Abstract—A progressive lossless 3D mesh encoder using the octree-based space partitioning of vertex data is proposed in this work. Given a 3D mesh, the quantized 3D vertices are first partitioned into an octree 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 the local geometry and connectivity changes are encoded. Furthermore, selective cell subdivision is performed in the tree front to provide better rate-distortion performance. It is shown by experimental results that the geometry coding costs of the proposed codec are 4.2 and 14.5 bits per vertex (bpv) for 8-bit and 12-bit coordinate quantizations, respectively, and the connectivity coding cost is 3.3 bvp on the average, which provide the state-of-the-art coding performance.

I. INTRODUCTION

Graphics data are widely used in multimedia applications such as video gaming, engineering design, architectural walkthrough, virtual reality, e-commerce and scientific visualization, etc. 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, texture coordinates, etc. The majority of the 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 focused on the connectivity coding which guides the geometry coding in turn. However, since the geometry data are dominant in the compressed file size in most cases, algorithms focusing on the geometry coding, which even guides the connectivity coding, have emerged in recent years [1], [2]. Furthermore, the very early research on 3D mesh compression focused on single-rate compression techniques to save the bandwidth between the CPU and the graphics card. 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).

Among the single-rate lossless mesh codecs, the best achievable bitrate is 6~10 bpv for the geometry coding, at a quantization resolution of 8 bits per coordinate, and less than 3 bpv for the connectivity coding [3], [4]. As for the progressive mesh coding, there exist lossy codecs [5], [6] and lossless codecs [1], [2], [7], [8]. Among the progressive lossless mesh codecs, the kd-tree-based approach [1], [2] produces the best results. As reported, its geometry coding cost is 15.7 bpv at the quantization resolution of 12/10 bits per coordinate, and its connectivity coding cost is 3.5 bvp, on the average. In this work, we propose an octree-based progressive lossless mesh encoder. It deals with 3D meshes of any topology, and achieves excellent coding efficiency. Our experiments demonstrate that the average geometry coding costs are 4.2 bvp and 14.5 bvp with respect to the 8-bit and 12-bit coordinate quantization schemes, respectively, and the average connectivity coding cost is 3.3 bvp. Thus, the proposed new codec provides the state-of-the-art performance.

II. OCTREE-BASED GEOMETRY ENCODER

A. Drawbacks of the kd-tree-based algorithm

The kd-tree-based mesh encoder [1], [2] employs a kd-tree decomposition based on 3D cell subdivisions [2]. When it subdivides a cell into two child cells, it encodes the number of vertices in one of the two child cells. If the parent cell contains \( p \) vertices, the number of vertices in one of the child cells can be encoded using \( \log_2(p + 1) \) bits with an arithmetic coder. This subdivision is recursively applied until each non-empty cell is small enough to contain only one vertex and enables a sufficiently precise reconstruction of the vertex position. Corresponding to each cell subdivision, the local connectivity refinement is encoded using either the vertex split [7] or the generalized vertex split [8]. Since the vertex to split next is implied by the next cell subdivision, there is no need to explicitly encode the index of that vertex.

One drawback of the kd-tree-based algorithm is the high overhead of the vertex number encoding at higher tree levels (i.e., levels closer to the root). During the progressive mesh transmission and reconstruction process, what really matters is not the actual number of vertices in each cell, but whether each cell is empty or not, since all vertices in a nonempty cell are approximated with only one point, i.e. the centroid of the nonempty cell in any LOD. Cells have more vertices at higher levels so that the kd-tree algorithm needs more coding bits although the exact vertex number in each cell is not needed for the purpose of progressive mesh reconstruction. The other drawback of the kd-tree-based algorithm is that cells in the tree front are subdivided one by one without any discrimination.
The division of some cells may result in a higher distortion decrease while demanding the same amount of coding bits. Then, it is desirable to assign them a higher priority in division due to their better rate-distortion performance.

