3D Res (2016) 7:13
DOI 10.1007/s13319-016-0091-x

3DR EXPRESS

Novel 3D Compression Methods for Geometry, Connectivity and Texture

M. M. Siddeq · M. A. Rodrigues

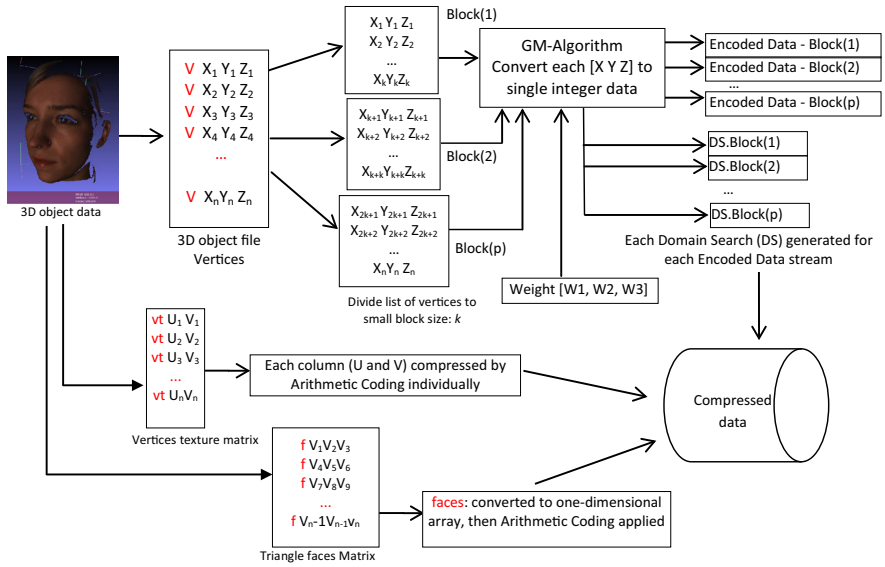
Received: 1 March 2016 / Revised: 31 March 2016 / Accepted: 3 April 2016 / Published online: 9 April 2016

Abstract A large number of applications in medical visualization, games, engineering design, entertainment, heritage, e-commerce and so on require the transmission of 3D models over the Internet or over local networks. 3D data compression is an important requirement for fast data storage, access and transmission within bandwidth limitations. The Wavefront OBJ (object) file format is 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, normals, texture coordinates and other parameters) describing the mesh surface. In this paper we introduce a new method to compress geometry, connectivity and texture coordinates by a novel Geometry Minimization Algorithm (GM-Algorithm) in connection with arithmetic coding. First, each vertex (x, y, z) coordinates are encoded to a single value by the GM-Algorithm. Second, triangle faces are encoded by computing the differences between two adjacent vertex locations, which are compressed by arithmetic coding together with texture coordinates. We demonstrate the method on large data sets achieving compression ratios between 87 and 99 % without reduction in the number of reconstructed vertices and triangle faces. 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 VRML, OpenCTM and STL highlighting the performance and effectiveness of the proposed method.

M. M. Siddeq (✉) · M. A. Rodrigues
GMPR-Geometric Modelling and Pattern Recognition Research Group, Sheffield Hallam University, Sheffield, UK
e-mail: mamadmmx76@yahoo.com

M. A. Rodrigues
e-mail: M.Rodrigues@shu.ac.uk

Graphical Abstract



Keywords 3D object compression and decompression · GM-algorithm · Parallel-FMS-algorithm

1 Introduction

Algorithms for 3D data compression and reconstruction are enabling technologies where cheap storage and data transmission over the network are required. Examples of applications can be found in security, engineering, CAD/CAM collaborative design, medical visualisation, entertainment, e-commerce and geographical information systems among others [16]. Concerning geometry and connectivity compression, Rodrigues and Robinson [17] used PDE-Partial Differential Equations to lossycompression and reconstruction of mesh surfaces with no loss of accuracy in the context of 3D face recognition. They proposed that surface patches can be compressed as a 2D image together with 3D calibration parameters, transmitted over a network and remotely reconstructed (geometry, connectivity and texture map) at the receiving end with the equivalent resolution as the original data. Extending that research, Siddeq and Rodrigues [21] proposeda 2D image compression method based on DWT and DCT which has been demonstrated in the context of structured light 3D reconstruction. The method produces anested series of high-frequency matrices which are coded by a Minimize-Matrix-Size Algorithm (MMS). At decompression stage, a Limited-Sequential Search Algorithm (LSS) is used torecover the original 2D image which is then used for 3D reconstruction of the original mesh. The advantagesofthe method arethe recovery of high resolution imageswith compression ratios up to 98 %. Although superior to the PDE compression for the same application, the method proved very complex resulting in execution time of the order of minutes for a high resolution image.

The methods proposed in this paper are more general than the compression of structured light images for 3D reconstruction, as they are applicable directly to 3D data geometry and connectivity and include texture mapping. Peng et al. [11] reviewed technologies for 3D data compression with particular focus on triangular meshes. In an earlier survey of 3D compression methods Alliez and Gotsman [18] focus on compression techniques for single-rate and progressive mesh coding based on geometry and connectivity. Compression methods are thus, focused on representing the geometry and connectivity of the vertices in the triangulated mesh. Geometrical methods aim to reduce the size of the mesh by simplifying its geometry and approaches include geometry coding [1], Generalized Triangle Mesh [3], triangulated model techniques, and quantization techniques [11, 23] where rates of over

