

Hybrid Coding for Animated Polygonal Meshes: Combining Delta and Octree

Jinghua Zhang

Department of Computer Science and Engineering
Michigan State University
East Lansing, Michigan, 48824, USA

Charles B. Owen

Department of Computer Science and Engineering
Michigan State University
East Lansing, Michigan, 48824, USA

Abstract

A new hybrid coding method for compressing 3D animated sequences of geometric data is presented. It is based on the most common representation of the geometric data: polygonal meshes for each discrete frame in the animation sequence. The method utilizes a delta coding and an octree-based method. It can achieve higher compression ratios than the octree-only method or the delta-only coding method. In this hybrid method, both the octree approach and the delta coding approach are applied to each single frame in the animation sequence in parallel. The approach that generates the smaller encoded file size is chosen to encode the current frame. Given the same quality requirement, the hybrid coding method is much better than either single approach in terms of compression ratio.

Keywords: compression, animated polygonal meshes, octree, motion vectors, and quantization.

1. Introduction

3D graphics are rapidly achieving mainstream success, both in motion pictures and computer gaming. The confluence of fast real-time hardware and advanced modeling and rendering capabilities is opening the door to a world where the animated films of the future will be rendered and viewed on the desktop with varying viewpoints, high resolutions, and with output devices ranging from cell-phones to theatre-sized projection. But, as these models grow more complex, more detailed, and longer format, it becomes increasingly difficult to efficiently store and transmit the huge 3D models that form the basic input for these future rendering systems. As a simple example, an efficient and powerful mechanism for animated geometry compression allows for the delivery of an animated motion picture such as *Shrek* as a 3D model in real time on demand or as a multicast. A user can view the film from arbitrary viewpoints, in stereo, or in an immersive Virtual Reality environment. To achieve this data delivery in real time, we need a method that can achieve high compression ratio

while maintaining good quality. This paper presents a hybrid coding method for compressing animated polygonal meshes: combining the octree-based method proposed in our previous paper [1] and a delta coding method. Given the same quality requirement, the hybrid coding outperforms either single approach in terms of compression ratio.

Geometry compression has become a very active field since Deering [2] proposed the triangle strip based geometry compression technique. The state of the art in geometry compression in the last ten years can be organized into two categories: vertex data compression techniques [2-6] and connectivity compression techniques [2-5, 7-9]. Boosen [10] gives a comprehensive survey of the notable static geometry compression techniques.

This paper focuses on compressing animated polygonal meshes with fixed connectivity. The straightforward way to compress 3D animated meshes is to apply vertex data compression individually to each frame in the animation sequence, but this approach does not take advantage of the temporal coherence present in the animation sequence. Animation compression techniques need to exploit not only the spatial coherence, but also the temporal coherence in the sequence.

Recent notable efforts in animated geometry compression are listed as follows: Lengyel [11] presented a prediction compression method that splits vertices into sets and uses transformation matching techniques to approximate the vertex paths. His method is effective only when the animation is well represented by the supported transformations. Alexa and Müller [12] proposed an interpolation predictor to represent animations by principal components. Principal components analysis (PCA) makes their approach expensive. Their approach requires significant memory usage and processing time. It cannot achieve high compression ratios if the number of frames is significantly smaller than the number of vertices. The quality of individual frames in the animation sequence is not easy to control but this approach is good in terms of smoothness of the shape. Ibarria and Rossignac [13] proposed a time-space predictor for all the vertices and key frames. The geometry of a vertex is predicted from the geometry of its

neighbors in the same frame and the geometry of the corresponding vertices in the previous frame. Briceño et al.[14] proposed an algorithm to generate a geometry video from 3D animated meshes. This approach is based on a geometry image representation proposed by Gu et al. [15]. They generate a geometry image (a picture) for each frame in the sequence. The resulting geometry image sequence is called a geometry video. They then apply the conventional video compression techniques to the geometry video.