B. Overview of the proposed algorithm

Given a 3D mesh, the vertex coordinates are uniformly quantized, and partitioned into an octree structure whose depth equals to the number of quantization bits per coordinate. After that, the octree is traversed from the root to leaves 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 along the X, Y and Z axes, respectively. Nonempty child cells are recursively subdivided but empty child cells are truncated.

For each cell subdivision, we encode both the local geometry change and the local connectivity change. For the former, we specify which child cells are nonempty using the approach described in Subsection II-C. For the latter, we specify the connectivity between nonempty child cells, and the connectivity between the nonempty child cells and the parent cell’s neighbor cells using the approach described in Section III.

C. Nonempty-child-cell-tuple coding

1) Basic algorithm: For each cell subdivision, instead of encoding the vertex numbers in the newly generated child cells, we only specify which child cells are nonempty. This is the major difference between our algorithm and the kd-tree algorithm. If there are $T$ nonempty child cells after a cell subdivision, the indices of all nonempty child cells form a nonempty-child-cell-tuple, $(t_1, \ldots, t_i, \ldots, t_T)$, with $t_i \in \{1, 2, \ldots, 8\}$, which is a $T$-tuple. To specify which child cells are nonempty, we first record the tuple number $T$, and then record the nonempty-child-cell-tuple.

The numbers of nonempty child cells, $T$, are arithmetic coded using the parent cell’s octree level as the context. This is done based on the observation that cells in the same octree level tend to have similar numbers of nonempty child cells. The set of the nonempty-child-cell-tuple is also arithmetic coded, using the parent cell’s number of nonempty child cells as the context, since, for cells with the same number of nonempty child cells (e.g. $T$), there are the same number, $K_T = C_8^T = 8!/(T! \times (8-T)!)$, of possible $T$-tuples in total.

2) Nonempty-child-cell-tuple probability estimation: Assuming that a cell has $T$ nonempty child cells. To further improve coding efficiency, we can estimate the probability of each $T$-tuple’s being the nonempty-child-cell-tuple, and pass the estimated probability table to the arithmetic coder. First, we compute a priority value for each child cell, which represents our estimation of its probability of being nonempty. This estimation is possible since the more nonempty neighbor cells a child cell has, the more likely it will not be empty, either.

As mentioned earlier, a parent cell is subdivided into eight child cells through three cell bi-partitionings. Associated with each cell bi-partitioning, the partitioning plane also partitions the neighbor cells into two subsets. In the following, we denote each neighbor cell bi-partitioning as $b_{pi}$, $i = 1, 2, 3$, which partitions the neighbor cells into two subsets, $S_{i,1}$ and $S_{i,2}$, containing $n_{i,1}$ and $n_{i,2}$ neighbor cells, respectively. For each $b_{pi}$, we can calculate its extent of unbalance as

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

Sorting $u_i$, $i = 1, 2, 3$, in a descending order, we obtain $u_{i_1}$, $i_1 = 1, 2, 3$ and $i_j = 1, 2, 3$, such that $u_{i_1} \geq u_{i_2} \geq u_{i_3}$. Generally speaking, the more unbalanced a bi-partitioning is, the more helpful it is for nonempty child cell prediction, and the more weight we should give to that bi-partitioning. Generally, we assign to each neighbor cell bi-partitioning, $b_{pi}$, a weight as

$$w_{i_1} = 3, \quad w_{i_2} = 2, \quad w_{i_3} = 1.$$  

After cell bi-partitioning $b_{pi}$, $i = 1, 2, 3$, if child cell $c_k$, $k = 1, \ldots, 8$, is located at the same side of subset $S_{i,k_1}$, $k_1 = 1, 2$, of the bi-partitioning plane, its priority $p_k$ is calculated as

$$p_k = \sum_{i=1}^{3} (w_i \times n_{i,k}).$$

![Fig. 1. A 2D example of priority value calculation.](image-url)