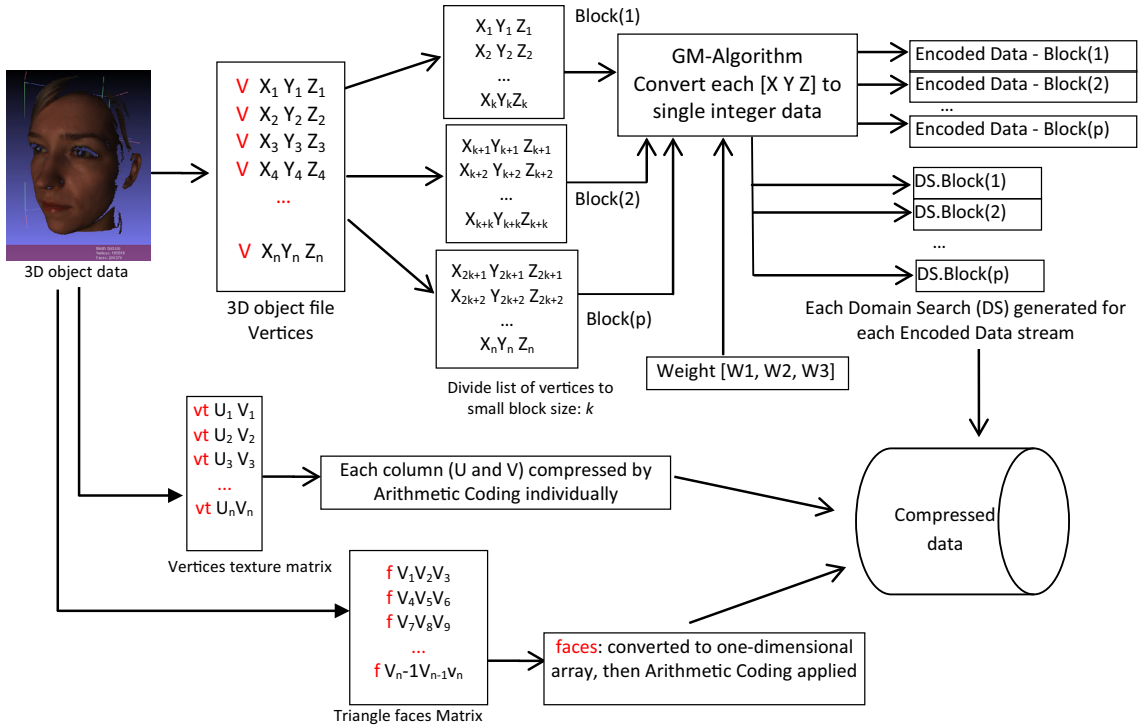


Fig. 1 The proposed 3D data compression method

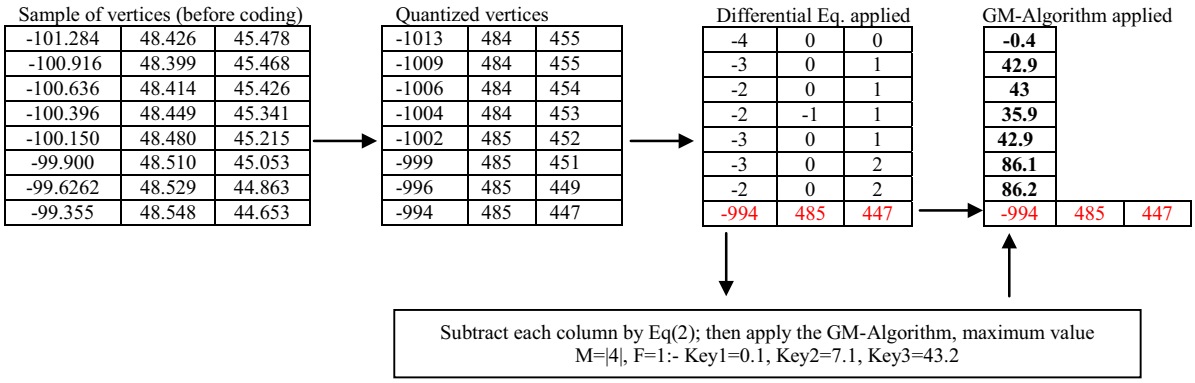


Fig. 2 Sample of vertices compressed by GM-Algorithm

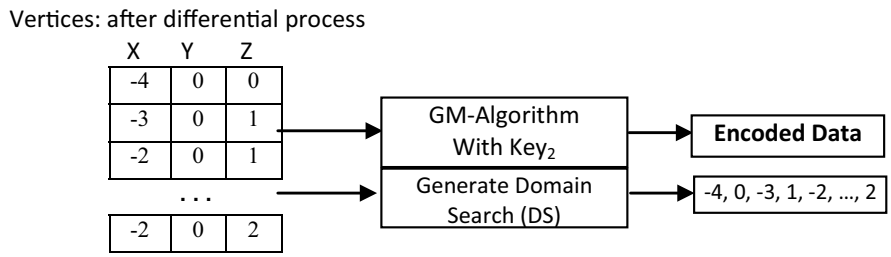


Fig. 3 Domain Search (DS) generatedfrom a block of vertices

80:1 have been achieved. Examples of coding connectivity include the topological surgery algorithm [8], and the Edgebreaker algorithm [20, 25]. Products also exist in the market that claim a 95 % lossless file reduction [22] for regular geometric shapes. 3D data compression related researchconcentrate either on fast rendering or on maximum compression and therefore the following discussion is focused on these topics.

1.1 Compression for Fast Rendering

In this section we discuss representations of triangle meshes that are used for transmission to graphics hardware. 3D-hardware support is primarily based on the rendering of triangles, specified by its three vertices, where each vertex contains three coordinates, possibly the surface normal, material attributes and/or

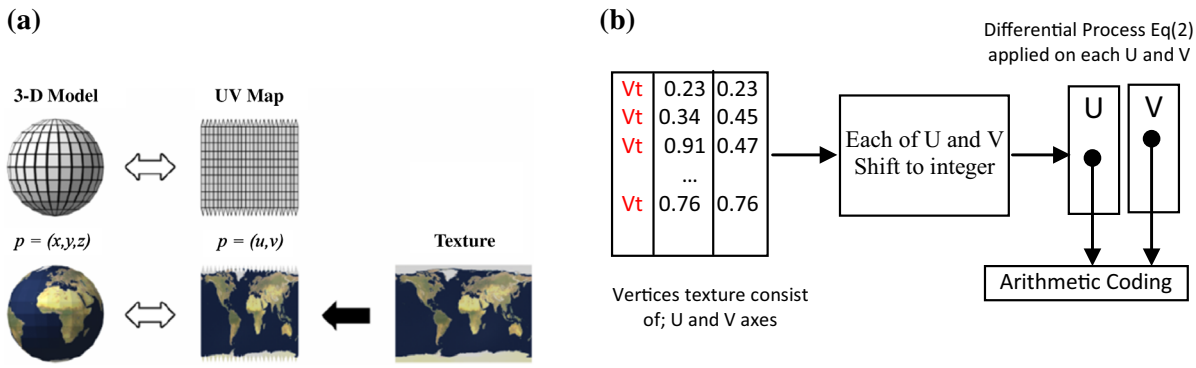
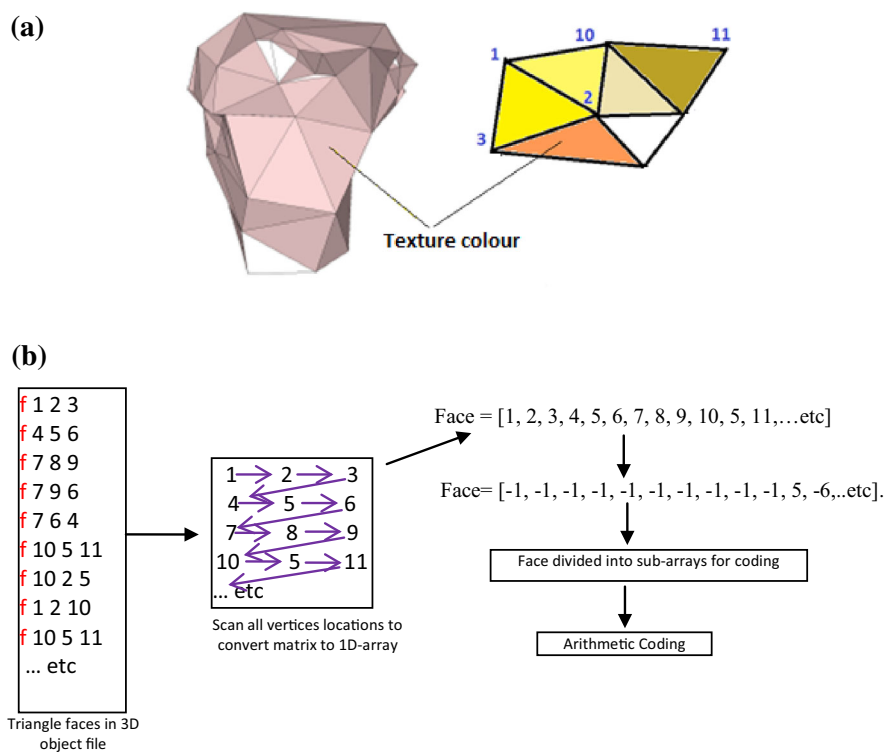


Fig. 4 **a** Texture in UV space related to 3D object, **b** UV texture mapping compression

Fig. 5 **a** 3D mesh represented as vertex indices in a 3D OBJ file; **b** In our method, differential vertex indices are lossless compressed by arithmetic coding



texture coordinates. The coordinates and normals are specified with floating point values, such that a vertex may contain data of up to 36 bytes. Thus the transmission of a vertex is expensive and the simple approach of specifying each triangle by the data of its three vertices is wasteful as for an average triangle mesh each vertex must be transmitted six times [7].

The introduction of triangle strips helped to save unnecessary transmission of vertices. Two successive triangles in a triangle strip are joined at an edge. Therefore, from the second triangle on, two vertices of the previous triangle can be combined with only one new vertex to form the next triangle. As with each triangle at least one vertex is transmitted and as an average triangle mesh has twice as many triangles as vertices, the maximal gain is that each vertex has to be transmitted only about twice. Two kinds of triangle strips are commonly used—the sequential and the generalized triangle strips. In generalized triangle strips an additional bit is sent with each vertex to specify to which of the free edges of the previous triangle the new vertex is attached. Sequential strips even drop this bit and impose that the triangles are attached alternating. OpenGL [6] evolved from the commonly used standard for graphics libraries allowing generalized triangle strips in earlier versions, but the current version is restricted to sequential strips.

1.2 Maximum Mesh Compression

The strategy for fast rendering described above can also be used for the compression of triangle mesh connectivity. Instead of retransmitting a vertex, a reference is inserted into a compressed representation. If a vertex from the buffer is referenced its index within the buffer enters the compressed representation. In the triangle strips of Evans et al. [2] each vertex appears about 2.5 times. The vertices can be