Our work seeks a compact representation of a 3D animation that takes advantage of the large data coherence in space and time. Compared to the previous published approaches, our approach does not need to split the dataset to fit different predictors as in Lengyel’s method [11]. Unlike PCA approach, our approach requires much smaller memory usage, runs much faster and compresses the animation sequences equally well no matter how many frames are in the sequence. We can manage the quality of each frame by setting the same threshold for each frame in the sequence. The hybrid approach can achieve high compression ratio and quality with no restriction on the data set. A simple and low-cost encoding process and a fast decoding process make our approach quite suitable for real time applications.

The remainder of this paper is organized as follows: Section 2 shows the data representation for the 3D animation. Section 3 reviews the octree-based approach. Section 4 presents the delta coding method. Section 5 presents the hybrid method that combines the octree-based approach and delta coding approach. Section 6 shows the results of the hybrid method in detail and compares it with the octree-only approach and delta-only approach. Section 7 presents some conclusions.

2. Data representation

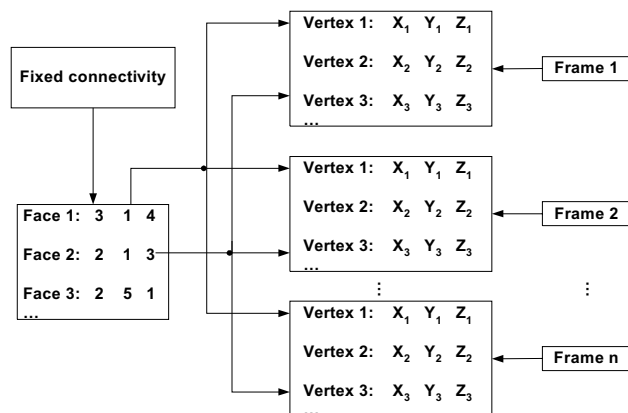


Figure 1. 3D animation representation

This paper assumes a common representation of the geometric data: triangle meshes for each discrete frame

in the animation sequence. It also assumes the connectivity is fixed for the lifetime of the animation. Any polygonal mesh can be converted into a triangle mesh. A triangle mesh is represented by its vertex data and connectivity information. The vertex data consist of the coordinates of all the vertices and, optionally, vertex normal vectors and texture coordinates. Connectivity information is simply a triangle list and each triangle contains indices referring back to the vertices. Figure 1 shows the data representation for the 3D animation.

3. Octree-based motion representation

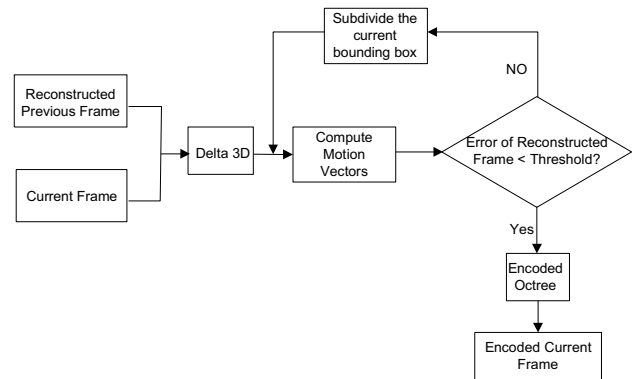


Figure 2. Encode a frame using the octree approach

In a previous paper, we [1] proposed an octree-based animated geometry compression method that assumes the animation sequences are represented as shown in Figure 1. The logical flow of encoding a frame using the octree-based approach is shown in Figure 2.

To encode the current frame, the reconstructed version of the previous frame needs to be available in the encoder. A set of motion vectors is generated to represent the differential motion between the previously reconstructed frame and the current frame, and these motion vectors are stored in an octree. Octree is then encoded for further data reduction. The first frame in the animation sequence is encoded differently using intra-frame encoding method. An encoded animation sequence consists of an intra-coded first frame followed by an encoded octree for each of subsequent frames. The size of encoded octree is significantly smaller than the corresponding frame in the animation sequence. The vertex positions for each frame can be reconstructed in the decoder side by the corresponding octree and the previous reconstructed frame in the sequence.

