3D Point Cloud Data and Triangle Face Compression by a Novel Geometry Minimization Algorithm and Comparison with other 3D Formats

*M. M. Siddeq[1], †M. A. Rodrigues[2]

[1,2] GMPR-Geometric Modeling and Pattern Recognition Research Group,
Sheffield Hallam University, Sheffield, UK

*Presenting author: mamadmmx76@gmail.com
†Corresponding author: M.Rodrigues@shu.ac.uk

Abstract
Polygonal meshes remain the primary representation for visualization of 3D data in a wide range of industries including manufacturing, architecture, geographic information systems, medical imaging, robotics, entertainment, and military applications. Because of its widespread use, it is desirable to compress polygonal meshes stored in file servers and exchanged over computer networks to reduce storage and transmission time requirements. 3D files encoded by OBJ format are commonly used to share models due to its clear simple design. Normally each OBJ file contains a large amount of data (e.g. vertices and triangulated faces) describing the mesh surface. In this research we introduce a novel algorithm to compress vertices and triangle faces called Geometry Minimization Algorithm (GM-Algorithm). First, each vertex consists of (*x, y, z*) coordinates that are encoded into a single value by the GM-Algorithm. Second, triangle faces are encoded by computing the differences between two adjacent vertex locations, and then coded by the GM-Algorithm followed by arithmetic coding. We tested the method on large data sets achieving high compression ratios over 90% while keeping the same number of vertices and triangle faces as the original mesh. The decompression step is based on a Parallel Fast Matching Search Algorithm (Parallel-FMS) to recover the structure of the 3D mesh. A comparative analysis of compression ratios is provided with a number of commonly used 3D file formats such as MATLAB, VRML, OpenCTM and STL showing the advantages and effectiveness of our approach.

Keywords: 3D Object Compression and Reconstruction, Data Compression, GM-Algorithm, Parallel-FMS Algorithm

1. Introduction

Polygonal meshes are the primary representation used in the manufacturing, architectural, and entertainment industries for the visualization of 3D data, and they are central to Internet and broadcast multimedia standards such as MPEG-4 [1,2,4] and VRML [3]. In these standards, a polygonal mesh is defined by the position of its vertices (geometry); by the association between each face and its sustaining vertices (connectivity); and optional colour, normal and texture coordinates (properties). Deering [5] introduced the first geometry compression scheme to compress the bit stream sent by a CPU to a graphics adapter, generalizing the popular triangle strips and fans. Motivated by Deering’s work, but optimized for transmission over the internet instead, Taubin and Rossignac introduced the Topological Surgery (TS) method [6], the first connectivity preserving single-resolution manifold triangular mesh compression scheme. TS was later extended to handle arbitrary manifold polygonal meshes with attached properties, and proposed as a compressed file format to encode VRML files [9]. With a more efficient encoding, Topological Surgery is now part of the MPEG-4 standard.

Several closely related methods were subsequently developed by Touma and Gotsman [12], Gumhold and Strasser [7], Li and Kuo [8] and Rossignac [10]. The methods proposed by Gumhold and Strasser, and by Rossignac only capable of encoding connectivity. The method proposed by Touma and Gotsman, predicts geometry and properties better, and the method proposed by Li and Kuo improves on the entropy encoding of prediction errors. More recently, Bajaj *et al*. [11] proposed yet another method to encode single-resolution triangular meshes. It is based on a decomposition of the mesh into rings of triangles originally used by Taubin and Rossignac in their compression algorithm, but with a different and more complex encoding. All of these schemes require $O(n)$ total bits of data to represent a single-resolution mesh in compressed form.

While single resolution schemes can be used to reduce transmission bandwidth, it is frequently desirable to send the mesh in progressive fashion. A progressive scheme sends a compressed version of the lowest resolution level of a level-of-detail (LOD) hierarchy, followed by a sequence of additional refinement operations. In this manner, successively finer levels of detail may be displayed while even more detailed levels are still arriving. To prevent visual artefacts, sometimes referred to as popping, it is also desirable to be able to transition smoothly from one level of the LOD hierarchy to the next by interpolating the positions of corresponding vertices in consecutive levels of detail as a function of time [11].

The Progressive Mesh (PM) scheme introduced by Hoppe [13] was the first method to address the progressive transmission of multi-resolution manifold triangular mesh data. PM is an *adaptive refinement* scheme where new faces are inserted in between existing faces. Every triangular mesh can be represented as a base mesh followed by a sequence of *vertex split* refinements. Each vertex split is specified for the current level of detail by identifying two edges and a shared vertex. The mesh is refined by cutting it through the pair of edges, splitting the common vertex into two vertices and creating a quadrilateral hole, which is filled with two triangles sharing the edge connecting the two new vertices. The PM scheme is not an efficient compression scheme. Since the refinement operations perform very small and localized changes, the scheme requires $O(V\ log2(V))$ bits to double the size of a mesh with V vertices. Later on Hoppe proposed a more efficient implementation based on changing the order of transmission of the edge split operations [14].

In progressive representations discussed above, multi-resolution polygonal models are represented in compressed form. However, as compression schemes, these are not as efficient as the single-resolution schemes described earlier. Taubin *et al*. [15] recently introduced a method to compress any multi-resolution mesh produced by a vertex clustering algorithm with compression ratios comparable to the best single resolution schemes. In this scheme, the connectivity of the LOD hierarchy is transmitted from high resolution to low resolution, followed by the geometry and properties from low resolution to high resolution. The main contribution of this scheme is a method to compress the *clustering* mappings which relate consecutive levels of detail, from high to low resolution. The method achieves high compression ratios but is not progressive.

The MPEG-4 3D Mesh Coding scheme is based on the Topological Surgery and Progressive Forest Split schemes. But it incorporates improvements to connectivity encoding for progressive transmission proposed by Bossen [16], non-manifold encoding proposed by Guéziec *et al*. [17], error resiliency proposed by Jang *et al*. [18], parallelogram prediction proposed by Touma and Gotsman [15], and error encoding proposed by Li and Kuo [8]. It allows the encoding of any polygonal mesh (including non-manifolds) with no loss of connectivity information and no repetition of geometry and property data associated to singular vertices as a progressive single-resolution bit stream, and any manifold polygonal mesh in hierarchical multi-resolution mode. Extensive experimentation performed during the course of the MPEG-4 process has shown that the resulting methods are state-of the-art.

Siddeq and Rodrigues proposed a new way to compress vertices by using a Geometry Minimization Algorithm (paper submitted to a journal and under review – for more information please contact the authors). In this paper we introduce a new concept for geometry and mesh connectivity compression. The proposed method encodes both the point cloud data representing the integer vertices (geometry) and the triangulated faces (connectivity). Thereafter, the encoded output is subjected to arithmetic coding. We demonstrate the approach by performing a comparative analysis with a number of 3D data file formats focusing on compression ratios.

This remainder of this paper is organized as follows: Section 2 introduces geometry coding and describes the proposed Geometry Minimization (GM-Algorithm) applied to the vertices. Section 3 describes mesh connectivity lossless coding by the GM-Algorithm, while section 4 describes the Parallel Fast Matching Search algorithm (PFMS), used to reconstruct vertices and triangulated faces. Section 5 describes experimental results with a comparative analysis followed by conclusions in Section 6.

2. Geometry Compression

Geometry compression combines quantization and statistical coding. Quantization truncates the vertex coordinates to a desired accuracy and maps them into integers that can be represented with a limited number of bits. The quantization parameter, α, is a scale parameter that normally moves the decimal place of each vertex to the right. A tight (min-max) axis aligned bounding box around each object is computed. The minima and maxima of the (x, y, z) coordinates, which define the box, together with the parameter α are encoded and transmitted with the compressed representation of each object. In this way, the 3D structure can be reconstructed in the same units and scale as the original.