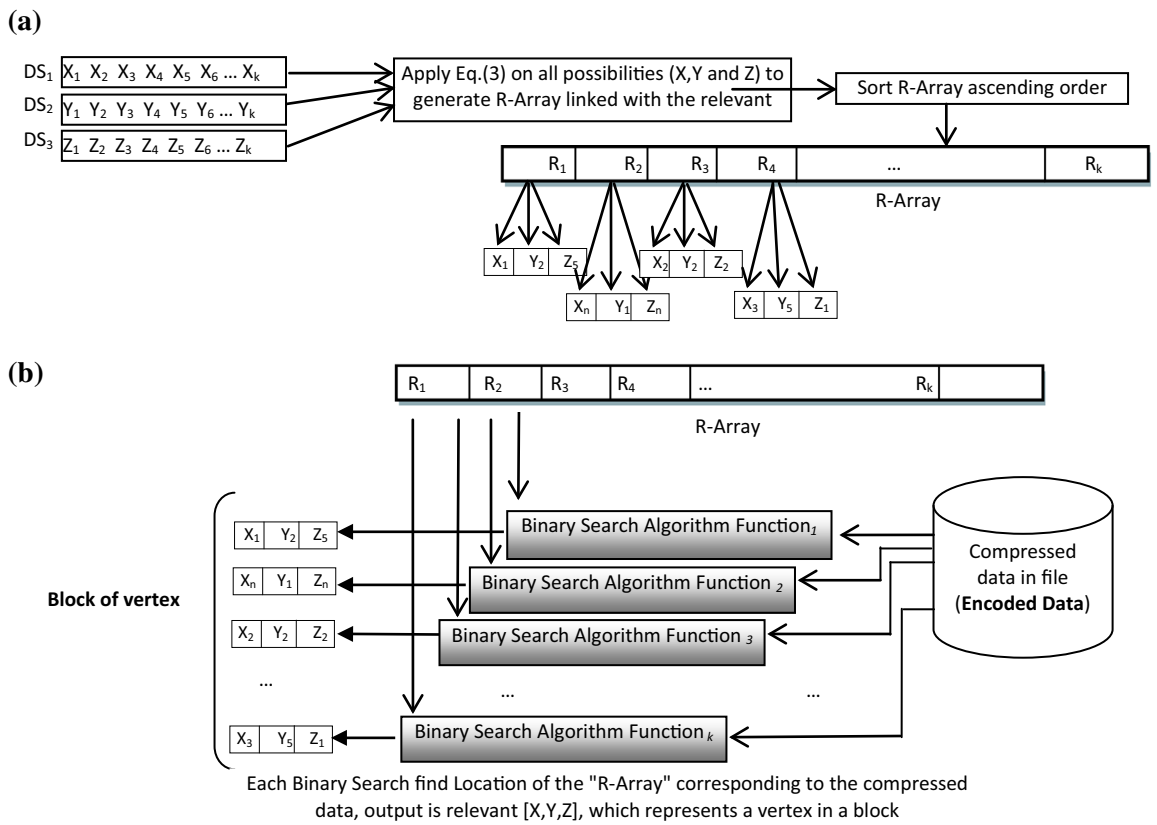


Fig. 6 Parallel-FMS algorithm to reconstruct the reduced array. **a** Compute all the probabilities for all possible *k*-Encoded Data (R-Array) by using Weight combinations with DS. **b** All Binary Search Algorithms run in Parallel to recover the decompressed vertices data approximately at same time

rearranged into the order they appear the first time and only the indices of 1:5n vertices need to be inserted in the compressed representation. One additional bit per triangle is needed to specify whether the next vertex is used the first time or the index of an already used vertex follows. This sums up to about 1 + 0.75 (log2 n) bits per triangle. The disadvantage of this approach is that the storage needs to grow with the size of the triangulated mesh. The measurements of Deering in [10] show that the generalized mesh approach theoretically consumes between eight and eleven bits per triangle if an optimal stripper is available.

Taubin et al. [4] propose a very advanced global compression technique for the connectivity of triangle meshes. The method is based on a similar optimization problem as for sequential triangle strips and the authors suggest that it is NP-complete. Their approximation allows compression to only two bits per triangle and there exist triangle meshes which consume only one bit per triangle. The decompression splits into several processing steps over the complete mesh, which makes the approach unsuitable to speed up hardware driven rendering.

Our compression technique introduces a new idea for geometry compression by geometry minimization algorithm and the triangle faces (connectivity) encoded by computing the differences between two adjacent vertices and then encoding each group of connectivity by arithmetic coding. This approach compares favourably with a number of 3D data file formats techniques focusing on compression ratio as it is demonstrated in the experimental section of this paper. This paper is organized as follows. Section 2 introduces the Geometry Minimization algorithm (GM-algorithm) applied to vertices. Section 3 describes the texture compression by using soft-quantization with arithmetic coding, also in this section the connectivity (triangle faces) are compressed by computing the differences between two adjacent faces. Section 4 describes the Parallel Fast Matching Search Algorithm (Parallel-FMS) used to reconstruct the vertex data. Section 5 describes

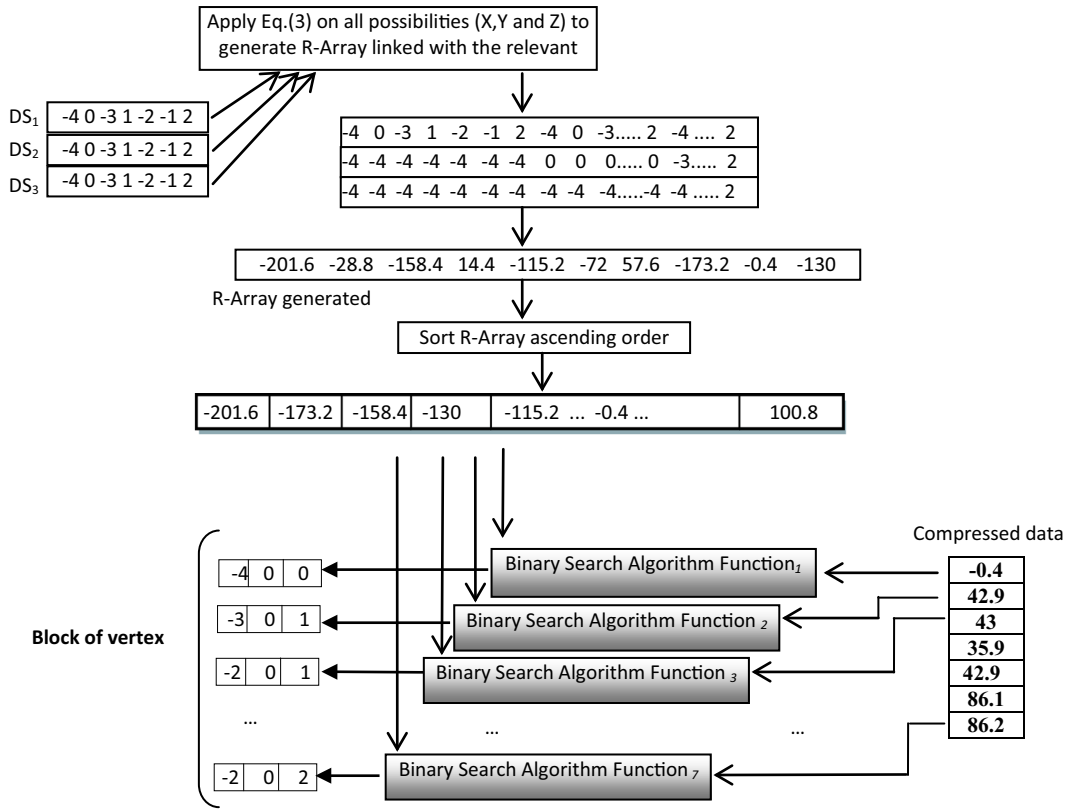


Fig. 7 All Binary Search Algorithm run in Parallel to recover the decompressed vertices data approximately at same time

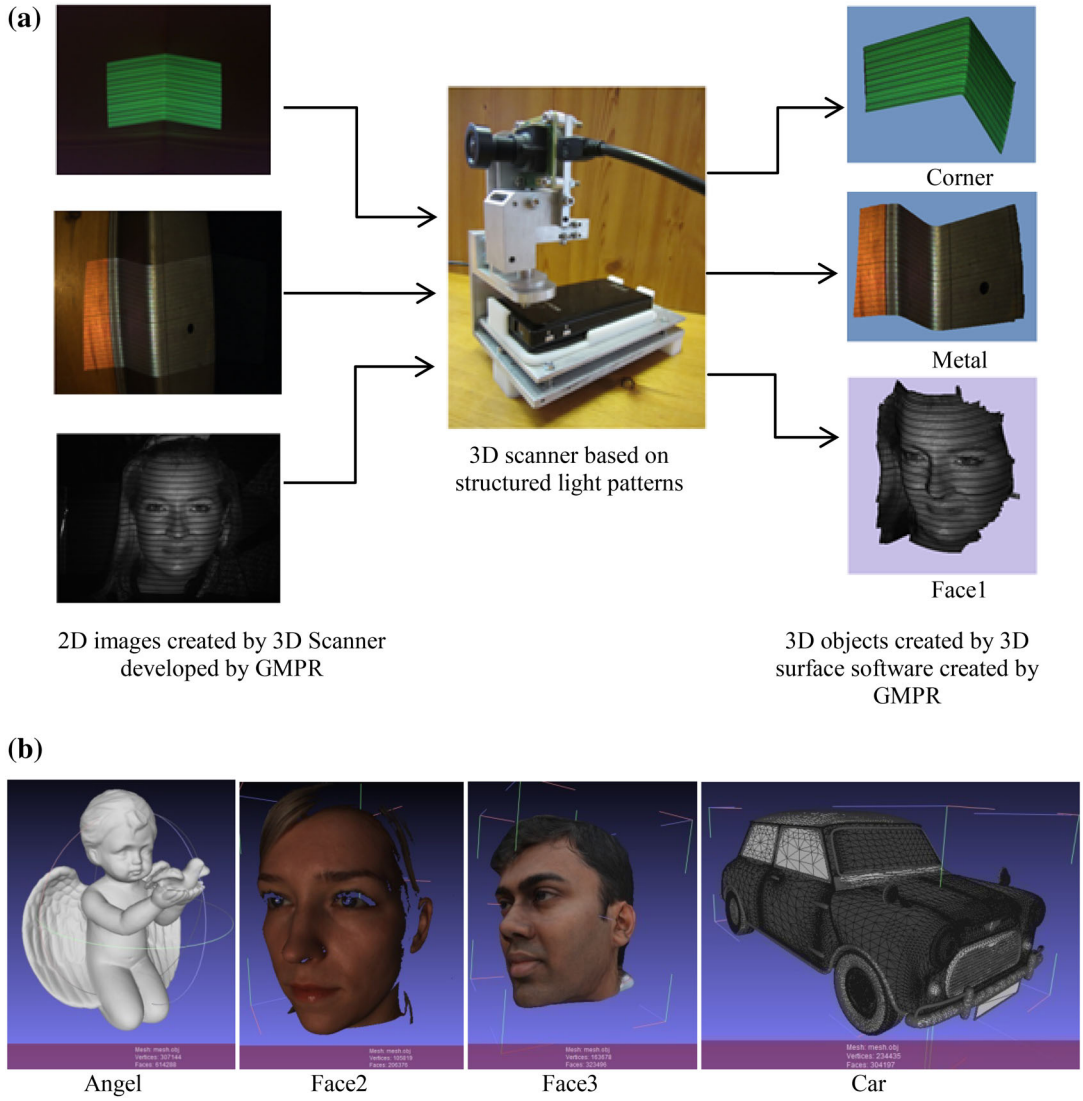


Fig. 8 3D objects tested by our compression and decompression algorithm. **a** 3D objects created by 3D scanner and 3D software developed by GMPR, **b** 3D objects created by other scanning methods and 3D model building software

experimental results for 3D data compressionand the results are compared with other 3D file formats. Finally, Sect. 6 provides a summary and a conclusion.