3.1 Octree construction

Jackins and Tanimoto [16] demonstrated a way to represent a 3D object using an octree structure. Ahuja and Nash [17] improved the idea presented by Jackins and

Tanimoto, and proposed a method to represent moving 3D objects using an octree structure. In the octree-based approach [1], an octree is used to present the differential motion between frames in space.

A threshold requirement is set before constructing octrees for the animation sequence. The threshold is used to manage the reconstructed error. It is set to be the maximum allowed Euclidean distance between the corresponding vertices in the original frame and the reconstructed frame.

To build an octree for the current frame, a minimum cubic box (cell) is the starting point to construct an octree. It includes all the vertices of the 3D objects within the current frame. A motion vector is associated with each corner of the cell. These eight motion vectors approximate the motion of the enclosed vertices using interpolation of the motion vectors over the space. Motion vectors are obtained by using a Least Square Error estimation method to estimate the motion of the enclosed vertices. If the error of reconstructed motion is smaller than the specified threshold, no splitting process is needed. Otherwise, the current cell needs to be further subdivided into eight smaller cells and each has eight new motion vectors. The splitting process continues until the motion of the enclosed vertices can be approximated within the specified threshold.

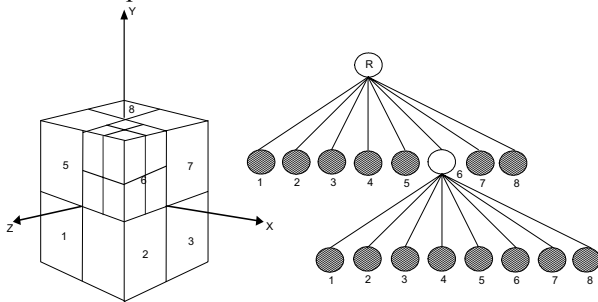


Figure 3. The splitting process and the octree

Each cell corresponds to a tree node in an octree and its eight motion vectors are stored in the corresponding tree node. When the current bounding box needs to be subdivided into eight smaller bounding boxes, the corresponding tree node will generate eight children and each of them has eight new motion vectors. This process is illustrated in Figure 3.

3.2 Motion vector computation

Motion vectors are obtained by using a Least Square Error method to estimate the motion of enclosed vertices. In the octree approach, tri-linear interpolation of corner representational motion vectors over space is used for the estimation of the motion of the enclosed vertices. The vertex motion Δv for the target point is computed using tri-linear interpolation of the eight motion vectors of the

cell, as shown in Figure 4 and illustrated in Equation 3.1. Where w_i is the weight of each motion vector.

$$\Delta v = \sum_{i=1}^8 w_i m_i \quad (3.1)$$

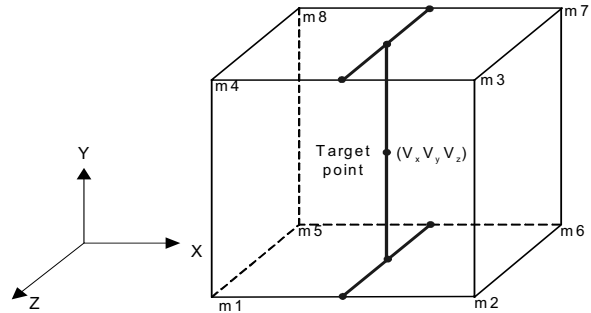


Figure 4. Tri-linear interpolation

The motion vectors for a given cell are computed using least square estimation of x in the equation $Ax=b$, where x is a 24 element vector consisting of a sequential listing of the $\langle x, y, z \rangle$ components of eight motion vectors. Matrix A is a $3n$ ($3 \times$ number of vertices) by 24 matrix, which is composed of the weights of eight motion vectors. Vector b is computed by the delta 3D of the previous frame and the current frame. For detail implementation, please refer to the paper [1].

4. Delta coding method

During the course of the research, delta coding was explored as an alternative to the octree-based approach. The delta coding method does not achieve the consistent level of the performance demonstrated by the octree approach. However, in limited cases the delta approach does outperform. Hence, it is presented here in isolation, and then incorporated into a hybrid compression approach in the next section.