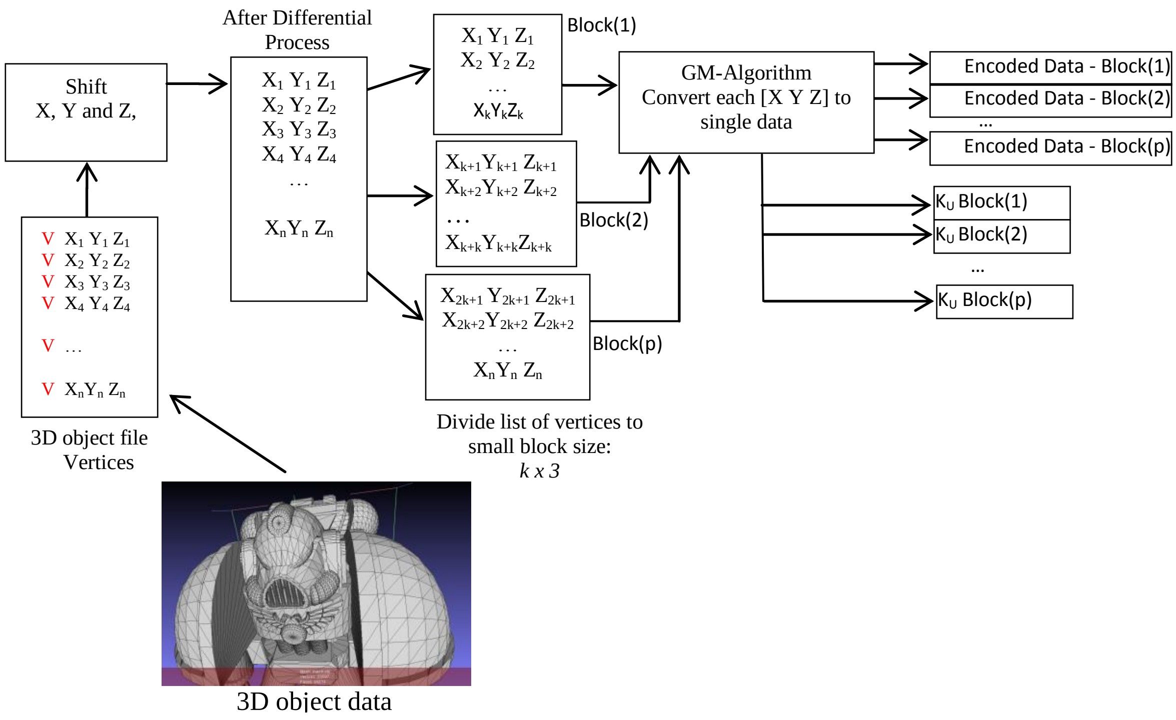
The quantization by α transforms each (x, y, z) coordinates into integers ranging from 0 to $2B–1$, where B is the maximum number of bits needed to represent the quantized coordinates. Normally, 12bit integers are sufficient to ensure geometric fidelity for most applications and most models. Thus, such lossy quantization step reduces the storage cost of geometry from 96-bits to less than 36-bits. The quantization of vertices (x, y, z) is defined as:

$$V_{x,y,z} = floor(V_{x,y,z}\ \alpha) \tag{1}$$

Where $2 \le \alpha \le 10{,}000$. In addition to reducing the storage cost of geometry, we reduced the number of bits for each vertex to less than 16-bit by calculating the differences between two adjacent coordinates for increased redundancy data and thus, more susceptible to compression. The differential process defined in Eq. (2) below is applied to axes X, Y and Z independently [19].

$$D(i) = D(i) - D(i+1) \tag{2}$$

Where *i=1, 2, 3... m-1* and *m* is the size of the list of vertices.



Shift X, Y and Z,
After Differential Process
$X_1\ Y_1\ Z_1$ $X_2\ Y_2\ Z_2$ $X_3\ Y_3\ Z_3$ $X_4\ Y_4\ Z_4$ … $X_n Y_n\ Z_n$
V $X_1\ Y_1\ Z_1$ V $X_2\ Y_2\ Z_2$ V $X_3\ Y_3\ Z_3$ V $X_4\ Y_4\ Z_4$ V … V $X_n Y_n\ Z_n$
3D object file Vertices
$X_1\ Y_1\ Z_1$ $X_2\ Y_2\ Z_2$ … $X_k Y_k Z_k$
Block(1)
$X_{k+1} Y_{k+1}\ Z_{k+1}$ $X_{k+2} Y_{k+2}\ Z_{k+2}$ … $X_{k+k} Y_{k+k} Z_{k+k}$
Block(2)
$X_{2k+1}\ Y_{2k+1}\ Z_{2k+1}$ $X_{2k+2} Y_{2k+2}\ Z_{2k+2}$ … $X_n Y_n\ Z_n$
Block(p)
Divide list of vertices to small block size: k x 3
GM-Algorithm Convert each [X Y Z] to single data
Encoded Data - Block(1)
Encoded Data - Block(2)
…
Encoded Data - Block(p)
K_U Block(1)
K_U Block(2)
…
K_U Block(p)
3D object data

Figure 1. The GM-Algorithm applied to each block of vertices

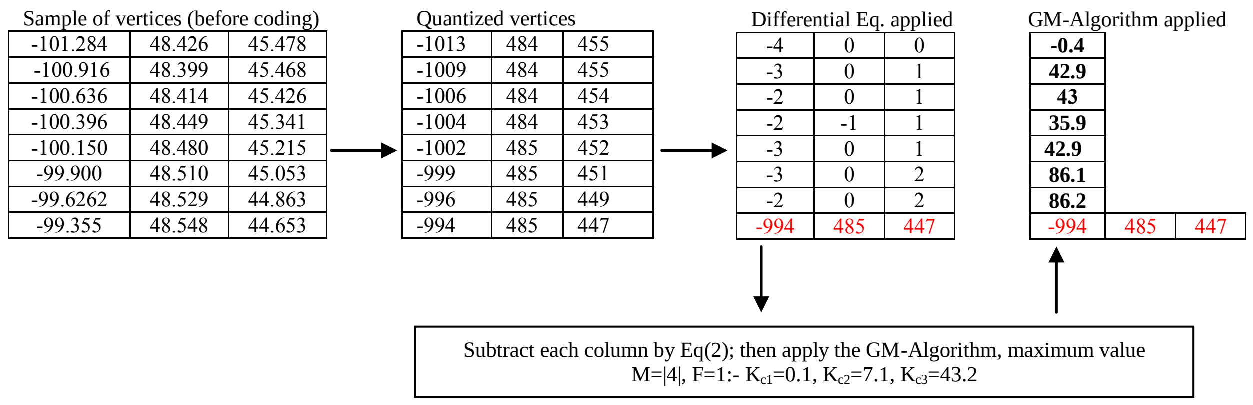
Once the differential process is applied to the vertices, the list of vertices is divided into blocks, and the GM-Algorithm is applied to each block of vertices (i.e. the vertex matrix from 3D object file is divided into *k* non-overlapping blocks) as illustrated in Figure 1. The main reason for placing vertices into separate blocks is to speed up the compression and decompression steps. Each *k* block is reduced to an encoded data array. The GM-Algorithm is defined as taking three key values and multiplying these by three geometry coordinates (*x, y, z*) from a block of vertices which are then summed over to a single integer value. A 3-value compression key K_C is generated from vertex data as follows:

$$M = \max(V_X, V_Y, V_Z) + \frac{\max(V_X, V_Y, V_Z)}{2} \quad \textit{\% Define M as a function of maximum}$$
$$K_{C1} = random(0,1) \quad \textit{\% First weight} \leq \textit{1 defined by random between 0 and 1}$$
$$K_{C2} = (K_{C1} + M) + F \quad \textit{\% F is an integer factor F=1,2,3,…}$$
$$K_{C3} = (M * K_{C1} + M * K_{C2}) * F$$

Where *F* is a positive factor multiplier, each vertex is then encoded as:

$$V(i) = V_x(i)K_{C1} + V_y(i)K_{C2} + V_z(i)K_{C3} \tag{3}$$

Figure 2(a) illustrates the GM-Algorithm by applying Equation (3) to a sample of vertices. After this operation, the likelihood for each block of vertices is selected from which a *Ku (unique Key)* is generated to be used in the decompression stage as illustrated in Figure 2(b) with a numerical example.