2 Vertex Compression

A 3D object is a mesh of polygons whose union defines a surface bounding a solid object. A 3D OBJ file considered here consists of a list of vertices (geometry), a list of texture and a list of triangle faces (connectivity—each represented by 3 or 4 vertex indices). Each vertex consists of X, Y and Z coordinates (the size for each vertex no less than 96-bit floating point value). In the method proposed here, the (X, Y, Z) floating point is shifted to an integer value to reduce the number of bits (i.e. this process meanslossyvertex data). The shift S can be any integer value but for efficiency reasons we define $1 < S \leq 9000$. The new vertex V is defined as:

Table 1 Compression and decompression results by the proposed method

3D image Name	Original file size (MB)	Shift value	Compressed file size	No. of vertices (compressed size)	No. of triangle faces (compressed size)	3D RMSE (X, Y and Z)	2D RMSE (Texture UV)
Corner	2.91	1	19.1 KB	24,168 (11 KB)	46,878 (4 KB)	1.034	5.77×10^{-4}
		10	32.5 KB	24,168 (24 KB)	46,878 (4 KB)	1.035	5.77×10^{-4}
Metal	5.34	10	36.7 KB	43,932 (23 KB)	86,330 (6 KB)	1.0618	5.78×10^{-4}
		50	53.2 KB	43,932 (39 KB)	86,330 (6 KB)	1.061	5.78×10^{-4}
Face1	4.7	10	36.1 KB	38,831 (20 KB)	76,098 (9 KB)	1.360	5.78×10^{-4}
		20	40.2 KB	38,831 (23.7 KB)	76,098 (9 KB)	1.362	5.78×10^{-4}
Angel	23	10	2.67 MB	307,144 (905 KB)	614,287 (1.79 MB)	2.022	NON
		50	3.09 MB	307,144 (1.29 KB)	614,287 (1.79 MB)	2.023	NON
Car	14.1	1	917 KB	234,435 (314 KB)	304,197 (603 KB)	0.726	NON
		10	1.34 MB	234,435 (772 KB)	304,197 (603 KB)	0.733	NON
Face2	13.3	2	290 KB	105,819 (99.4 KB)	206,376 (174 KB)	1.283	5.77×10^{-4}
		10	378 KB	105,819 (186 KB)	206,376 (174 KB)	1.285	5.77×10^{-4}
Face3	55	1000	6.21 MB	323,496 (833 KB)	163,678 (5.18 MB)	0.993	4.715×10^{-4}
		5000	6.42 MB	323,496 (1.02 MB)	163,678 (5.18 MB)	0. 995	4.715×10^{-4}

$$V_{XYZ} = round[V_{XYZ} \cdot S] \quad (1)$$

by above Eq. (1) yielding a newset of quantized vertices $V_{XYZ} = [X, Y, Z]$ where each vertex size is less than 32-bit (i.e. 32-bit minimum size to represent (*X*, *Y*, *Z*) while in previous work each axes need at least 16-bit [18], [7]). Additionally, we reduced the number of bits for each vertex to less than 16-bit by calculating the differences between two adjacent coordinates as follows:

$$D_i = D_i - D_{(i+1)} \quad (2)$$

where $i = 1, 2, 3 \ldots m - 1$ and m is the size of the list of vertices. The differential process defined in Eq. (2) increases data redundancy and it is applied to each axis independently [21]. Thus, these two steps reduce the storage cost of geometry and ensure a sufficient geometric accuracy for most applications.

Figure 1 illustrates the main steps in the proposed method. After the differential process is applied to the vertices, the resulting list is divided into k number of non-overlapping blocks such that each block can be worked independently and in parallel by the GM-Algorithm. This significantly speeds up compression and decompression. The GM-Algorithm is based on defining three weight values (W_1, W_2 and W_3) which are multiplied by the three vertex coordinates (*X*, *Y*, *Z*) in turn and then summed to a single integer value. Equation (3) defines the*Encoded Data* by the GM-Algorithm [21] for each triplet of vertex values:

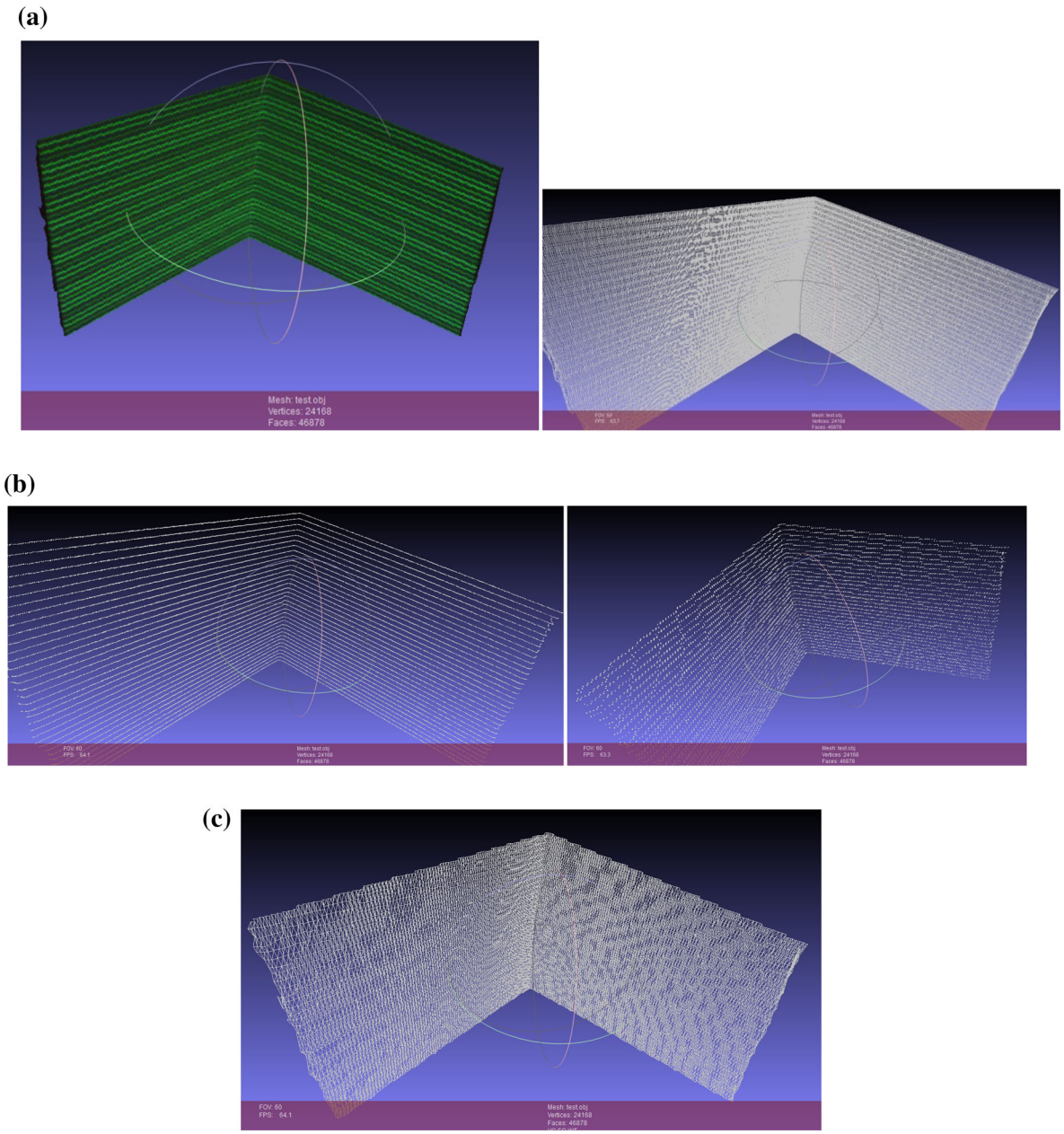


Fig. 9 Decompressed 3D Corner image at compression size: 32.5 and 19.1 KB **a** (*left*) 3D Corner object with texture, (*right*) 3D mesh Corner shows the details of the object, at compresses size: 32.5 KB, **b** (*left*) 3D Corner's vertices organized as structure lines at compression size: 32.5 KB, (*right*) shows vertices for 3D corner at compression size:19.1 KB, the verticesmoved slightly from their original position, **c** 3D mesh corner shows the details of the object, at compresses size: 19.1 KB, the image shows the trianglesslightly more degraded than in previous result

$$Encoded\,Data_{(i)} = W_1 V_{X(i)} + W_2 V_{Y(i)} + W_3 V_{Z(i)}. \quad (3)$$

where V_X, V_Y, V_Z represent the list of vertices (X, Y, Z) within a block of size k. The weight values (W_1, W_2 and W_3) are generated by the following algorithm described in pseudo code from the maximum value in the list of vertices:

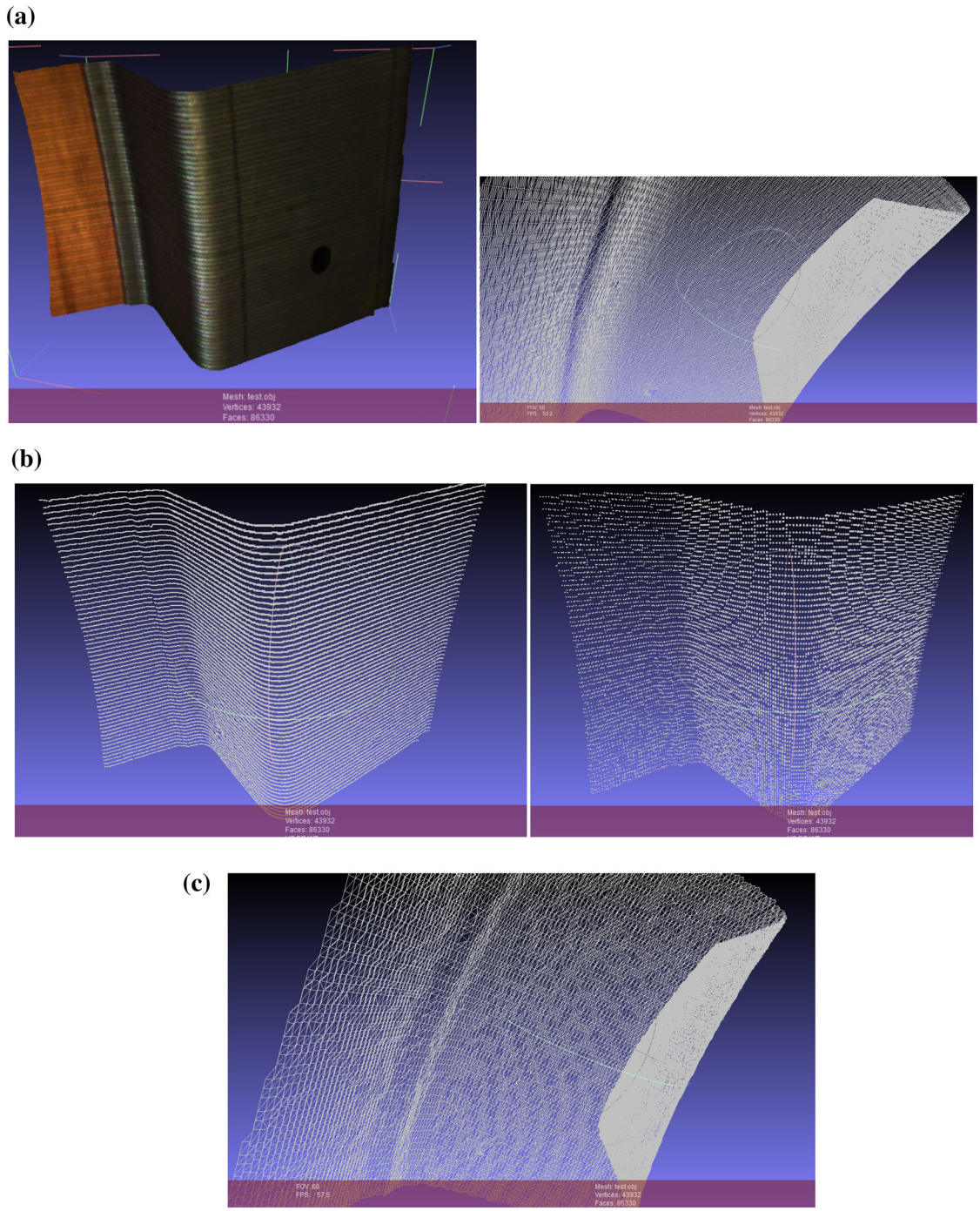


Fig. 10 Decompressed 3D Metal image at compression size: 53.2 and 36.7 KB. **a** (*left*) 3D Metal object with texture, (*right*) 3D mesh details at compresses size: 53.2 KB, **b** (*left*) 3D Metal's vertices organized as structure lines at compression size: 53.2 KB, (*right*) shows vertices for 3D corner at compression size: 36.7 KB, some vertices moved to up and down from their original position, **c** 3D mesh details of the object, at compresses size: 36.7 KB, the image shows similar high quality 3D mesh surface as in previous result

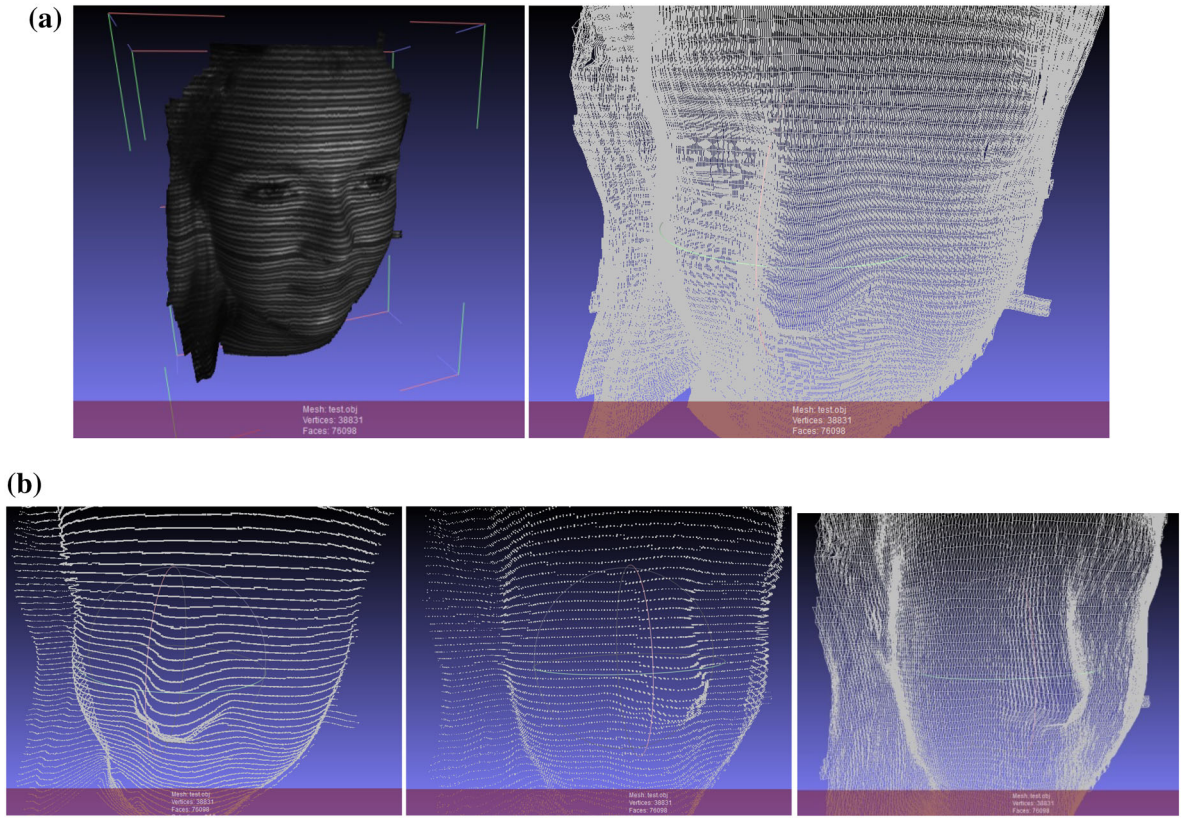


Fig. 11 Decompressed 3D Face1 image at compression size: 40.2 and 36.1 KB **a** (*left*) 3D Face1 object with texture, (*right*) 3D mesh Face1 details at compresses size: 40.2 KB, **b** (*left*) 3D Face1 vertices organized as structure lines at compression size: 40.2 KB, (*middle*) shows vertices for 3D Face1 at compression size: 36.1 KB, the vertices in both 3D images approximately at same original positions. (*right*) 3D mesh Face1 shows the details of the object, at compresses size: 36.1 KB, the image shows same high quality as 3D mesh in previous result

$$M = \max(V_X, V_Y, V_Z) + \frac{\max(\mathrm{V_X}, \mathrm{V_Y}, \mathrm{V_Z})}{2}$$
$$\%\, Define\ M\ as\ a\ function\ of\ maximum$$

$$W_1 = random(0, 1)$$
$$\%\, First\ weight \leq 1\ defined\ by\ random\ between\ 0\ and\ 1$$

$$W_2 = (W1 + M) + F$$
$$\%\, F\ is\ an\ integer\ factor\ F = 1, 2, 3, \ldots$$

$$W_3 = (M * W_1 + M * W_2) * F$$

The following Fig. 2 illustratesthe GM-Algorithm applied to a sample of vertices. After applying the GM-Algorithm, the likelihood for each block of vertices is selected from which a *Domain Search* (*DS*) is generated to be used in the decompression stage as illustrated in Fig. 3 with a numerical example.

