SCALABLE POINT CLOUD GEOMETRY CODING WITH BINARY TREE EMBEDDED QUADTREE

Birendra Kathariya\textsuperscript{1}, Li Li\textsuperscript{1}, Zhu Li\textsuperscript{1}, Jose Alvarez\textsuperscript{2}, Jianle Chen\textsuperscript{2} *

\textsuperscript{1}University of Missouri-Kansas City, \textsuperscript{2}Futurewei Technologies Inc.

bkkvh8@mail.umkc.edu, \{lili,lizhu\}@umkc.edu, \{jose.roberto.alvarez,jianle.chen\}@huawei.com

ABSTRACT

Many applications of point cloud have recently been identified in automobile navigation system, visual communication, and so on. However, the huge data size of point cloud has been a bottleneck for the practical implementations. In this paper, we present a compression scheme that utilizes variable-rate coding of a same point cloud data at different quality. Point cloud is encoded at fixed-rate for highest representation. Encoder, however, can present variable-rate encoded data for any lowest to highest representation to decoder which is then decoded to reconstruct point cloud at different quality. Variable-rate encoding is achieved through the so-called binary tree quadtree (BTQT) scheme. The BTQT scheme made the compression more effective by dividing point cloud frame into blocks using binary-tree and encoding flat surfaces in the blocks by quadtree and non-flat surfaces by octree. Simulation results show that scalable coding solution can efficiently compress point cloud data at variable rate compensating the quality.

Index Terms— Binary-tree, Scalable coding, Octree, Quadtree, Point cloud.

1. INTRODUCTION

Point cloud technology recently got much attention as its application potential has been discovered in wide variety of field including immersive 3D teleconferencing, geospatial inspection, 3D modeling and printing, architectural design, autonomous navigation system etc. The emergence of miniaturized and inexpensive 3D scanner made it popular in the consumer level.

3D sensing technology has matured a lot. The data acquired from these high-speed 3D sensors consumes huge amount of storage space and transmission bandwidth. With the efficient compression scheme, the point cloud technology can be made more accessible to the consumer level and thus help to explore new frontier of its possible applications.

The size of a point cloud depends on the resolution or voxel size, the number of frames captured per second, and the surface area of an object or a scene scanned. Its size may also increase if 3D data constitutes one or more attributes like color, normal, reflectance, transparency etc. in addition to geometry. To compress the point cloud, every aspect of point cloud that contributes the size should be explored. Most works focus on compression of geometry and attributes separately. They use octree to generate the occupancy bitstream and represent the geometry. The octree is used mostly to compress the static point cloud or intra-code the frame in dynamic point cloud. For example, [1] and [2] use octree space decomposition and predictive coding to estimate the child cell configurations utilizing local surface estimation. Other works that use octree for static point cloud compression can be found in [3] and [4].

Dynamic point cloud compression schemes use structural differences between two frames. For example, [5] encodes the structural difference between the octree structure in two point cloud frames. [6] constructs the fixed size block in the voxelized point cloud and computes the motion associated with these blocks in the next frame. More works related to dynamic point cloud compression can be found in [7] and [8]. The color attributes of point cloud on the other hand are compressed separately. Some state-of-art works such as [9] and [10] use orthogonal graph transform to decorrelate color signal and arithmetic coding to encode the coefficients. Besides, one recent work [11] adopts regional-adaptive hierarchical transform (RAHT) method to compress the color attribute. Works such as [12], [13], and [14] use mesh compression techniques by converting point cloud into mesh and apply method to reduce number of vertices and edges using surface approximation. These methods however involve computationally very expensive conversion of point cloud to mesh.