The priority calculation with a 2D example is illustrated in Fig. 1. A 2D cell, $c$, and its neighbor cells are given in Fig. 1(a). The bi-partitioning, $b_{p_1}$, partitions the neighbor cells into two subsets with $n_{2,1} = 3$ and $n_{2,2} = 2$ cells as shown in Fig. 1(b). Another bi-partitioning, $b_{p_2}$, partitions neighbor cells into two subsets with $n_{3,2} = 4$ and $n_{3,3} = 1$ as shown in Fig. 1(c). Since $b_{p_2}$ is more unbalanced than $b_{p_1}$, we assign more weight to $b_{p_2}$. Thus, we have $w_1 = 1$ and $w_2 = 2$. Then,
we can calculate the priority for each child cell. For example, the priority value for child cell \( c_1 \) is
\[
p_1 = w_1 n_{1,1} + w_2 n_{2,1} = 3 + (2 \times 4) = 11.
\]
The priority values for other child cells can be similarly calculated. They are shown in Fig. 1(d).

Having calculated the priority \( p_i \) for each child cell \( c_i \), we can calculate the pseudo-probability \( PP_i \) for each \( T \)-tuple as
\[
PP_i = \sum_{j=1}^{T} p_{ij},
\]
where \( p_{ij} \) is the priority for cell \( i_j \) in the \( T \)-tuple \( TP_i = (i_1, i_2, \ldots, i_T) \). By normalizing the sum of all \( PP_i \)’s to unity, we obtain the estimated probabilities \( EP_i \)’s for all cells in the \( T \)-tuple. As compared with the direct nonempty-child-tuple coding without probability estimation, the coding of the nonempty-child-cell-tuple with the proposed probability estimation technique improves coding efficiency by more than 15% on the average in our experiments.

D. 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 important cells earlier to provide better mesh quality at lower bitrates. The key issue is how to define the cell importance. First, let us define the valence of a cell to be the number of edges adjacent to the cell’s representative vertex in the current intermediate mesh. Generally speaking, we have the following 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; and (3) a shorter average length of adjacent edges implies more impact of cell subdivision to local curvature refinement. Thus, for each cell \( c \), we define its importance value \( I(c) \) as a function of the cell valence \( v \), the cell size \( s \), and the average length \( l \) of edges adjacent to the cell’s representative vertex in the current intermediate mesh.

\[
I(c) = \frac{v s}{l}.
\]

After each cell subdivision, we update the cell importance list, and move to the cell with the highest value in the updated list for the next subdivision task.

III. OCTREE-BASED CONNECTIVITY ENCODER

A. Overview of the algorithm

For each octree-based cell subdivision that subdivides a cell into \( K \) \((K = 2, \ldots, 8)\) nonempty child cells, we use \( K - 1 \) kd-tree subdivisions to simulate it, where each kd-tree cell subdivision divides a nonempty child octree cell into two child cells. Furthermore, we deal with the connectivity coding in a unified way, without discriminating between the vertex split and the generalized vertex split, which are both called the vertex split in our work. For each vertex split that corresponds to a kd-tree cell subdivision, let us denote the neighbor vertices before the vertex split as \( N_i, i = 1, 2, \ldots, M \), where \( M \) is the number of neighbor vertices, and the two new vertices resulted from the vertex split as \( v_1 \) and \( v_2 \). Then, we need to encode the following information associated with this vertex split.

1) The vertices in \( N_i \) that are connected to both \( v_1 \) and \( v_2 \). They are called the pivot vertices.
2) For each of the rest vertices in \( N_i \), whether it is connected to \( v_1 \) or \( v_2 \).
3) Whether \( v_1 \) and \( v_2 \) are adjacent in the refined mesh.

Of the above three types of information, the second one is arithmetic coded and the third is also arithmetic coded using spatial-distance prediction. All together, the cost of the second and the third types of information is less than 0.5 bpv in our experiments. The coding of the first type of information is discussed in the next subsection.