3 Texture Mapping Encoding (UV) and Mesh Connectivity Encoding

In order to allow polygons in a 3D structure to be painted from a 2D image a mapping is required. This map is defined by the normalized horizontal and vertical components (u, v) of an image corresponding to each polygon in 3D. A 3D object in model space [9] and corresponding (u, v) mapping is shown in Fig. 4a. In our proposed compression algorithm, the (u, v) texture coordinates are compressed as lossy data in two steps. First, the (u, v) map is quantized by

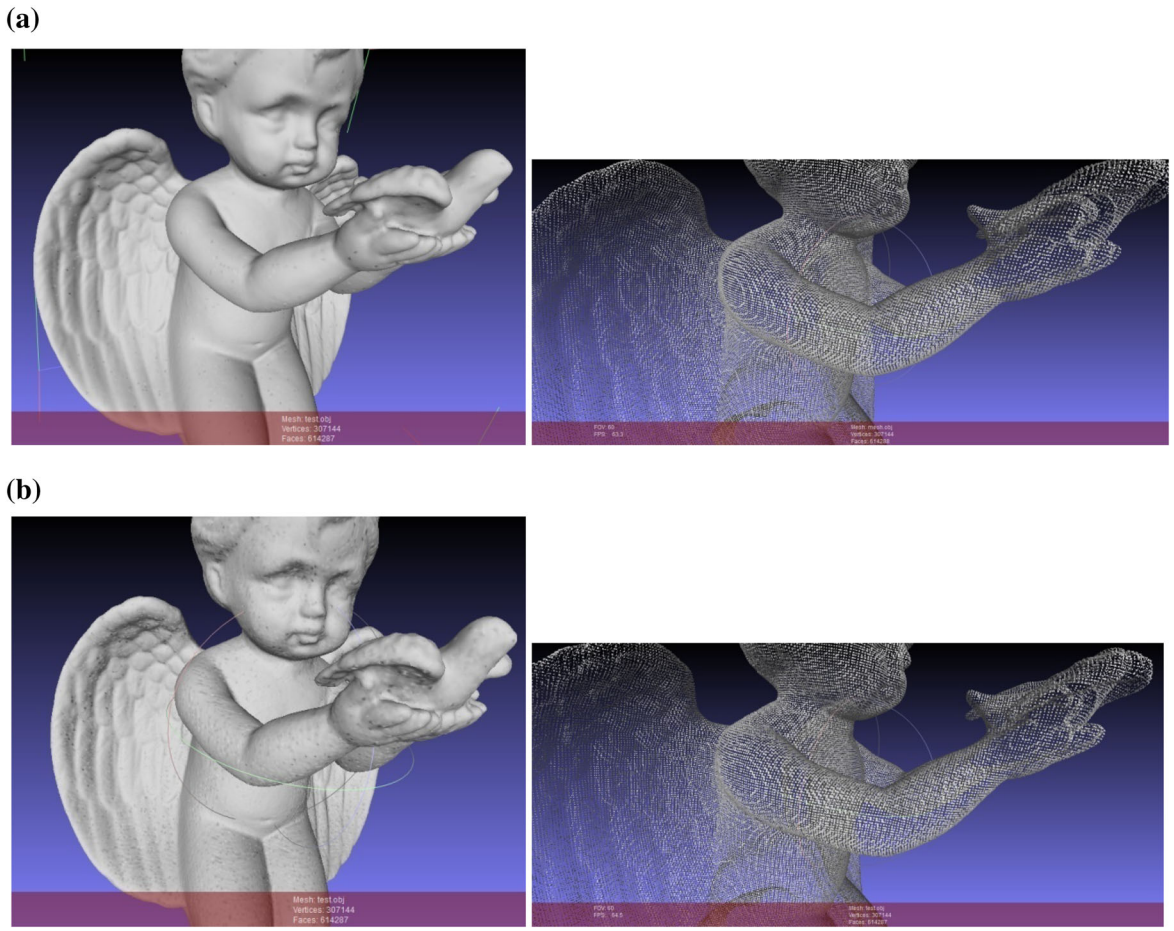


Fig. 12 Decompressed 3D Angel image at compression size: 3.09 and 2.67 MB **a** (*left*) 3D Angel object, (*right*) 3D mesh Angel shows the details of the object, at compresses size: 3.09 MB, **b** (*left*) 3D Angel object, (*right*) 3D mesh Angel shows the details of the object, at compresses size: 2.67 MB, similar from the 3D mesh in previous result

shifting the floating point to integer, i.e. Eq. (1). We adopted a shift value of 1000, meaning that each value (u, v) is in the range {8-bit and 16-bit}. Second, the differences between two adjacent data are computed by Eq. (2) applied to each axis independently as shown in Fig. 4b. Finally, arithmetic coding is applied to each (u, v) independently to produce a stream of bits.

Triangle faces represent geometric connectivity. In a 3D OBJ file, each face contains the vertex indices in the order they appear in the file, normally arranged in ascending order. Triangular faces and vertex indices are illustrated in Fig. 5a. The advantage of this representation is that each triangle can be compressed in just a few bits. In our proposed method, triangle faces are compressed by applying the differential process defined in Eq. (2). The faces are scanned row-by-row and representedas a one-dimensional array, and the final encoded array is compressed by arithmetic coding. The texture of each triangular face is represented by their (u, v) map which maps a pixel value in the image to a vertex in the 3D structure. Each value in the (u, v) map is normalised and shifted to integer. Texture values are separated from vertex indices (triangle face), they are both compressed in same way as illustrated in Fig. 5b.

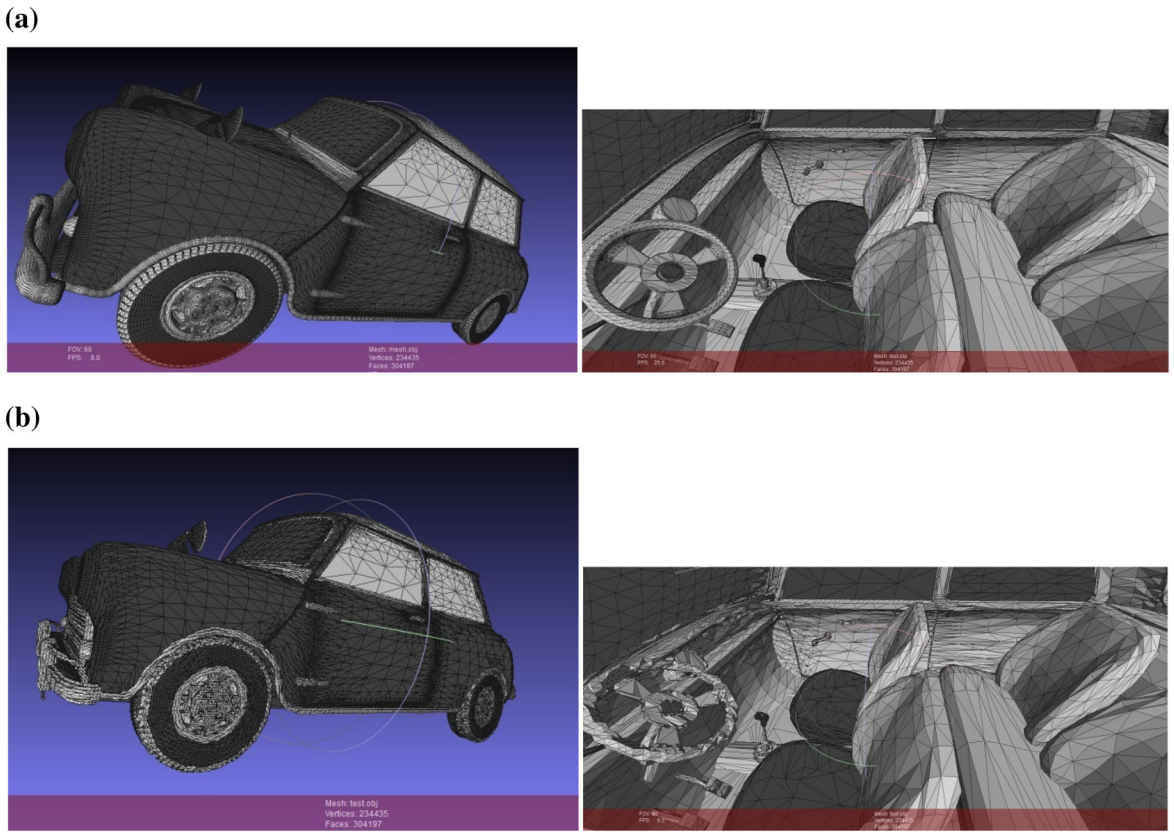


Fig. 13 Decompressed 3D Car image at compression size: 1.34 MB and 917 KB **a** (*left*) 3D mesh Car object, (*right*) 3D mesh Inside the Car, shows the details of the object, at compresses size: 1.34 MB, **b** 3D mesh Car shows the details of the object, at compresses size: 917 KB, the image shows the vertices for some objects move slightly away from their original positions