Sample of vertices (before coding)

-101.284	48.426	45.478
-100.916	48.399	45.468
-100.636	48.414	45.426
-100.396	48.449	45.341
-100.150	48.480	45.215
-99.900	48.510	45.053
-99.6262	48.529	44.863
-99.355	48.548	44.653

Quantized vertices

-1013	484	455
-1009	484	455
-1006	484	454
-1004	484	453
-1002	485	452
-999	485	451
-996	485	449
-994	485	447

Differential Eq. applied

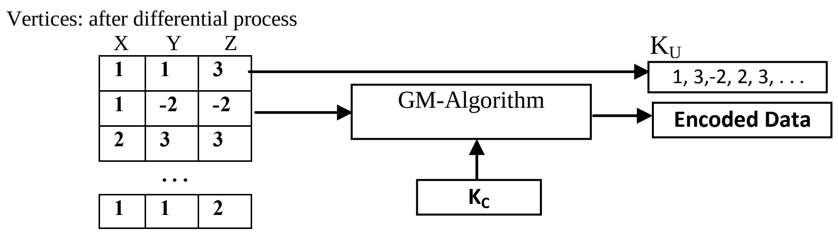
-4	0	0
-3	0	1
-2	0	1
-2	-1	1
-3	0	1
-3	0	2
-2	0	2
-994	485	447

GM-Algorithm applied

-0.4		
42.9		
43		
35.9		
42.9		
86.1		
86.2		
-994	485	447

Subtract each column by Eq(2); then apply the GM-Algorithm, maximum value M=|4|, F=1:- $K_{c1}=0.1$, $K_{c2}=7.1$, $K_{c3}=43.2$

(a) Floating point vertices



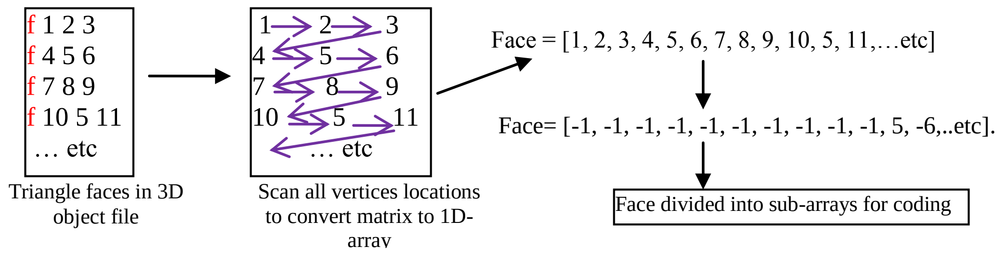
(b) Unique Key

Figure 2: (a): Sample of vertices compressed by GM-Algorithm, (b) The set of $K_U$values generated from a block of vertices

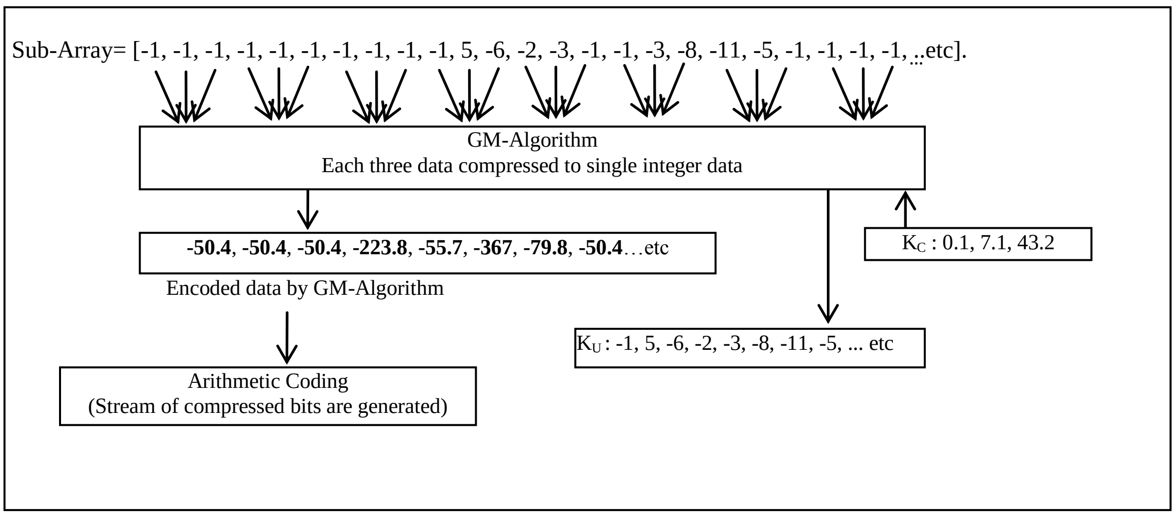
3. Connectivity Compression

Several algorithms have been developed to address the problem of compactly encoding the connectivity of polygonal meshes, both as the theoretical problem of short encodings of embedded graphs and as a practical problem of compressing the incidence table of the triangle mesh in a 3D model.

Triangulated meshes represent geometric connectivity. In a 3D OBJ file, each triangle is followed by reference numbers representing the index of the vertices in the 3D file. These reference numbers are arranged in ascending order in most 3D OBJ files. We refer to these as *regular triangles*. One of regular triangles' advantages is that they can be lossless compressed in a few bits by applying a differential process (e.g. the differential processed fined by Equation (2) applied to all reference numbers). The resulting 1D-array is divided into sub-arrays, and each sub-array encoded independently by the GM-Algorithm followed by arithmetic coding as illustrated in Figure 3. The GM-Algorithm works in the same way as applied to the vertices: three key values are generated and multiplied by three adjacent values which are then summed to a single value by Equation (3).


f 1 2 3
f 4 5 6
f 7 8 9
f 10 5 11
… etc
Triangle faces in 3D object file
1 2 3
4 5 6
7 8 9
10 5 11
… etc
Scan all vertices locations to convert matrix to 1D-array
Face = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 5, 11,…etc]
Face= [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 5, -6,..etc].
Face divided into sub-arrays for coding

(a) Triangle Face are scanned row-by-row


Sub-Array= [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 5, -6, -2, -3, -1, -1, -3, -8, -11, -5, -1, -1, -1, -1, ...etc].
GM-Algorithm
Each three data compressed to single integer data
-50.4, -50.4, -50.4, -223.8, -55.7, -367, -79.8, -50.4…etc
Encoded data by GM-Algorithm
Kc : 0.1, 7.1, 43.2
Ku : -1, 5, -6, -2, -3, -8, -11, -5, ... etc
Arithmetic Coding
(Stream of compressed bits are generated)

(b), 1D-array divided into sub-arrays, each sub-array encoded independently

Figure 3. (a) and (b): Lossless Triangle Mesh Compression by GM-Algorithm and Arithmetic Coding

4. Data Decompression: Parallel Fast-Matching-Search Algorithm (Parallel-FMS)

The decompression algorithm represents the inverse of compression using the Parallel-Fast-Matching-Search Algorithm (Parallel-FMS) to reconstruct vertices and mesh connectivity. First, the Parallel-FMS is applied to encoded block of vertices to reconstruct the original vertices as a point cloud. Second, the Parallel-FMS is applied to each encoded sub-array resulting in the reconstructed triangle mesh sub-array. Thereafter, all the sub-arrays are combined together to recover the incidence table of triangulated faces of the 3D model. Figure 4 shows the layout of the decompression algorithm.