B. Pivot-vertex-tuple selection

To encode the neighbor vertices in \( N_i, i = 1, 2, \ldots, M \), that are connected to both of new vertices \( v_1 \) and \( v_2 \), we employ a method similar to the nonempty-child-cell-tuple coding method used in geometry coding.

Corresponding to the priority calculation for each child cell in the nonempty-child-cell-tuple coding, we need to calculate the priority, \( p_i, i = 1, 2, \ldots, M \), for each neighbor vertex \( N_i \). 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 triangle \( \Delta N_i v_1 v_2 \) is and the closer \( N_i \) is to the middle perpendicular plane, \( P \), of segment \( v_1 v_2 \), the more probable that \( N_i \) will be a pivot vertex. Observing that, for a given perimeter, the bigger the area, the more regular a triangle will be, we define the regularity \( r_i \) of \( \Delta N_i v_1 v_2 \) as
\[
r_i = \frac{\sigma_i}{2s} = \frac{\sqrt{(s-a)(s-b)(s-c)}}{2s},
\]
where \( a, b, \) and \( c \) are the lengths of the three edges in \( \Delta N_i v_1 v_2 \), respectively, \( \sigma_i \) is the area of \( \Delta N_i v_1 v_2 \) and \( s = (a+b+c)/2 \). If the distance of \( N_i \) to plane \( P \) is \( d_i \), the priority \( p_i \) of \( N_i \) is calculated as
\[
p_i = w_1 r_i + w_2 \frac{1}{d_i},
\]
where \( w_1 \) and \( w_2 \) are weights determined by experiments.

IV. EXPERIMENTAL RESULTS

Both 8-bit and 12-bit vertex coordinate quantization schemes were used in our experiments. The bitrates (in the unit of bpv) for geometry coding using the kd-tree algorithm(KDT) and the proposed octree algorithm (OCT) are listed in Tables I and II for the 8-bit and 12-bit quantization schemes, respectively. In these tables, the first column shows the mesh name, the second column shows the number of vertices in each mesh, and the remaining show the bitrates. The ‘subaverage’ row gives the average bitrates of the KDT and OCT algorithms using the same set of test meshes, while the ‘average’ row gives the average bitrates for a larger set of test meshes. Since we do not have the original executable for the KDT approach, some data are missing which are denoted as N/A in the table.

We see from Tables I and II that the average bitrates of the proposed octree geometry encoder are superior to the kd-tree approach [1], [2]. It is worthwhile mentioning that the reported average geometry coding performance, 15.7 bpv, in
The bitrates for the kd-tree connectivity encoder (KDT) and the octree connectivity encoder (OCT) are listed in Table III for comparison. We see that the average bpv of the OCT approach is about 15% less than that associated with the kd-tree-based algorithm.

TABLE I
Comparison of geometry encoding bit rate (bpv) for 8-bit coordinate quantization.

<table>
<thead>
<tr>
<th>Mesh</th>
<th>#v</th>
<th>KDT</th>
<th>OCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>blob</td>
<td>8,036</td>
<td>8.9</td>
<td>7.2</td>
</tr>
<tr>
<td>shape</td>
<td>2,562</td>
<td>12.1</td>
<td>11.1</td>
</tr>
<tr>
<td>cow</td>
<td>3,066</td>
<td>8.9</td>
<td>7.2</td>
</tr>
<tr>
<td>dumptruck</td>
<td>11,738</td>
<td>4.9</td>
<td>4.3</td>
</tr>
<tr>
<td>subaverage</td>
<td></td>
<td>7.4</td>
<td>6.2</td>
</tr>
<tr>
<td>horse</td>
<td>19,851</td>
<td>N/A</td>
<td>3.9</td>
</tr>
<tr>
<td>dinosaur</td>
<td>14,070</td>
<td>N/A</td>
<td>4.3</td>
</tr>
<tr>
<td>crocodile</td>
<td>17,332</td>
<td>N/A</td>
<td>1.5</td>
</tr>
<tr>
<td>average</td>
<td>N/A</td>
<td>N/A</td>
<td>4.2</td>
</tr>
</tbody>
</table>

TABLE II
Comparison of geometry coding bit rate (bpv) for 12-bit coordinate quantization.