In this work, we present a scalable octree and quadtree coding scheme that can represent the geometry of the same point cloud at different quality achieving different compression level. However, the advantage we get with octree/quadtree scheme is that once point cloud is encoded with octtree/quadtree to its highest representation level, decoder can request any representation from the same encoded data. Here, the highest representation implies point cloud with voxel size of one. This avoids encoding the same point cloud multiple times for different representation level and saves storage space. After encoding, we compress encoded bitstream with...
PAQ compressor. Encoding with octree/quadtree on a point cloud with large number of points is however problematic, as it has to go deeper to encode to its highest representation level where encoding becomes slow and occupancy-bit or octree-representation-bit explodes out exponentially. We therefore, convert encoding problem into subproblem by dividing the point cloud into smaller chunk using binary-tree and encode each chunk separately. The method discussed in this paper is intra-coding and thus can be used to encode static as well as dynamic point cloud.

The rest of the paper is organized as follows. In section 2, a detailed description of encoder is presented. This section is divided into four subsections, each one dedicated to a particular step in encoding process. Section 3 describes the PAQ compression scheme. Section 4 explains the decoding process in the decoder. Section 5 is for discussion of the result obtained. Section 6 concludes the whole paper.

2. EMBEDDED BTQT FOR GEOMETRY CODING

Encoder encodes both geometry and color of a point cloud frame by creating the highest representation possible. In its highest representation, decoder can decode point cloud with most of its voxel with size one. Thus, reconstructed point cloud has almost same number of points compared to the original and the reconstruction is almost lossless. Although, decoder can decode colors at different lower resolutions, in this paper we are concerned with geometry compression and thus we exclude encoding and compression process for colors. BTQT works with two-step approach. The outer layer of coding is a lossy geometry approximation from Binary Tree (BT), which is further refined by Quadtree (QT) if the points lie on approximate 2D planes, and Octree if cannot be approximated by 2D planes.

2.1. Binary-tree Block Construction

Encoding process starts with the block formation in the point cloud frame as shown in Fig. 1. Here, we use binary-tree to create blocks in the form of their leaf-nodes. Binary-tree recursively divides the data by splitting axes into two halves at their median. This binary splitting can be better visualized in a plane as shown in Fig. 2. If a point cloud has \( N \) points then, depth \( L \) binary tree will divide \( 2^L - 1 \) times recursively and at final level we have \( 2^L \) sub-blocks with almost \( N/2^L \) points in each block. The blocks at final level are known as binary-tree leaf-nodes. Point cloud is a 3D data i.e. \( x, y, z \) and division should take in all three dimensions in turn. However, order of division precedes with dimension that has larger variance.

Binary-tree is a data partition scheme and it’s leaf-nodes are adaptive to the surface of the data. Their leaf-nodes are balanced with almost same number of points with difference of 1 and none of them are empty.

Fig. 1. Block formation using binary-tree in the 1st frame of 4 datasets from 8i

Fig. 2. Binary-tree depiction in 2D plane

2.2. Plane Projection with Planner Mode Decision

Local surface in a block with flat characteristics has one of the three dimension almost same value. By projecting the surface in appropriate tangent plane, we can get rid of one whole dimension. By encoding this approximated 2D surface, we can save huge bits compared to encoding the same with octree in
original 3D domain. Before, we need to define some criteria to decide the flatness of the surface. For this we use eigenvalues of the covariance of the surface points in the block.

Let \( X_i = \{x_i, y_i, z_i\} \) are the \( N \) points in a binary-tree leaf-node where \( i = 1, 2, ..., N \). Let \( X = (X_1, X_2, ..., X_N)^T \). Then, we have covariance of leaf-node points as

\[
C = X \times X^T
\]  

Then \( C \) can be equivalently represented by eigenvalue decomposition as

\[
C = \Phi \Lambda \Phi^T
\]  

where \( \Lambda = \text{diag}(\lambda_1, \lambda_2, \lambda_3) \) is a \( 3 \times 3 \) diagonal matrix with eigenvalues \( \lambda_1 > \lambda_2 > \lambda_3 \) as the diagonal elements. \( \Phi \) is a \( 3 \times 3 \) eigenvector matrix. Flatness \( \theta \) is then computed as

\[
\theta = \frac{\lambda_3}{\lambda_1 + \lambda_2 + \lambda_3}
\]  