The Parallel-FMS provides the means for fast recovery of both vertices and triangulated meshes, which has been compressed by three different keys (K_C) for each three entries. The header of the compressed file contains information about the compressed data namely K_C and K_U followed by streams of compressed encoded data. The Parallel-FMS algorithm picks up in turn each block of encoded data to reconstruct the vertices and the triangle sub-array. The Parallel-FMS uses a binary search algorithm and is illustrated through the following steps A and B:

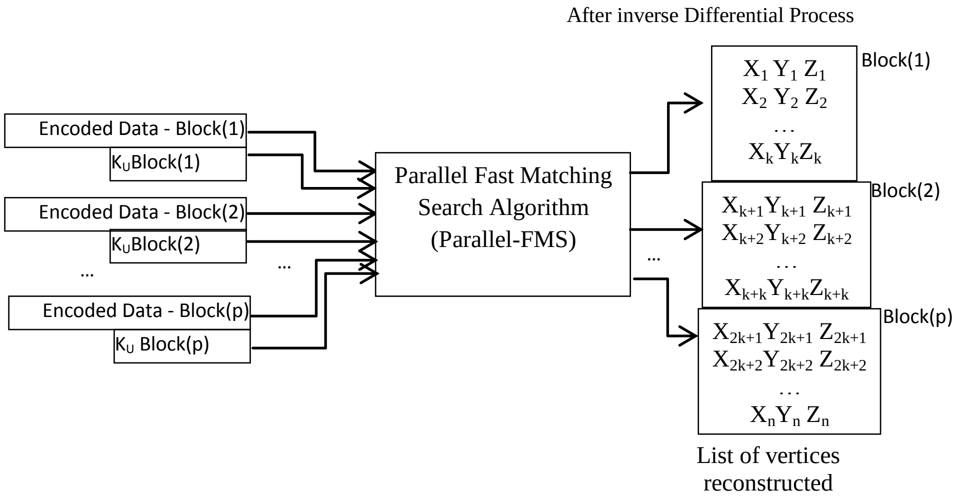
A) Initially, K_U is copied three times to sepatared arrays to estimates coordinates (X,Y,Z), that is X1=Y1=Z1, X2=Y2=Z2, X3=Y3=Z3 the searching algorithm computes all possible combinations of X with $K_U(1)$, Y with $K_U(2)$ and Z with $K_U(3)$ that yield a result R-Array illustrated in Figure 5(a). As a means of an example consider that $K_U(1)$=[X1 X2 X3] , $K_U(2)$=[Y1 Y2 Y3] and $K_U(3)$=[Z1 Z2 Z3]. Then, Equation (3) is executed 27 times to build the R-Array, as described in Figure 5(a). The match indicates that the unique combination of X, Y and Z are represented in the original vertex block.

B) A *Binary Search algorithm* **[**21] is used to recover an item in an array**.** In this research we designed a parallel binary search algorithm consisting of *k*-Binray Search algorithms working in parallel to reconstruct *k* block of vertices in the list of vertices, as shown in Figure 5(b). In each step *k*-Binary Search Algorithms compare *k*-Encoded Data (i.e. each binary search algorithm takes a single compressed data item) with the middle of the element of the R-Array, If the values match, then a matching element has been found and its R-Array's relevant (X,Y,Z) returned. Otherwise, if the search is less than the middle element of the R-Array, then the algorithms repeats its action on the sub-array to the left of the middle element or, if the value is greater, on the sub-array to the right. All *k*-Binary Search algorithms are synchronised such that the correct R-Array is returned. To illustrate our decompression algorithm, the compressed samples in Figure 2(a) (by our GM-Algorithm) can be used by our decompression algorithm to reconstruct X, Y and Z values as shown in Figure 5(c).

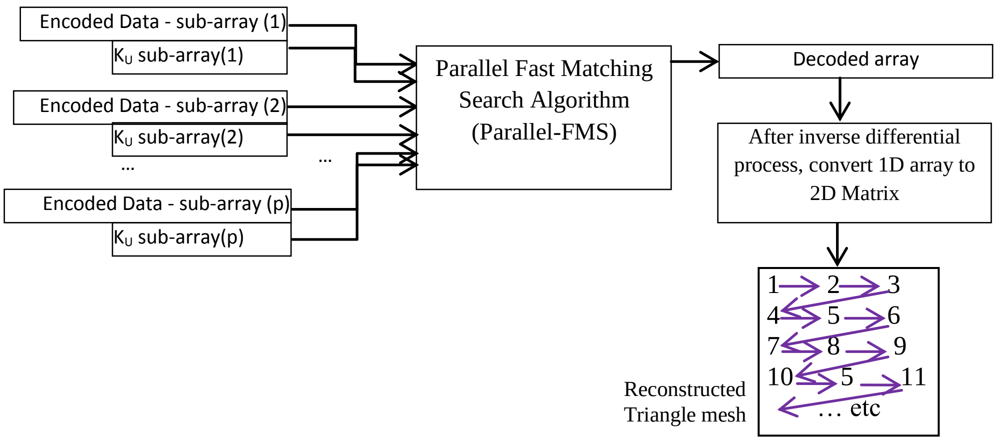
In order to Decode Triangle Faces and Vertices, reverse the differential process of Equation (2) by addition such that the encoded values in the triangle faces and vertices return to their original values. This process takes the last value at position *m*, and adds it to the previous value, and then the total adds to the next previous value and so on. The following equation defines the addition decoder [20].

$$A(i-1) = A(i-1) + A(i) \tag{4}$$

where *i= m, (m-1), (m-2), (m-3),…,2*


After inverse Differential Process
Encoded Data - Block(1)
K_UBlock(1)
Encoded Data - Block(2)
K_UBlock(2)
...
Encoded Data - Block(p)
K_U Block(p)
Parallel Fast Matching Search Algorithm (Parallel-FMS)
$X_1 Y_1 Z_1$ $X_2 Y_2 Z_2$ … $X_k Y_k Z_k$
Block(1)
$X_{k+1} Y_{k+1} Z_{k+1}$ $X_{k+2} Y_{k+2} Z_{k+2}$ … $X_{k+k} Y_{k+k} Z_{k+k}$
Block(2)
$X_{2k+1} Y_{2k+1} Z_{2k+1}$ $X_{2k+2} Y_{2k+2} Z_{2k+2}$ … $X_n Y_n Z_n$
Block(p)
List of vertices reconstructed

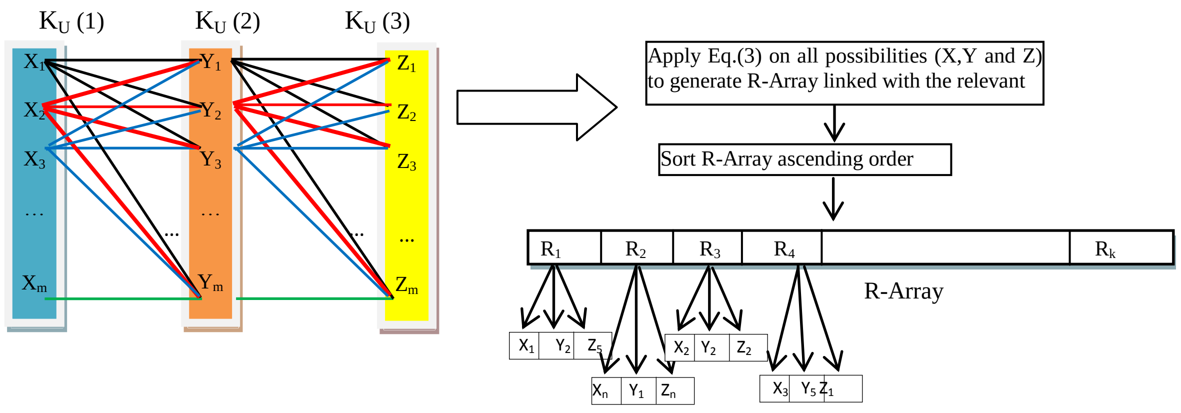
(a) Vertices (X,Y and Z) reconstructed



Encoded Data - sub-array (1)
K_U sub-array(1)
Encoded Data - sub-array (2)
K_U sub-array(2)
...
Encoded Data - sub-array (p)
K_U sub-array(p)
Parallel Fast Matching Search Algorithm (Parallel-FMS)
Decoded array
After inverse differential process, convert 1D array to 2D Matrix
Reconstructed Triangle mesh
1 2 3 4 5 6 7 8 9 10 5 11 … etc