<table>
<thead>
<tr>
<th>Mesh</th>
<th>#v</th>
<th>KDT</th>
<th>OCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>blob</td>
<td>8,036</td>
<td>20.1</td>
<td>19.5</td>
</tr>
<tr>
<td>horse</td>
<td>19,851</td>
<td>16.4</td>
<td>13.6</td>
</tr>
<tr>
<td>triceratops</td>
<td>2,832</td>
<td>19.2</td>
<td>18.6</td>
</tr>
<tr>
<td>subaverage</td>
<td></td>
<td>17.6</td>
<td>15.6</td>
</tr>
<tr>
<td>head</td>
<td>11,703</td>
<td>N/A</td>
<td>14.8</td>
</tr>
<tr>
<td>dinosaur</td>
<td>14,070</td>
<td>N/A</td>
<td>15.8</td>
</tr>
<tr>
<td>crocodile</td>
<td>17,332</td>
<td>N/A</td>
<td>11.4</td>
</tr>
<tr>
<td>average</td>
<td>N/A</td>
<td>N/A</td>
<td>14.5</td>
</tr>
</tbody>
</table>

TABLE III
Connectivity coding bit rate (bpv) comparison.

<table>
<thead>
<tr>
<th>Mesh</th>
<th>#v</th>
<th>KDT</th>
<th>OCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>blob</td>
<td>8,036</td>
<td>4.1</td>
<td>3.7</td>
</tr>
<tr>
<td>horse</td>
<td>19,851</td>
<td>3.9</td>
<td>3.1</td>
</tr>
<tr>
<td>triceratops</td>
<td>2,832</td>
<td>6.0</td>
<td>5.2</td>
</tr>
<tr>
<td>fandisk</td>
<td>6,475</td>
<td>2.9</td>
<td>2.7</td>
</tr>
<tr>
<td>average</td>
<td></td>
<td>3.9</td>
<td>3.3</td>
</tr>
</tbody>
</table>

To compare the rate-distortion performance between the proposed octree-based geometry encoder and the kd-tree-based geometry encoder, we have implemented the kd-tree-based geometry encoder that yields results similar to those reported in [1], [2]. In Fig. 2, we plot the rate-distortion curves for the horse mesh using the kd-tree-based geometry encoder (KDT) and the proposed octree-based geometry encoder (OCT) with nonempty-child-cell-tuple coding. In this figure, the distortion of any intermediate mesh is measured as the average distance between the original vertices (after 12-bit quantization) and their representative vertices. Fig. 2(a) shows the full range of the curves, with the low-bitrate portion enlarged in Fig. 2(b). We see that the proposed octree-based encoder achieves less distortion at any given bitrate for the horse mesh, which is particularly obvious at low bitrates. As shown in Fig. 2(b), for any given bitrate between 0 and 8 bpv, the distortion associated with the proposed octree-based algorithm is only about one half of that associated with the kd-tree-based algorithm.

![Rate-distortion curves](image)

Fig. 2. The rate-distortion performance for the horse mesh, where the low-bitrate part of (a) is enlarged in (b).

V. CONCLUSION

An octree-based progressive lossless 3D mesh encoder was proposed to encode 3D meshes of any topology in this work. It was demonstrated that the proposed method achieves the state-of-the-art coding performance. On the average, the geometry coding cost is 14.5 bpv with the 12-bit coordinate quantization scheme. For the 8-bit coordinate quantization scheme, the average geometry coding cost is 4.2 bpv, which is even better than the best known single-rate mesh encoders. As to the connectivity coding, the proposed encoder achieves 3.3 bpv on the average, which is about 15% less than the kd-tree algorithm.

ACKNOWLEDGMENT

The authors would like to thank Chao-Hua Lee in the Department of Electrical Engineering, USC, for his helpful discussion and suggestion.

REFERENCES