Here, a threshold is set for \( \theta \). If \( \theta < \text{threshold} \), the surface is considered as flat and qualifies for quadtree encoding. Otherwise, the surface is non-flat and encoded using octree. When local surfaces meet flatness criteria, they are projected onto their tangent planes as follows,

\[
Y = X \times \Phi
\]

Since \( X \) is projected onto the 2D subspace, we consider only the first two columns of \( \Phi \). \( Y \) will finally be a \( N \times 2 \) matrix that represents a plane in 2D domain.

At decoder this projection matrix \( \Phi \) is required for the reconstruction and needs to be signaled. This projection matrix can equivalently be represented by an arrow at the centroid of the local surface plane oriented in the direction of the projection. The smallest eigenvalue \( \lambda_3 \) gives the direction of the projection. We, however, use three angles \( (\alpha, \beta, \gamma) \) for orientation instead of \( \lambda_3 \) and \( b \)-bits for centroid to represent direction of tangent plane projection. These angles \( (\alpha, \beta, \gamma) \) are represented using 8-bits each as they represent numbers between 0 and 180. centroid however, can be represented with fewer bits then \( b \). With each recursive division, the size of binary-tree leaf-boxes shrinks and bits required to represent the centroid get reduced by \( m = \log_2\left(\frac{D_0}{D_j}\right) \), where \( D_0 \) is the diagonal of the whole frame and \( D_j \) is the diagonal of \( j \)th binary-tree block with the flat-surface. The total bit-cost to signal the projection matrix can be estimated as

\[
R_3 = \sum_{j}^{N_F} r_j
\]  

where \( N_F \) is total number of blocks satisfying the flatness criteria. \( r_j \) can be calculated as follows.

\[
r_j = 3 \times (b - m) + (3 \times 8)
\]

2.3. Octree and Quadtree Encoder

Octree is a hierarchical space partition scheme that splits a 3D space into eight sub-blocks or octants by dividing the space into two halves recursively in all three dimensions. The eight blocks generated are called child-cell or child-node. If a child-cell is non-empty, it is again divided further into eight child-cells until some stopping criteria is met. There are multiple ways to stop the recursive division but in our implementation we use depth as stopping criteria. Empty child-cells are not divided. The child-cells obtained at the final layer are called leaf-nodes. All the points lie in these leaf-nodes. Fig. 3 (a) shows the leaf-nodes after octree-decomposition in one of the block of the binary-tree. In this figure, boxes represent the leaf-nodes. The points that lie in a leaf-node are approximated by its centroid and attributes if any are averaged. In Fig. 3 (b), it can be observed that the data points (represented by blue dots) are approximated by the centroid (represented by red-cross) of the box. At highest representation level, leaf-node should contain only one point (which may not always be the case).

Encoding of an octree structure is carried out by storing the occupancy bits generated in all layers of the tree. Every division splits a cell into eight child-cells. Empty child-cells are assigned a 0 bit, and any non-empty a 1 bit. This constitutes a byte called occupancy bits. Formation of these occupancy bits is shown in Fig. 4. When these occupancy bits are scanned in “Breadth-First-Search (BFS)” or left-to-right and top-to-bottom manner of tree, we actually stored the structural information of an octree. Provided this bitstream of occupancy bits and bounding-box of the binary-tree leaf-node, we can construct the same octree structure within the same space of bounding-box. Thus, using the bounding box information of the binary-tree leaf-nodes and their corresponding occupancy bits information representing the octree structure, decoder can reconstruct the point cloud.