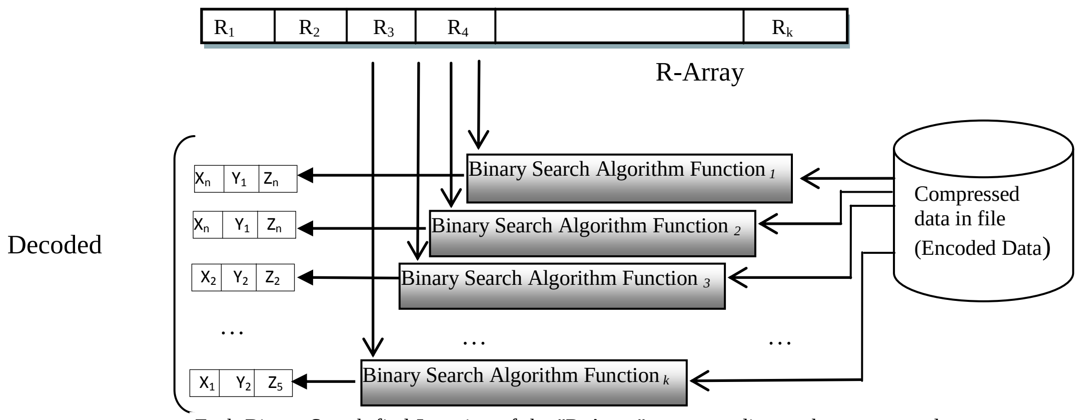
(b) Triangle mesh reconstructed

Figure 4. (a) and (b): Parallel-FMS Algorithm applied on encoded vertices and encoded triangle mesh



K_U (1)
K_U (2)
K_U (3)
X_1 X_2 X_3 … X_m
Y_1 Y_2 Y_3 … Y_m
Z_1 Z_2 Z_3 … Z_m
Apply Eq.(3) on all possibilities (X,Y and Z) to generate R-Array linked with the relevant
Sort R-Array ascending order
R_1 R_2 R_3 R_4 R_k
R-Array
X_1 Y_2 Z_5
X_n Y_1 Z_n
X_2 Y_2 Z_2
X_3 Y_5 Z_1

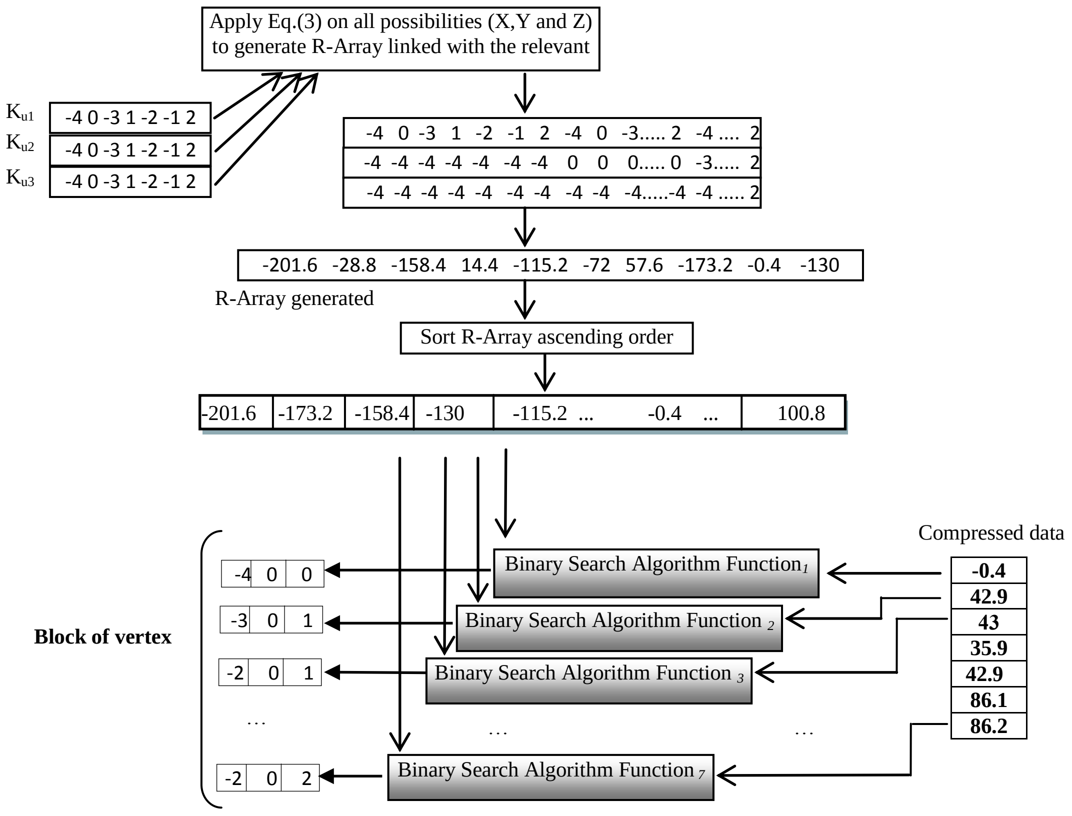
(a) Compute all the probabilities for compute all possible *k*-Encoded Data for reconstruct *k*-block of data



R_1 R_2 R_3 R_4 R_k
R-Array
Decoded
X_n Y_1 Z_n
X_n Y_1 Z_n
X_2 Y_2 Z_2
…
X_1 Y_2 Z_5
Binary Search Algorithm Function $_1$
Binary Search Algorithm Function $_2$
Binary Search Algorithm Function $_3$
Binary Search Algorithm Function $_k$
Compressed data in file (Encoded Data)

Each Binary Search find Location of the "R-Array" corresponding to the compressed data, output is relevant [X,Y,Z], which represents a original data

(b) All Binary Search algorithms work in Parallel to find group of decompressed data approximately at the same time.



(c) All Binary Search Algorithm run in Parallel to recover the sample of vertices, approximately at same time.

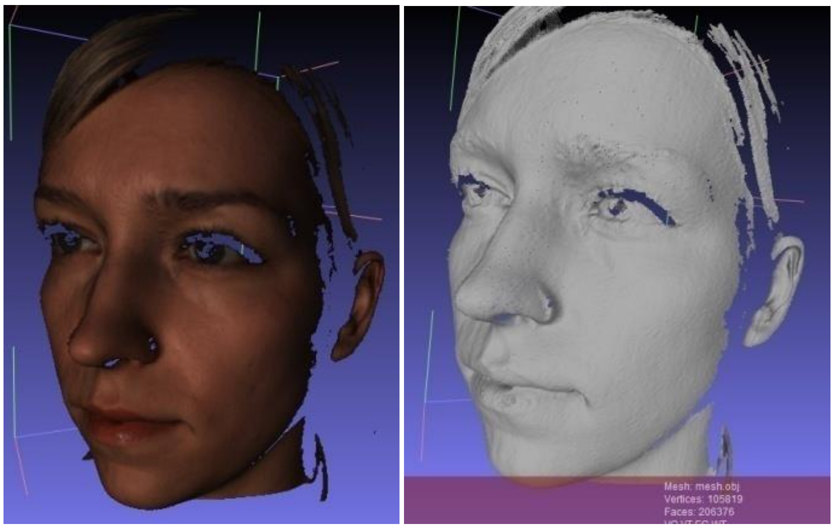
Figure 5. Parallel-FMS algorithm to reconstruct the reduced array (a) Compute all the probabilities for all possible k-Encoded Data (R-Array) by using KC combinations with KU. (b) All Binary Search Algorithm run in Parallel to recover the decompressed 3D data approximately at the same time. (c) Sample of data recovered.