4 Decompression Algorithm by Parallel Threads on Blocks of Data

The GM-Algorithm described in Sect. 2 compresses each vertex data (X, Y, Z) to a single integer value by three different weights (W_1, W_2, W_3). Here we described how to recover the original data. To this purpose we designed a parallel algorithm named *Parallel Fast-Matching-Search Algorithm* (*Parallel-FMS*). The header of the compressed file contains information about the compressed data pertaining the weights and *DS* (the Domain Searchfor each block of vertex as in Fig. 2) followed by streams of compressed encoded data (*Encoded Data*). The Parallel-FMS algorithm picks up in turn *each k*-encoded data to reconstruct k blocks of vertices (X, Y, Z). The Parallel-FMS is based on a binary search algorithm illustrated through the following steps (A) and (B):

A. Initially, the Domain Sarch (*DS*) is copied to three separate arrays to estimate X, Y and Z respectively. The searching algorithm computes all possible combinations of X with W_1, Y with $W_2$and Z with W_3 that yield a result R-Array. As a means of an example consider that $DS(1) = [X1\ X2\ X3]$, $DS(2) = [Y1\ Y2\ Y3]$ and $DS(3) = [Z1\ Z2\ Z3]$. Then, according to Eq. (1) these represent V_X, V_Y and V_Z respectively. The equation is executed 27 times to build the R-Array, as described in Fig. 6a. The match indicates that the unique combination of X, Y and Z represents the orginal vertex block.
B. A*binary search algorithm* [5] is used to recover an item in the array. Here we designed a parallel binary search algorithm consisting of k-binary search algorithms working in parallel to reconstruct $k$block of vertices in the list of vertices, as

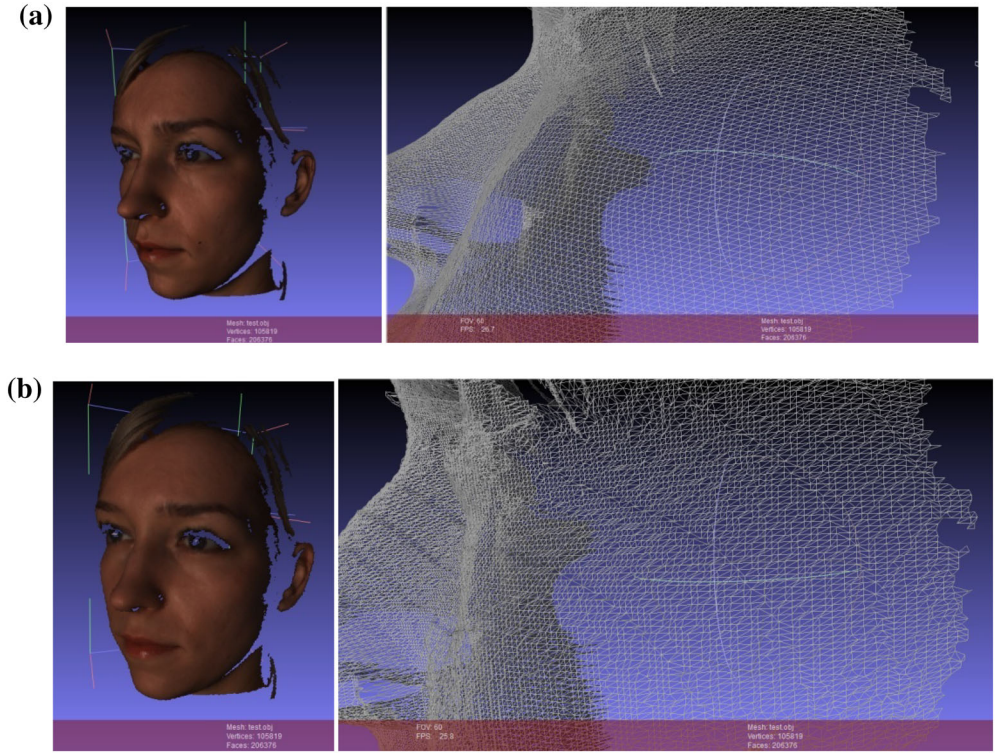


Fig. 14 Decompressed 3D Face2 image at compression size: 378 and 290 KB **a** (*left*) 3D texture Face2 object, (*right*) 3D mesh Face2, shows the details of the object, at compressed size: 378 KB, **b** 3D texture Face2 with 3D mesh shows the details of the 3D Face2, at compressed size: 290 KB, the image shows the vertices slightly move away from their original positions

shown in Fig. 6b. In each step in the *k*-binary search algorithm compares *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 vertices values are returned. To illustrate our decompression algorithm, the compressed samples in Fig. 2 (by our GM-Algorithm) can be used by our decompression algorithm to reconstruct X, Y and Z values as shown in Fig. 7.

Once vertices are recovered, we turn our attention to decoding triangle faces and vertex texture coordinates. The differential process of Eq. (2) is reversed by addition such that the encoded values in each triangle face and texture coordinates 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 [21].

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

where $i = m, (m-1), (m-2), (m-3), \ldots, 2$.

5 Experimental Results

The experimental results described here were implemented in MATLAB R2013a and Visual C++ 2008 running on an AMD Quad-Core microprocessor. We describe the results in two parts: first, we apply the compression and decompression algorithms to 3D data object generated by the GMPR 3D surface scanner [13, 17, 24]. The principle of operation of GMPR 3D

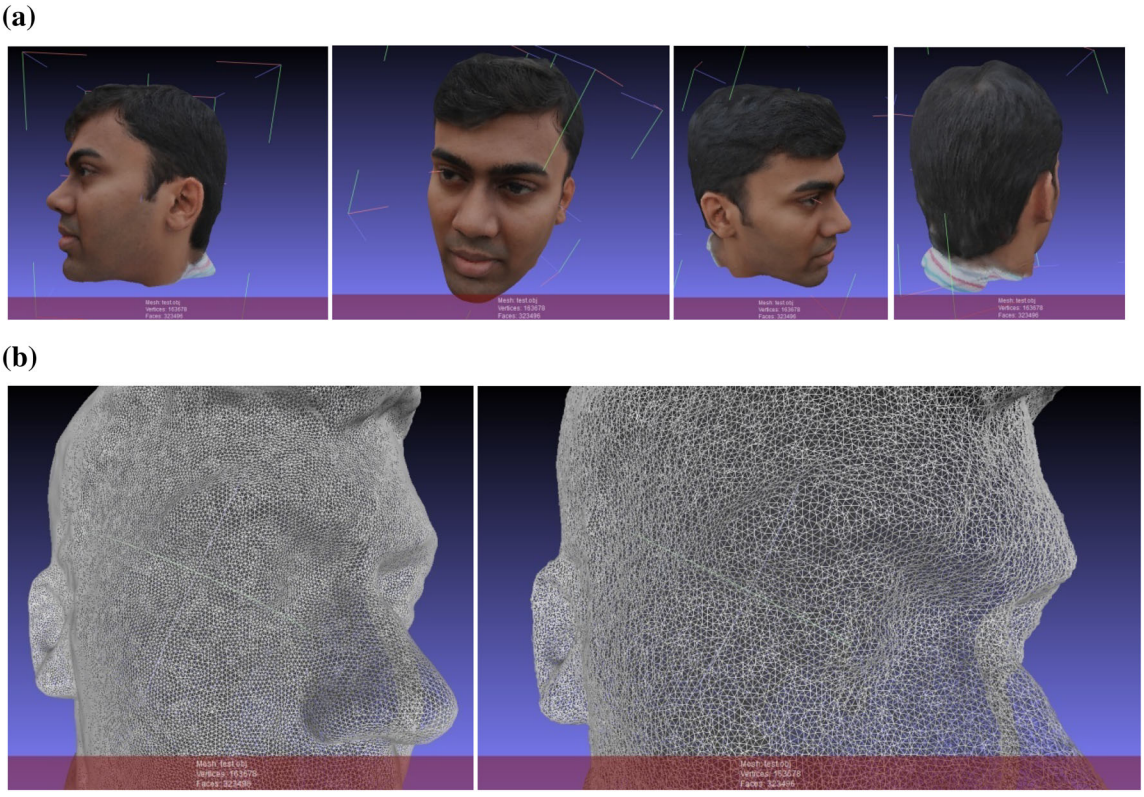


Fig. 15 Decompressed 3D Face3 image at compression size: 6.42 and 6.21 MB **a** Full 3D texture Face3 object, shows the details of the object, at compresses size: 6.42 MB, **b** (*left*) 3D mesh shows the details of the 3D Face3, at compresses size: 6.42 MB, (*right*) 3D mesh at compressed size 6.21 MB, the vertices slightly move away from their original positions