Similar procedures go for the quadtree as it is a 2D version of octree. Since quadtree encodes the surface in 2D planes, each recursive division splits the space into four child-cells with four occupancy bit. Any stopping criteria explained for octree can be used to stop the quadtree division.
2.4. Octree and Quadtree Depth Selection

Encoder encodes the local surface point cloud to its highest representation level, i.e., voxel size one. Although each binary-tree leaf-node contains the same number of points, their volumes are not the same. It would not be wise to encode all the binary-tree leaf-node to the same depth as leaf-nodes with smaller volume will be over-represented and leaf-nodes with larger volume will be under-represented. To overcome this issue, we utilize the diagonal of the binary-tree leaf-nodes to compute the depth so that the surface can be optimally represented to its highest representation. The depth is computed by the following equation.

\[ d_i = \log_2 \left( \frac{D_i}{v} \right) \]  

where \( d_i \) is the octree/quadtree depth and \( D_i \) is the diagonal length for the \( i \)th binary-tree leaf-node. \( v \) is the voxel-size, which is 1 for the highest representation.

3. SCALABILITY AND PAQ COMPRESSION

The proposed BTQT coding schemes are scalable, if we have a BFS scanning of the occupancy code. Both quadtree and octree coding schemes are hierarchical and the next level of a higher representation completely depends on the previous lower level representation. In other words, a lower level representation is completely independent of its higher level representation. If there is a request for lower level representation, then occupancy bits scanned in “BFS” manner up to the requested level of quadtree/octree is important but not the occupancy bit higher of it. This variable level scanning of occupancy bits to represent different quality of same point cloud makes the quadtree/octree a scalable coding tool. These occupancy bits up to the requested layer are scanned in all the leaf-nodes and concatenated which are then given to PAQ compressor.

PAQ [15] is a lossless data compressor which uses sophisticated “context-mixing” algorithm that combines predictions from multiple models into one probability. It evolved from PPM (Prediction by Partial Matching) which predicts probability in the next symbol based on the context from previously seen data. PAQ however, uses weighted combination of many separate shallow neural-networks to generate the contexts and predictions from these neural-network models are combined to output single probability. These output predictions are sent to an arithmetic coder to produce more compact bitstream. In this work, we used PAQ8M which can be obtained freely as a windows executable in web. This version of PAQ is designed for image compression. Since, the occupancy byte has values from 1 to 255 similar in image, use of this version of PAQ will provide us the best compression. After the compression of occupancy bitstream with PAQ, we get \( R_4 \) bits which are sent to decoder.

4. LOCAL SURFACE RECONSTRUCTION AT DECODER

Client/decoder requests a version of the point cloud representation to the encoder. Based on the request, encoder creates a representation by retrieving the occupancy bits up to requested level for all binary-tree sub-block and occupancy bitstream is sent to the decoder with other necessary information. The decoder receives the compressed bits and PAQ decompressor then decompresses it to generate the occupancy bits. Using one-bit information which indicates whether binary-tree leaf-node is encoded with octree or quadtree, and the encoding depth, the correct number of occupancy bits are retrieved from the bitstream and sent either to quadtree or octree decoder.

4.1. Octree and Quadtree Decoder

Octree decoder reconstructs an octree structure within the space specified by bounding-box information of corresponding binary-tree block. Quadtree, however, reconstructs a quadtree structure within the boundary of projected tangent plane. After reconstruction leaf-nodes are generated. The voxels are then approximated using the centroid of these leaf-nodes. Octree reconstruction approximates voxels directly in the original subspace. Quadtree, however, generates the voxels in the projected subspace and thus these voxels \( V \) are brought back to the original 3D domain using tangent plane back projection. Using a \( 3 \times 2 \) eigenvector matrix \( \Phi \), it is brought back to original subspace as follow,

\[ X = V \times \Phi^T \]  

Fig. 5 shows the reconstructed octree and quadtree voxels at different depths. Different quality of same point cloud is shown in the figure. As depth increases, the number of reconstructed voxels increases, and they represent more accurate approximation of the original point cloud. On the contrary, as depth decreases, the voxel size increases and the reconstructed voxels represent coarser approximation.