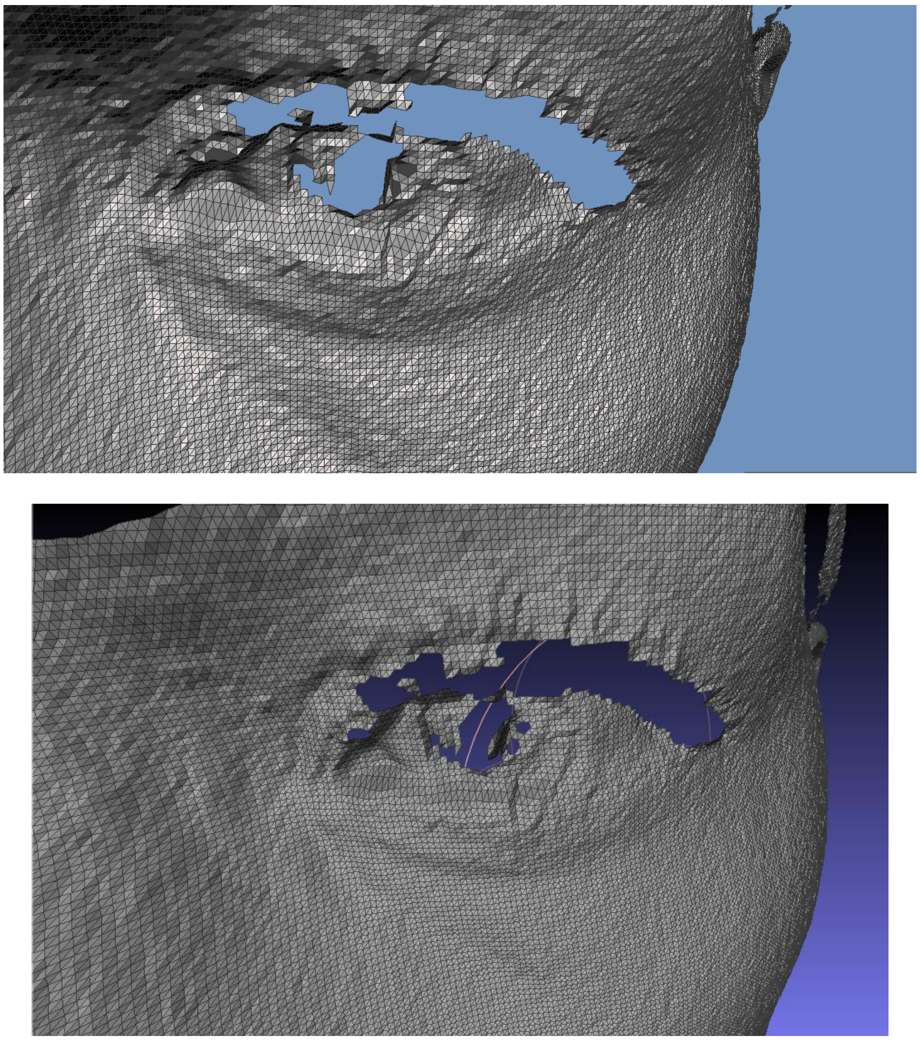
5. Experimental Results

The algorithms were implemented in MATLAB R2013a and Visual C++ 2008 running on an AMD Quad-Core microprocessor. We applied the compression and decompression algorithms to 3D data object generated by 3dsmax, CAD/CAM, 3D camera or other devices/software. Table 1 shows our compression algorithm applied to each 3D OBJ file, and Figure 6 shows the visual properties of the decompressed 3D object data for 3D images respectively. Additionally, 3D RMSE are used to compare 3D original file sizes with the recovered files. The Root Mean Square Error (RMSE) is used to refer to 3D mesh quality mathematically [22, 23] and can be calculated very easily by computing the differences between the geometry of the decompressed and the original 3D OBJ files.

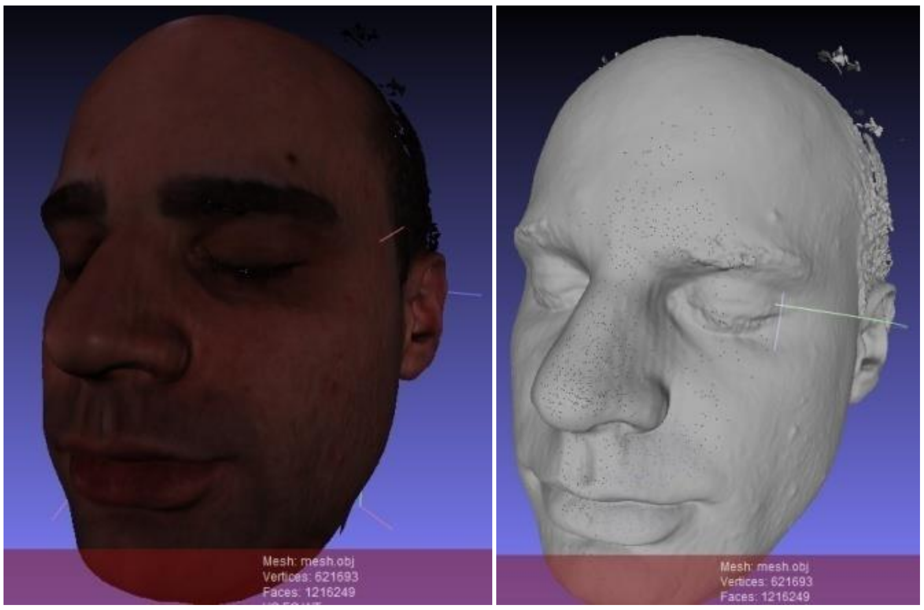
Table 1. Our compression approach results

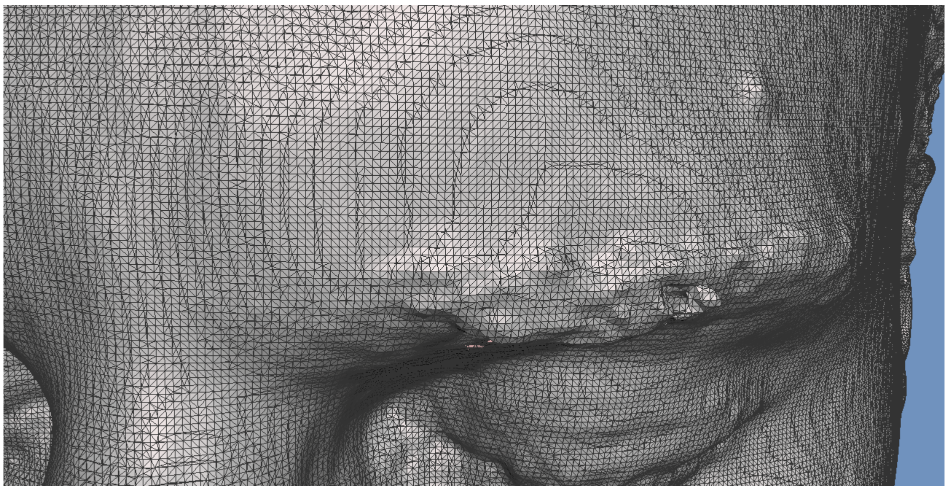
3D object Name	Original file size	Quantization value	Compressed file size	No. of Vertices (Compressed Size)	No. of Triangle faces (Compressed size)	3D RMSE (X Y Z)	Compression ratio
Face1	13.3MB	10	213 KB	105819 (187 KB)	206376 (26 KB)	0.288	98%
Face2	96MB	10	3.7 MB	621693 (1.8MB)	1216249 (1.9MB)	0.289	96%
Angel	23.5 MB	20	1.75MB	307144 (1.055MB)	614288 (715 KB)	0.288	93%
Robot	1.5 MB	400	88.9KB	23597 (56.3KB)	45814 (32.6KB)	0.289	94%
Cup	57KB	2	3.5 KB	594 (2.13 KB)	572 (1.36KB)	0.263	91%
Knot	178 KB	2	7.94KB	1440 (7.4 KB)	2880 (553 Bytes)	0.027	96%

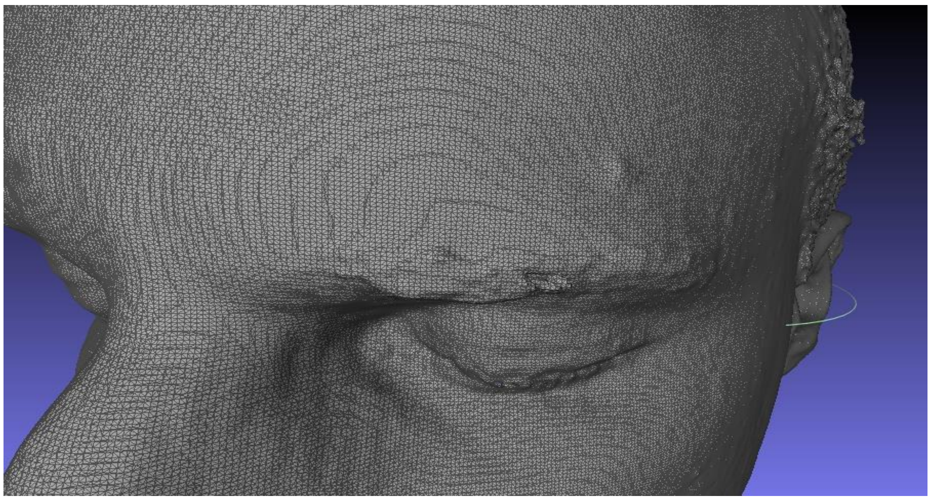




(a) (Top left) original 3D FACE1 object, (Top Right) reconstructed 3D mesh FACE1 without texture, compressed size: 213 KB, (middle) original 3D mesh zoomed by Autodesk application, (bottom) reconstructed 3D mesh zoomed by Meshlab application.