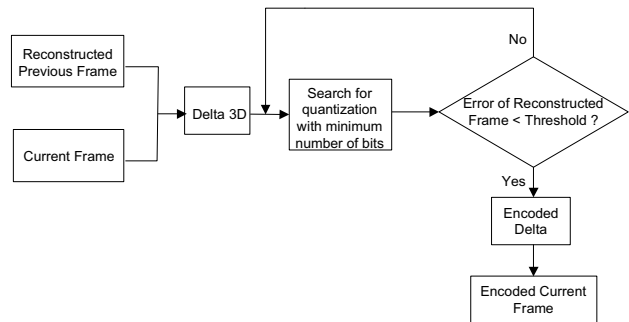


Figure 5. Encode a frame using the delta approach

Figure 5 shows the encoding process for one frame. To encode the current frame, the previous reconstructed frame needs to be available in the encoder side. Therefore

the first frame is encoded differently from the other frames in the sequence by using an intra-frame coding method. The first step in the delta encoding process is to generate delta 3D values between two consecutive frames, namely the reconstructed previous frame and the current original frame. Delta 3D data presents the differential motion between two frames. After the first step, delta approach seeks the best quantization level with the minimum number of bits for the current delta 3D that satisfies a predefined threshold. After the number of bits for the quantization is determined, the *linear quantization* over the range of each possible vertex is applied to the delta 3D as illustrated in Equation 4.1. The quantized values are encoded using adaptive arithmetic coding to achieve further data reduction. To this step, the current frame is completely delta encoded. In the local decoding step, a *linear inverse quantization* is applied to the quantized samples as shown in Equation 4.2.

$$\bar{x} = \left\lfloor 2^q \left(\frac{x - x_{\min}}{x_{\max} - x_{\min}} \right) \right\rfloor \quad (4.1)$$

$$x = \frac{(x_{\max} - x_{\min}) * \bar{x}}{2^q} + x_{\min} \quad (4.2)$$

5. Hybrid coding

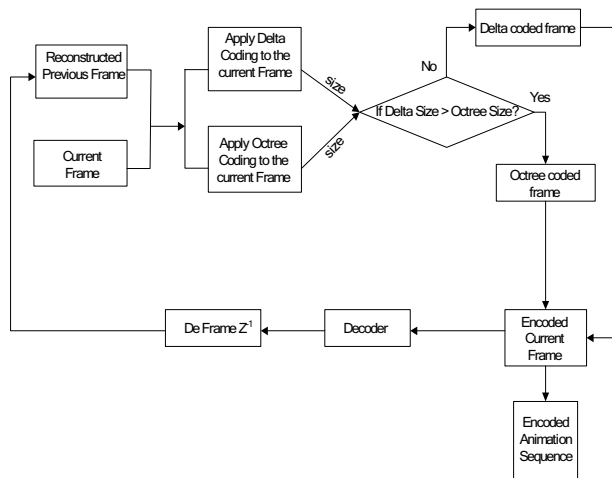


Figure 6. The hybrid encoding process

In this section, we present a hybrid coding approach to compress animated polygonal meshes by combining the octree-based approach and delta coding approach. The motivation of this approach comes from the observation that the encoded octree size is sometimes larger than the delta encoded size. To overcome the shortcomings of the octree approach, we propose the hybrid approach. The

logical flow of the encoding process using the hybrid approach is shown in Figure 6.