Fig. 4. Occupancy bits assignment in the octree

Fig. 5. Reconstructed voxels at different depths

Authorized licensed use limited to: University of Missouri Libraries. Downloaded on May 17,2021 at 21:15:42 UTC from IEEE Xplore. Restrictions apply.
5. RESULTS AND DISCUSSION

We evaluated our approach on 8i Sequence of category-2 point cloud dataset released by MPEG [16]. This includes 4 different datasets: “8i VFB Loot”, “8i VFB Red and Black”, “8i VFB Soldier”, and “8i VFB Long dress”. All datasets represent 10 seconds long videos captured at 30 fps with a total of 300 frames. These are 10-bit voxelized datasets with the number of points in each dataset varying from 700000 to 1 million per frame. These datasets also have color attributes other than geometry.

Our approach encodes frames in intra-mode and thus all frames are encoded separately. This approach is well suited for both static as well as dynamic point cloud sequences. We tested our method by varying binary-depth $11$ through $13$. We choose three threshold values for flatness: $0.005$, $0.01$, and $0.06$. Increasing threshold to $0.1$ did not change the number of flat surfaces much compared to $0.06$ and we did not go further. The change in number of flat surfaces with the change in threshold is tabulated in Table 1. As the threshold value $\theta$ increased, more local surfaces satisfied the flatness condition. For each configuration of binary-depth and the flatness threshold, the octree-depth is varied along $1$ to $4$ and the quadtree-depth is varied from $2$ to $8$ with step-size of $2$. Since octree generates more child-cells at each split than quadtree, we took depth of quadtree twice the depth of octree to make the reconstruction of point cloud uniform.

Finally, the total bits cost can be calculated as

$$R = (R_1 + R_2 + R_3 + R_4)/N \quad (11)$$

where $N$ is the total number of points in each frame. The PSNR is computed using the equation below,

$$PSNR = 10 \times \log_{10}(3 \times \frac{P^2}{MSE}) \quad (12)$$

where $P = 1023$ is a peak-value specified by MPEG-PCC for the given dataset. $MSE$ was computed using "pc_error.exe" software provided by the MPEG-PCC and was based on point-to-point error. The rate-distortion (R-D) plots are shown in Figure 6. Analyzing the plots, we can see that the curve for higher binary-tree depth has better PSNR but less $bpp$ value compared to lower binary-tree depth for the same $\theta$ value. With higher binary-tree depth, binary-tree leaf blocks become smaller and the surfaces contained in the block become flatter. Thus, more blocks were then encoded with quadtree resulting in a huge bits reduction. As real-flat surfaces were encoded with quadtree, the accuracy of reconstruction also got better. Similarly, with the increase of $\theta$ value, comparably more local surfaces were qualified for quadtree encoding and less with octree. Since these pseudo-flat (they were not flat but considered so) surfaces were encoded with quadtree, this decreased the reconstruction accuracy which is reflected by the result. Surprisingly, the $bpp$ is high. The reason for this increase in $bpp$ value is that to get comparably better accuracy in reconstruction, quadtree has to go very deep eventually accounting very high bit cost. This can be observed in the plot. The $bpp$ value is comparably very high for deeper quadtree/octree but almost similar for the shallower one.

6. CONCLUSION

In this paper, we addressed point cloud compression problem with scalable coding solution. The proposed BTQT approach encoded point cloud with fix-rate at highest possible representation. Decoder on the other hand reconstructed the point cloud data at variable qualities. The scalable capability
Fig. 6. Typical R-D curves for the four point cloud sequences of the encoder retrieved the correct version of the variable-rate encoded data and provided it to PAQ for further compression. The quality of reconstruction was varied to finer level by forming sub-block with binary-tree and encoding each block separately with octree/quadtree. The BTQT was more efficient at higher depth of binary-depth where many true flat surfaces were encoded by quadtree spending less bits but achieving better accuracy. Encoding with octree also provided better accuracy but required larger bit cost.

7. REFERENCES