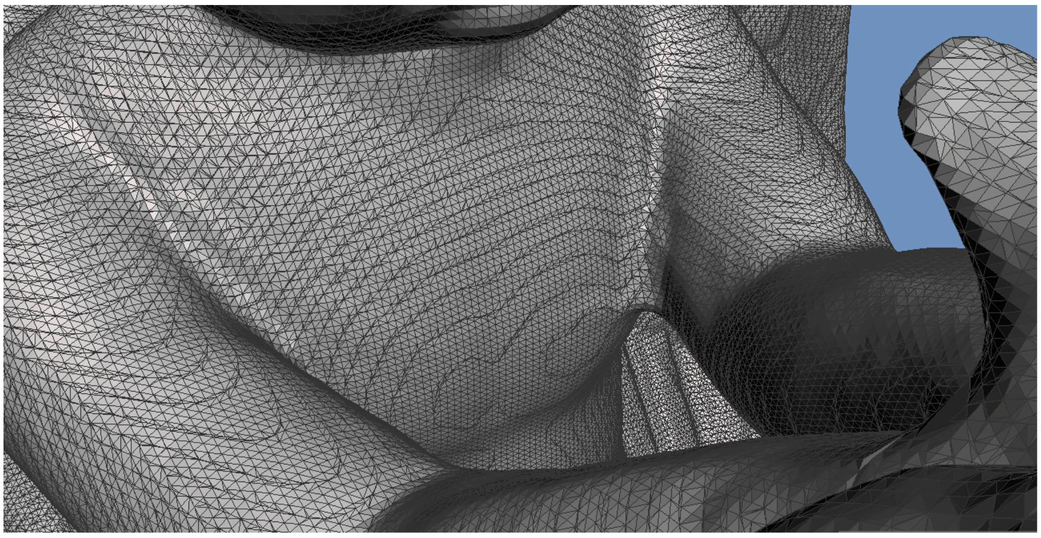


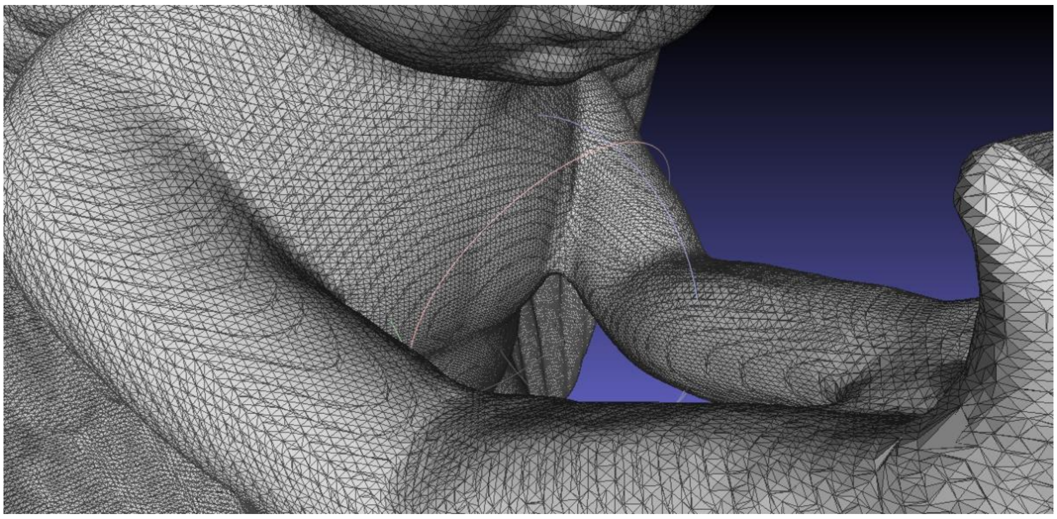


(b) (Top left) original 3D FACE2 object, (Top Right) reconstructed 3D mesh FACE2 without texture, compressed size: 3.7 MB, (middle), original 3D mesh zoomed by Autodesk application, (bottom) reconstructed 3D mesh zoomed by Meshlab application.

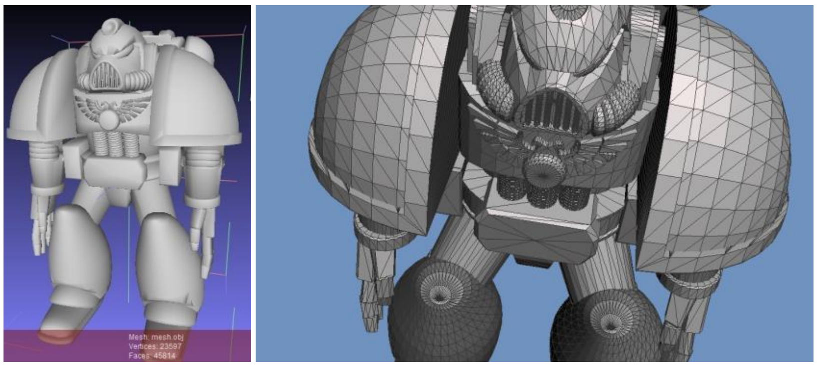
Mesh: test.obj
Vertices: 307144
Faces: 614287

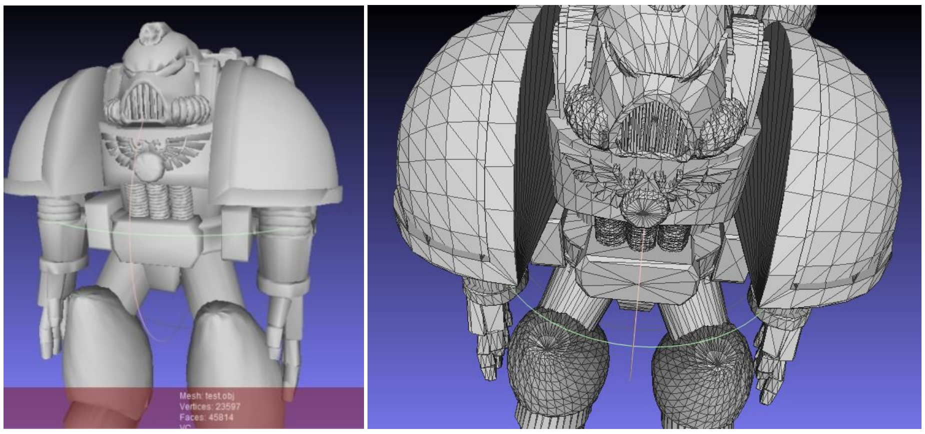
Mesh: test.obj
Vertices: 307144
Faces: 614287



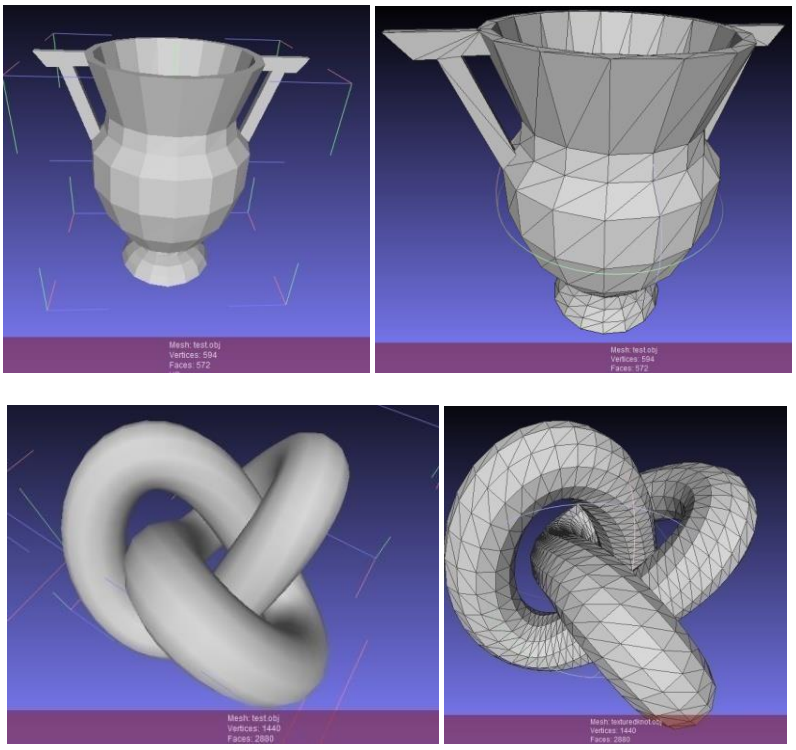


(c) (Top left) original 3D Angel object, (Right left) reconstructed 3D mesh Angel at compressed size: 1.75 MB, (middle) original 3D zoomed by Autodesk application, (bottom) reconstructed 3D mesh zoomed by Meshlab application.