This approach uses a threshold to control the quality of the output. The threshold is used to control the subdividing process of the octree coding and the number of bits used in the quantization step of the delta coding. For the octree encoding process, if the error (Euclidean distance) of any reconstructed vertex is larger than the threshold, it will subdivide the current bounding box and repeat the motion vector computation step. For the delta encoding process, if the error of any reconstructed vertex is larger than the threshold, it will continue search for the best quantization level that drops the reconstruction error below the threshold. Therefore, any reconstructed vertex in any frame in the reconstructed animation sequence satisfies the criteria that the error between this vertex and the corresponding original vertex is smaller than the threshold as shown in Equation 5.1. The threshold in the hybrid coding method is set to be the maximum allowed distance between the reconstructed vertex and its corresponding original vertex. It is represented by the percentage of the edge length of the initial cubic bounding box of the object as illustrated in the Equation 5.2.

$$\forall v \in \text{current cell} \quad \sqrt{(v_x - v'_x)^2 + (v_y - v'_y)^2 + (v_z - v'_z)^2} < \text{threshold} \quad (5.1)$$

v and v' represent the original vertex and the corresponding reconstructed vertex.

$$\text{Threshold} = \% \text{Length} \quad (5.2)$$

Length is the edge length of the initial cubic bounding box. The chosen percentage influences the quality; the quality decreases as the percentage increases.

To encode each frame except the first frame in the sequence, both the octree coding method and the delta coding method are applied in parallel. If the size of encoded delta is smaller than the encoded octree, the encoded octree is replaced with the encoded delta. A single bit in the output data stream is used to indicate if it is an octree coding or delta coding.

6. Results

To compare the hybrid coding method with the delta-only coding and the octree-only coding method, we tested the performance of the hybrid coding method on three animation datasets: “Chef”, “Chicken Crossing” and “Dance”. The Chef data was generated from a 3D Studio Max animation. It was used as an initial test data set. “Chicken Crossing” was created by Andrew Glassner et al. The “Dance” sequence was created by the MIT CSAIL Graphics Lab. Table 1 shows the properties of the “Chef”, “Chicken Crossing” and “Dance” animation sequences.

Table 1. Test data sets information

Data sets	Chef	Chicken	Dance
Num vertices	4241	3030	7061
Num triangles	8162	5664	14118
Num frames	75	400	201
Size (bytes)	3,816,900	14,544,000	17,031,132

Because the hybrid, octree-only and delta-only techniques use the same threshold settings, the quality of the reconstructed animations are similar. Given the same quality requirements (threshold), the compression ratios obtained from these three methods are compared. Figure 7 shows the compression ratio comparison among the hybrid, octree and delta coding for the Chef sequence. Similarly Figure 8 and Figure 9 show the comparison results for the Chicken animation and Dance animation.

Figure 10 shows the reconstructed frame 20 in the Chef animation given different threshold. Figure 11 and Figure 12 show a reconstructed frame in the Chicken and Dance animation sequence.