Table 2 Compression and Decompression execution time by the proposed method

3D image Name	Original file size (MB)	Shift value	Estimated compression Time (s)		Estimated decompression Time (s)	
			Vertices (X, Y, Z)	Triangle faces and texture	Vertices (X, Y, Z)	Triangle faces and texture
Corner	2.91	1	1.5	34.8	2.25	40.2
		10	1.9	34.4	2.35	39.1
Metal	5.34	10	2.8	63.5	5.1	70
		50	3	64.48	5.4	70
Face1	4.7	10	2.2	54.1	4.42	65
		20	2.2	56.2	4.39	66
Angel	23	10	210.4	370.8	316.8	400
		50	420.4	355.98	828.89	409
Car	14.1	1	40	176.6	71.48	210
		10	180.2	169.9	338.2	200
Face2	13.3	2	25.5	160.4	47.89	200
		10	10	162.46	18.5	200
Face3	55	1000	1.554e+03	1.4239e+03	2.954e+03	1.622e+03
		5000	1.4412e+03	1.3905e+03	2.8824e+03	1.572e+03

Table 3 Our approach compared with other encoding 3D data format

Method	3D Object file						
	Corner (Original file size 2.91 MB)	Metal (Original file size 5.34 MB)	Face1 (Original file size 4.7 MB)	Angel (Original file size 23.5 MB)	Car (Original file size 14.1 MB)	Face2 (Original file size 13.3 MB)	Face3 (Original file size 55 MB)
Proposed method							
Compressed file size	32.5 KB	53.2 KB	40.2 KB	2.02 MB	1.34 MB	378 KB	6.42 MB
Compression ratio (%)	98.9	99	99.1	91	90.4	97.2	88.3
MATALB format							
Compressed file size	850 KB	1.53 MB	1.32 MB	5.31 MB	4.4 MB	4.04 MB	11.9 MB
Compression ratio (%)	71.4	71.3	71.9	77.4	68.7	69.6	78.3
VRML format							
Compressed file size	2.39 MB	4.43 MB	3.9 MB	23.2 MB	13.7 MB	9.19 MB	17.5 MB
Compression ratio (%)	17.8	17	17	1.27	2.9	30.9	68
STL format							
Compressed file size	2.23 MB	4.11 MB	3.62 MB	29.2 MB	14.5 MB	9.84 MB	15.4 MB
Compression ratio (%)	6.69	23	22.9	NON	NON	26	72
OpenCTM format							
Compressed file size	145 KB	220 KB	176 KB	1.92 MB	1.36 MB	808 KB	789 KB
Compression ratio (%)	95	95.9	96.3	91.6	90.3	94	98.5

surface scanning is to project patterns of light onto the target surface whose image is recorded by a camera. The shape of the captured pattern is combined with the spatial relationship between the light source and the camera, to determine the 3D position of the surface along the pattern [14] as shown in Fig. 8a. Second, we apply the method to general 3D object data (i.e. 3D object generated by 3Dmax, CAD/CAM, 3D camera or other devices/software) as shown in Fig. 8b.

Table 1 show our compression algorithm applied on each 3D object file, and Fig. 9, 10, 11, 12, 13, 14, 15 shows the visual properties of the decompressed 3D object data for 3D images respectively. Additionally, RMSE are used to compare between 3D original object file content (geometric (X, Y, Z) and (u, v) texture coordinates) and the recovered 3D object file content. Root Mean Square Error (RMSE) is used to refer to image quality mathematically [12, 19]; it can be calculated by computing the differences between the decompressed 3D object and the original 3D object (as shown in Table 2).

Table 3 shows the comparison between the 3D file formats VRML, OpenCTM and STL. The file format referred to as MATLAB format contains geometric, texture and triangle facesas lossless data, and using MATLAB language to read/write 3D data. We investigate this format obtaining compression ratios of over 50 % for most 3D OBJ files. Our approach used unique algorithms to compress 3D OBJ files leading to compression rates of over 98 % in the best case and 85 % for the worst case; the ratio is dependent on the triangle face details.

6 Conclusion

This research has presented and demonstrated a new method for 3D data compression/reconstruction and compared the quality of compression through 3D reconstruction, 3D RMSE and the perceived quality of the 3D visualisation. The method is based on converting geometric values to a stream of encoded integer data by the GM-algorithm. Connectivity data are partitioned into groups where each group is addressed by arithmetic coding for lossless compression. The results demonstrate that our approach yields high quality 3D compression at higher compression ratios compared with other 3D data formats. Although the lossy compression controlled by the shift parameter introduces small reconstruction errors as observed by RMSE and detailed visualization, the methods provide high compression ratios with high quality data and it is appropriate for most applications, including demanding applications such as 3D face recognition. A disadvantage of the method is the large number of steps for compression and decompression. The main bottleneck is related to the Parallel-FMS algorithm leading to increased execution time through a binary search method. Methods to speed up execution are being investigated and will be reported in the near future.

References

1. Alliez, P. & Gotsman, C. (2003). *Recent advances in compression of 3D meshes*. Inria Sophia Antipolis Research Report 4966, Oct 2003 pp. 26
2. DCT (2012). 3D compression technologies (3DCT), www.3dcompress.com/web/default.asp. Accessed Oct 2012.
3. Deering, M. (1995). Geometry compression. In *SIGGRAPH 95 Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*.
4. Deering, M. (1995). Geometry compression. In *Computer Graphics* (*SIGGRAPH'95 Proceedings*), pp. 13–20.
5. Donald, K. (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.
6. Evans, F., Skiena, S.S., & Varshney, A. (1996). *Optimizing triangle strips for fast rendering*. In IEEE visualization. IEEE, October 1996. ISBN 0-89791-864-9.
7. Gurung, T., Luffel, M., Lindstrom, P., & Rossignac, J. (2013). Zipper: A compact connectivity data structure for triangle meshes. *Journal of Computer –Aided Design, 45*(2), 262–269.
8. Hollinger, S.Q., Williams A.B., & Manak, D. (1998). 3D data compression of hyper spectral imagery using vector quantization with NDVI-based multiple codebooks, *IEEE International Geosciences and Remote Sensing Symposium IGARSS'98*, Vol. 5, 2680–2684.
9. Murdock, K. L. (2008). *3DS max 2008 Bible* (1st ed.). Indianapolis, Indiana: Wiley Publishing, Inc. ISBN 9780470417584.
10. Neider, J., Davis, T., & Woo M. (1997). *OpenGL programming guide—The official guide to learning openGL, Version 1.1*. Addison-Wesley, Reading, MA, USA.
11. Peng, J., Kim, C. S., & Kuo, C. C. (2005). Technologies for 3D mesh compression: A survey. *Journal of Visual Communication and Image Representation, 16*(6), 688–733.
12. Richardson, I. E. G. (2002). *Video codec design*. USA: Wiley.
13. Rodrigues, M., Kormann, M., Schuhler, C., & Tomek, P. (2013b) Robot trajectory planning using OLP and structured light 3D machine vision. *Lecture notes in Computer Science Part II. LCNS Springer*, *Heidelberg*, Vol. 8034, pp.244–253

14. Rodrigues, M., Kormann, M., Schuhler C., & Tomek, P. (2013c). Structured light techniques for 3D surface reconstruction in robotic tasks.In *Proceedings of the 8th International Conference on Computer Recognition Systems CORES 2013*, Springer, pp. 805–814.
15. Rodrigues, M., Kormann, M., Schuhler C., & Tomek, P. (2013d). An intelligent real time 3D vision system for robotic welding tasks. In: *Mechatronics and its applications. IEEE Xplore*, pp. 1–6.
16. Rodrigues, M., Osman, A., & Robinson, A. (2011). Efficient 3D data compression through parameterization of free-form surface patches. In: *Signal Process and Multimedia Applications* (*SIGMAP*), *Proceedings of the* (*2010*) *International Conference on. IEEE*, 130–135.
17. Rodrigues, M., Osman, A., & Robinson, A. (2013). Partial differential equations for 3D data compression and reconstruction. *Journal Advances in Dynamical Systems and Applications, 8*(2), 303–315.
18. Rossignac, J. (2003). *3D mesh compression*. College of Computing and GVU Center Georgia institute of Technology Report, June 2003 Chapter Five-Visualization HandBook.
19. Sayood, K. (2000). *Introduction to Data Compression* (2nd ed.). USA: Academic Press, Morgan Kaufman Publishers.
20. Shikhare, D., Babji, S.V., & Mudur, S.P. (2002). Compression techniques for distributed use of 3D data: an emerging media type on the internet. In *15th International Conference on Computer Communication*, *India*, pp. 676–696.
21. Siddeq, M. M., & Rodrigues, M. E. (2014). A novel image compression algorithm for high resolution 3D reconstruction, *3D Research. Springer*,*5* (2). Doi:10.1007/s13319-014-0007-6.
22. Szymczak, A., King, D., & Rossignac, J. (2000). An edge-breaker-based efficient compression scheme for regular meshes. In *12th Canadian Conference on Computational Geometry*, pp. 257–265.
23. Taubin, G., Horn, W., Lazarus, F., Rossignac, J. (1998). Geometry coding and VRML, *Proceedings of The IEEE*, *86*(6).
24. Taubin, G., & Rossignac, J. (1996). *Geometric compression through topological surgery*. Technical report, Yorktown Heights, NY 10598, Jan 1996. IBM Research Report RC 20340.
25. Taubin, G., & Rossignac, J. (1998). Geometric compression through topological surgery. *ACM Transactions on Graphics, 17*(2), 84–115.