(d) (Top) original 3D Robot object, (bottom) reconstructed 3D mesh Robot, at compressed size: 88.9KB



(e) (Top) original and reconstructed 3D mesh cup, at compressed size: 3.5 KB, (bottom) original and reconstructed 3D mesh Knot, at compressed size: 7.94 KB

Figure 6: (a – e) shows decompressed 3D objects by the proposed algorithms

Tables 2 and 3 show a comparison of the proposed method with the 3D file formats: VRML, OpenCTM and STL. In this research we also used a new simple file format referred here as MATLAB format. This format saves the geometry, texture and triangle faces as lossless data, in separated matrices and all the matrices are collected into a single file. We investigate this format obtaining compression ratios over 50% for most of 3D OBJ files. In comparison, our approach uses a unique format to compress 3D files over 98% in the best case; this is mostly dependent on the triangle face details.

Table 2. Our approach compared with other encoding 3D data format according to compressed size

3D object Name	Original file size	**Proposed Algorithm**	MATLAB format	VRML format	OpenCTM	STL
Angel	23.5 MB	**1.75MB**	5.31 MB	23.2 MB	1.92 MB	29.2 MB
Face1	13.3 MB	**213 KB**	4.04 MB	9.19 MB	808 KB	9.84 MB
Face2	96MB	**3.7MB**	23.3 MB	47.7MB	3.7MB	57.9MB
Robot	1.5 MB	**88.9 KB**	449 KB	1.7 MB	151 KB	2.18 MB
Cup	57 KB	**3.5 KB**	12 KB	25.2 KB	3.24 KB	28 KB
Knot	178 KB	**7.94 KB**	23.6KB	95.4KB	14.2KB	140 KB
Total Compressed Size		**5.75 MB**	33.12 MB	81.9 MB	6.57 MB	99.28 MB
Mean Compression Ratio		**95.7 %**	75.3 %	39.4%	95.1 %	26.2 %

Table 3. Our approach compared with other encoding 3D data format according to 3D RMSE

3D object	**Proposed Method**	MATLAB	VRML	OpenCTM	STL
Angel	**0.288**	0	0.0002	44.86	46.32
Face1	**0.289**	0	0.00021	64.79	42.05
Face2	**0.288**	0	0.000109	82.23	43.44
Robot	**0.289**	0	0	0.0587	0.137
Cup	**0.263**	0	0.00000075	37.7	39.2
Knot	**0.027**	0	0.000105	47.65	12.62

6. Conclusion

This research has presented and demonstrated a new method for 3D data compression and compared the quality of compression through 3D reconstruction, 3D RMSE and the perceived quality of the 3D visualisation. The method is based on minimization of geometric values to a stream of new integer data by the GM-Algorithm. Mesh connectivity is partitioned into groups of data, where each group is compressed by the GM-Algorithm followed by arithmetic coding. We note that some of the existing 3D file formats do not efficiently encode geometry and connectivity, as a simple format developed in MATLAB showed higher compression ratios than STL and VRML. The results show that our approach yields high quality encoding of 3D geometry and connectivity with high compression ratios compared to a number of standard 3D data formats. The slight disadvantage is a larger number of steps for decompression, leading to increased execution time at decoding stage, making the method slower than 3D standard compression methods. Further research includes investigation of methods to speed up decoding, possibly by sorting the *R-Array* entries by frequency. Also, a comparative analysis with a larger number of 3D file formats and compression technique is forthcoming.

References

[1] R. Koenen. (1999) Mpeg-4: Multimedia for our time. *IEEE Spectrum*, 36(2):26–33.
[2] Mpeg-4 overview Seoul revision, (1999). ISO/IEC JTC1/SC29/WG11 Document No. W2725
[3] The Virtual Reality Modeling Language (1999) . http://www.web3d.org, September 1997. ISO/IEC 14772-1.
[4] M.M. Chow. Optimized geometry compression for real-time rendering. In *IEEE Visualization'97 Conference Proceedings*, pages 347–354, 1997.
[5] M. Deering. (1995) *Geometric Compression*. *In Siggraph'95 Conference Proceedings*, pages 13–20,
[6] G. Taubin and J. Rossignac.(1998) Geometry Compression through Topological Surgery.*ACM Transactions on Graphics*, 17(2):84–115.
[7] S. Gumhold and W. Strasser (1998). *Real time compressions of triangle mesh connectivity - In Siggraph'98 Conference Proceedings*, pages 133–140, July 1998.
[8] J. Li and C.C. Kuo (1998). *Progressive Coding of 3D Graphics Models - Proceedings of the IEEE*, 86(6):1052–1063.
[9] G. Taubin, W.P. Horn, and F. Lazarus (1997) . The VRML Compressed Binary Format, June 1997 http://www.research.ibm.com/vrml/binary.
[10] J. Rossignac. Edgebreaker (1999) : Connectivity compression for triangular meshes. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):47–61.
[11] C. Bajaj, V. Pascucci, and G. Zhuang.Single (1999) *Resolution compression of arbitrary triangular meshes with properties - In IEEE Data Compression Conference Proceedings*.
[12] C. Touma and C. Gotsman (1998). *Triangle mesh compression* - In *Graphics Interface Conference Proceedings*, Vancouver.
[13] H. Hoppe (1996) *Progressive meshes - In Siggraph'96 Conference Proceedings*, pages 99–108, August 1996.
[14] H. Hoppe (1998) Efficient implementation of progressive meshes. *Computers& Graphics, 1998*.
[15] G. Taubin,W. Horn, and P. Borrel (1999). Compression and transmission of multi-resolution clustered meshes. *Technical Report RC-21398, IBM Research*, February 1999.
[16] F. Bossen (1999) On The Art Of Compressing Three-Dimensional Polygonal Meshes And Their Associated Properties .PhD thesis, École Poly technique Fédérale de Lausanne (EPFL), June 1999.
[17] A. Guéziec, G. Taubin, F. Lazarus, and W.P. Horn (1998). *Converting sets of polygons to manifold surfaces by cutting and stitching. In IEEE Visualization'98 Conference Proceedings*, pages 383–390.
[18] E.S. Jang, S.J. Kim, M. Song, M. Han, S.Y. Jung, and Y.S. Seo (1998). Results of ce m5 error resilient 3D mesh coding. *ISO/IEC JTC 1/SC 29/WG 11 Input Document No. M4251.*
[19] M. M. Siddeq, M. A. Rodrigues (2014) A Novel Image Compression Algorithm for high resolution 3D Reconstruction, *3D Research. Springer Vol. 5 No.2.*DOI 10.1007/s13319-014-0007-6
[20] M. M. Siddeq, M. A. Rodrigues (2015) A Novel 2D Image Compression Algorithm Based on Two Levels DWT and DCT Transforms with Enhanced Minimize-Matrix-Size Algorithm for High Resolution Structured Light 3D Surface Reconstruction, *3D Research. Springer Vol. 6 No.3.*DOI 10.1007/s13319-015-0055-6
[21] Knuth, Donald (1997). Sorting and Searching: Section 6.2.1: Searching an Ordered Table, *The Art of Computer Programming (3rd Ed.), Addison-Wesley*. pp. 409–426. ISBN 0-201-89685-0
[22] I.E. G.Richardson (2002) Video Codec Design, *John Wiley & Sons*.
[23] K. Sayood, (2000) Introduction to Data Compression, *2nd edition, Academic Press, Morgan Kaufman Publishers*.