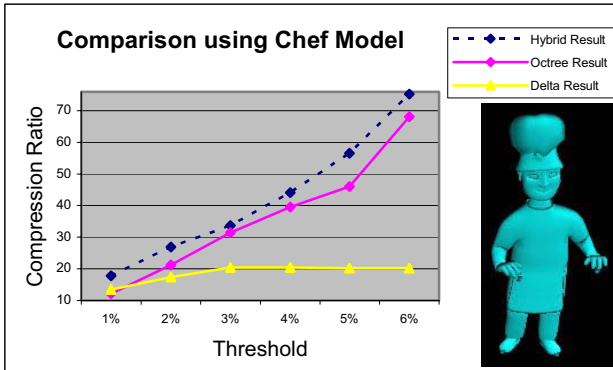


Figure 7. Comparison results for the Chef model

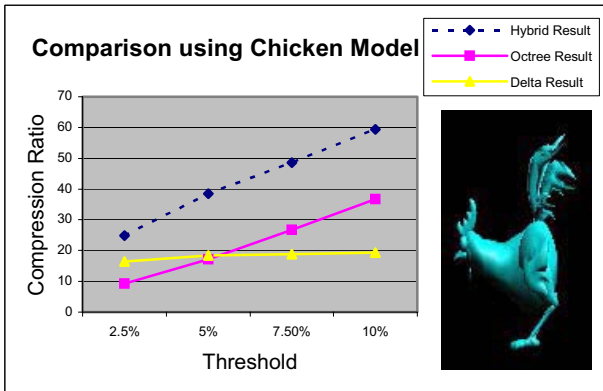


Figure 8. Comparison results for the Chicken model

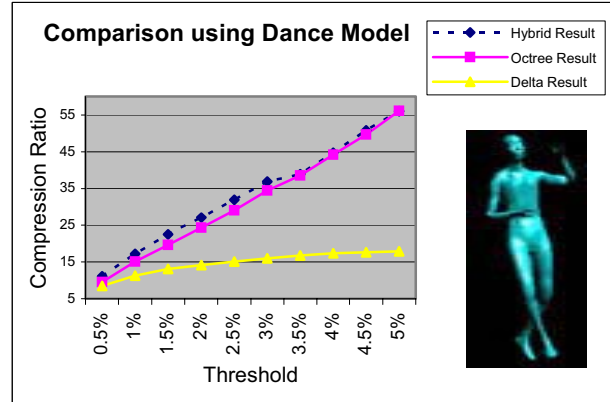


Figure 9. Comparison results for the Dance model

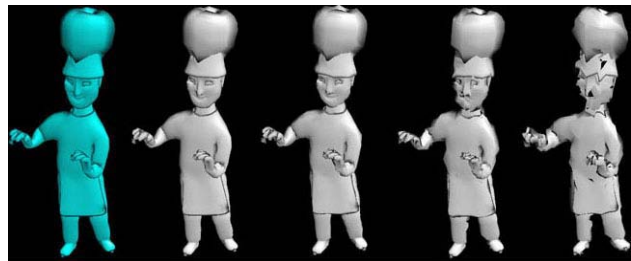


Figure 10. The image in the first column is the original frame 20. The reconstructed frames by using threshold=1%, 2%, 3%, 4% are shown in the column 2, 3, 4, 5 respectively.



Figure 11. The image in the first column is the original frame 100. The reconstructed frame by using threshold=2.5%, 5%, 7.5%, 10% are shown in the column 2, 3, 4, 5 respectively.

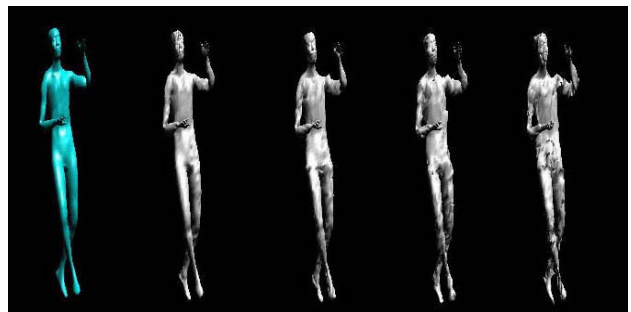


Figure 12. The image in the first column is the original frame 30. The reconstructed frames by using threshold=1.5%, 2.5%, 3.5%, 4.5% are shown in the column 2, 3, 4, 5 respectively.

The hybrid method generates a mixed sequence of the octree encoded and delta encoded frames. Table 2 shows the percentage of the octree encoded and delta encoded frames for different models and different thresholds. We found that the delta encoding contributes significantly to the hybrid method. It takes up to 40% of the encoded frames. However, as the threshold increases, the percentage of delta coded frames in the encoded sequence decreases. This is because the performance of the octree approach is much better than the delta coding approach as the threshold increases.

Table 2. Percentage of the octree encoded and delta encoded frames in the hybrid method

	Threshold	Octree	Delta
Chef	1%	59.5%	40.5%
	2%	86.5%	13.5%
	3%	92%	8%
	4%	93.3%	6.7%
Chicken	2.5%	68%	32%
	5%	79%	21%
	7.5%	87.2%	12.8%
	10%	89.2%	10.8%
Dance	1.5%	90%	10%
	2.5%	95%	5%
	3.5%	96.5%	3.5%
	4.5%	97%	3%

7. Conclusion

The hybrid approach presented in this paper is an efficient way to compress animated sequences with fixed connectivity. It is easy to implement and can achieve high compression ratios. Given the same quality requirements, the hybrid approach outperforms both the octree-based and delta coding approaches in terms of compression ratio. This hybrid approach provides a better technique to predict the differential motion of any consecutive frames in the animation sequence.

