and Transmission*, 470–479.

LEE, H., DESBRUN, M., AND SCHRÖDER, P. 2003. Progressive encoding of complex isosurfaces. In *ACM SIGGRAPH*.

LI, J., AND KUO, C.-C. J. 1998. Progressive coding of 3-D graphic models. *Proceeding of the IEEE 86*, 6 (Jun), 1052–1063.

PAJAROLA, R., AND ROSSIGNAC, J. 2000. Compressed progressive meshes. *IEEE Trans. Visualization and Computer Graphics 6*, 1, 79–93.

PENG, J., KIM, C.-S., AND KUO, C.-C. J. Technologies for 3D mesh compression: A survey. *to appear in Journal of Visual Communication and Image Representation*.

POPOVIC, J., AND HOPPE, H. 1997. Progressive simplicial complexes. In *ACM SIGGRAPH*, 217–224.

ROSSIGNAC, J., AND BORREL, P. 1993. *Geometric Modeling in Computer Graphics*. Springer-Verlag, Jul.

ROSSIGNAC, J. 1999. Edgebreaker: Connectivity compression for triangle meshes. *IEEE Trans. Visualization and Computer Graphics 5*, 1, 47–61.

SAUPE, D., AND KUSKA, J.-P. 2001. Compression of isosurfaces for structured volumes. In *Proceedings of Vision, Modeling and Visualization*, 333–340.

SCHMALSTIEG, D., AND SCHAUFLER, G. 1997. Smooth levels of detail. In *IEEE Virtual Reality Annual International Symposium*, 12–19.

SZYMCZAK, A., ROSSIGNAC, J., AND KING, D. 2002. Piecewise regular meshes: construction and compression. *Graphical Models 64*, 3/4, 183–198.

TAUBIN, G., AND ROSSIGNAC, J. 1998. Geometric compression through topological surgery. *ACM Trans. Graphics 17*, 2, 84–115.

TAUBIN, G., GUEZIEC, A., HORN, W., AND LAZARUS, F. 1998. Progressive forest split compression. In *ACM SIGGRAPH*, vol. 32, 123–132.

TOUMA, C., AND GOTSMAN, C. 1998. Triangle mesh compression. In *Proceedings of Graphics Interface*, 26–34.