Acknowledgements

We want to thank the Media Interface and Network Design Lab (MIND lab) for some support of this work. We want to thank Andrew Glassner for providing access to the Chicken data. The chicken character was created by Andrew Glassner, Tom McClure, Scott Benza, and Mark Van Langeveld. This short sequence of connectivity and vertex position data is distributed solely for the purpose of comparison of geometry compression techniques. We also want to thank MIT CSAIL Graphics Lab for the dance sequence.

References

- [1] J. Zhang and C. B. Owen, "Octree-based Animated Geometry Compression," *Proceedings of Data Compression Conference (DCC'04)*, pp. 508-517, Snow Bird, UT, 2004.
- [2] M. Deering, "Geometry Compression," *Proceedings of ACM SIGGRAPH'95*, pp. 13-20, 1995.
- [3] G. Taubin and J. Rossignac, "Geometric compression through topological surgery," *ACM Transactions on Graphics*, vol. 17, no. 2, pp. 84-115, 1998.
- [4] C. Touma and C. Gotsman, "Triangle Mesh Compression," *Proceedings of 24th Conference on Graphics Interface (GI-98)*, pp. 26-34, San Francisco, 1998.
- [5] R. Pajarola and J. Rossignac, "Compressed progressive meshes," *IEEE Transactions on Visualization and Computer Graphics*, vol. 6, no. 1, 2000.
- [6] Z. Karni and C. Gotsman, "Spectral compression of mesh geometry," *Proceedings of ACM SIGGRAPH'00*, pp. 279-286, 2000.
- [7] J. Rossignac, "Edgebreaker: Connectivity Compression for Triangle Meshes," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, no. 1, pp. 47-61, 1998.
- [8] S. Gumhold and W. Strasser, "Real time compression of triangle mesh connectivity," *Proceedings of ACM SIGGRAPH'98*, pp. 133-140, 1998.
- [9] D. King, J. Rossignac, and A. Szymczak, "Connectivity compression irregular quadrilateral meshes," Georgia Tech, Tech. Rep. TR-99-36, GVU, 1999.
- [10] F. Bossen, "On The Art Of Compressing Three-Dimensional Polygonal Meshes And Their Associated Properties," Ph.D. Thesis, cole Polytechnique Fdrale de Lausanne (EPFL), 1999.
- [11] J. E. Lengyel, "Compression of Time-Dependent Geometry," *Proceedings of ACM Symposium on Interactive 3D Graphics*, pp. 89 -95, New York, ACM Press, 1999.
- [12] M. Alexa and W. Müller, "Representing Animations by Principal Components," *Computer Graphics Forum*, vol. 19, no. 3, pp. 411-418, 2000.
- [13] L. Ibarria and J. Rossignac, "Dynapack: Space-Time Compression of the 3D animations of triangle meshes with fixed connectivity," *Proceedings of Eurographics/SIGGRAPH Symposium on Computer Animation*, 2003.
- [14] H. M. Briceño, P. V. Sander, L. McMillan, S. Gortler, and H. Hoppe, "Geometry Videos: A new representation for 3D Animations," *Proceedings of Eurographics/SIGGRAPH Symposium on Computer Animation (SCA03)*, San Diego, California, 2003.
- [15] X. Gu, S. Gortler, and H. Hoppe, "Geometry images," *Proceedings of ACM SIGGRAPH '02*, pp. 355--361, 2002.
- [16] C. L. Jackins and S. L. Tanimoto, "Oct-tree and their use in representing three-dimensional objects," *Proceedings of Computer Graphics and Image Processing*, vol. 14, pp. 249-270, 1980.
- [17] N. Ahuja and C. Nash, "Octree Representations of moving objects," *Proceedings of Computer Vision, Graphics and Image Processing*, vol. 26, pp. 207-216, 1984.