Polygonization of 3D Objects using Norm Similarity

Somrita Saha
Department of Information Technology
Indian Institute of Engineering Science and Technology
Shibpur, India, Email: somrita.besu@gmail.com

Arindam Biswas
Department of Information Technology
Indian Institute of Engineering Science and Technology
Shibpur, India, Email: barindam@gmail.com

***Abstract*—This work proposes a method of polygonization of the surface of triangulated 3D digital objects by merging the edge-adjacent face triangles having similar norms. Least square method is applied to the boundary vertices of the set of the merged face triangles, which yields a new norm optimizing the norms of the individual face triangles constituting the merged set. The coordinates of the boundary vertices are recalculated with respect to this new norm. Finally, all the instances of each vertex is averaged to obtain the final coordinates. In this process, many of the vertices of the original input object are dropped, giving rise to a number of polygons describing the object, which is substantially less than the number of original face triangles. Thus, a considerable amount of compression is achieved, Also, polygons representing the object can reveal its structure and shape and are amenable to improved shape analysis.**

***Contribution*: It proposes an algorithm to polygonize 3D digital object surface achieving significant compression.**

***Index Terms*—Polygonization, Mesh Simplification, Approximations, Data Compression**

I. Introduction

In the last three decades significant effort has been put to the concept of mesh simplification. The field presently has multiple options for the method of mesh simplification, from which a particular one can be chosen based on the type of input and the required level of details in the output image. These algorithms can be largely categorized into the given classes based on Vertex Clustering, Incremental Decimation, Mesh Re-tiling, and Mesh Optimization.

Turk [1] described a surface re-tiling algorithm where newly recalculated vertices are contoured on the original mesh following topological and geometry constraints. A general algorithm for mesh simplification based on vertex clustering is proposed in [2]. Vertex decimation was proposed by Schroeder [3]. Kalvin and Taylor [4] presented another efficient algorithm based on merging adjacent quasi-coplanar face triangles to form polygonal faces, simplifying the boundaries of these faces, and finally retriangulating the polygons themselves with a smaller number of primitives. The time complexity of this model is O(N), where N is the number of polygon faces in the original mesh. Ronfard and Rossignac [5] came up with another fast method for approximating surface deviation which computes surface deviation error for each vertex as a sum of squared distance to a set of planes. Garland and Heckbert proposed some improvement over [5]. An algorithm based on vertex removal operation is proposed in [6]. Hoppe proposed another algorithm based on mesh simplification operation 'edge collapse' [7]. Triangle Decimation based approach was presented in [8]. Some improvement over [5] is suggested by vertex pair contraction [9]. Zhu et al. presented an algorithm on mesh simplification based on vertex importance value and iterative edge collapse [10]. Feature extraction based mesh simplification, where edge-weight based edge-collapse is suggested, was proposed in [11]. Another mesh simplification algorithm based on edge collapse combines surface curvature and quadric error metric to decide the position of a new vertex [12]. [13] suggests triangle contraction to simplify the mesh, and two tolerance areas with respect to the reference mesh to preserve the surface characteristics. [14] proposed an algorithm which is an extended quadric-error metrics based on the geometric and visual characteristics of the object. Another method based on half-edge collapsed scheme is presented in [15] considering the length of edges and the distance between two adjacent triangle normal vectors to validate the decimation of a vertex. Another method computes a view-independent hybrid saliency for each vertex of the input mesh and then drops the low hybrid saliency vertices for mesh simplification [16]. [17] proposed a mesh simplification algorithm based on reverse interpolation loop subdivision (RILSP). Our work is, in a sense, analogous to constructive solid geometry, here, the primitive being the triangles of a triangulated object which are combined to produce polygonal surfaces describing the object.

Most of the above mentioned mesh simplification algorithms' final outputs are face triangles. But, sometimes polygons are preferable as the final output over triangles as these polygonal patches determine stretches of the seemingly planar regions of the object surface. The proposed algorithm in this document is motivated by this observation. In addition to that, it achieves significant rate of compression, requiring less space for storage and less bandwidth for transmission.

The paper is organized as follows. First we present the required definitions related to this work, followed by the simplification algorithm and complexity analysis. Finally, a discussion on the experimental outputs are given.

II. Definitions

Definition 2.1: Mesh Simplification: It is the process by which a large and complicated digital object mesh is simplified to an object with lesser number of vertices and faces, while maintaining minimum distortion of the visual quality.

978-1-7281-9331-1/20/$31.00 ©2020 IEEE

Definition 2.2: Coplanar faces: Two faces are called coplanar if they lie on the same geometric plane, hence the dot product of their norms is the product of their magnitudes.

Let $\vec{a}$ and $\vec{b}$ be the norms of two adjacent face triangles and the dihedral angle subtended by them is θ. Then, the dot product of the normal vectors can be expressed as

$$\vec{a} \cdot \vec{b} = \mid a \mid \cdot \mid b \mid \cdot \cos\theta$$

The sharper the dihedral angle is, the higher the dot product is. When θ is $0°$, the dot product equals the product of the magnitudes of $\vec{a}$ and $\vec{b}$, therefore, the planes being coplanar.

Definition 2.3: Quasi-coplanar faces: Two faces are called quasi-coplanar if cosine of the dihedral angle θ is less than a predefined threshold ϵ (called as ϵ-quasi-coplanar faces).

The data structure used is Doubly Connected Edge List (DCEL). Apart from the basic structure of DCEL, some additional information are stored as a part of the DCEL for this work. For the object vertex, whether it is a part of any polygon's boundary, number of polygon boundaries it is part of, the coordinate values of those incarnations, and the final value of the vertex coordinates are stored. For the edge object, additional information on destination vertex, twin half edge, new previous half edge, new next half edge, whether the edge is dropped, and the polygon it is a part of are maintained. Finally, for the face objects, additional information on its norm, color, area, and the base face corresponding to the polygon it is part of are stored. Moreover, another object polygon is maintained for all the polygons to be generated, which contains total number of faces in the polygon, total number of vertices on the polygon boundary, norm, colour code, and area of the polygon, first edge on the boundary, base face of the polygon, and the outer polygon.

III. Polygonization procedure

The 3D objects are generally represented as triangulated objects. While triangles, being the polygon with the smallest number of sides and the most granular representation, we observe that there may be adjacent triangles with a small difference (below a threshold) in their norms and these can be merged together to form a polygon which represents the original surface with a minimal change and fewer vertices.

The procedure starts with the first face in face list, assigned as the base face. Then the edge adjacent faces with respect to the base triangle are considered. Each face should have exactly three edge adjacent faces, as we have considered watertight models only. Norms of these three faces are compared with the base face. If the cosine of the dihedral angle between two faces is above a predefined threshold ϵ, then the concerned faces are said to be similar with respect to their normal vectors. Adjacent faces are added to the queue for checking their norm similarity, if a face is similar then it is merged. This process is repeated till the queue is exhausted. Once the queue is empty, we get the ϵ-quasi-coplanar polygon, with the first face of the face list being the base face. For the next set, the next unvisited face from the face list is considered as the base face and its adjacent face triangles are checked for similarity of norm. This process is repeated till the last unvisited face of the face list. Now, we have a list of quasi-coplanar polygons. These sets are called ϵ-quasi-coplanar polygons, which are then converted to polygons by assigning them a uniform norm using Least Sqaure Method [18].

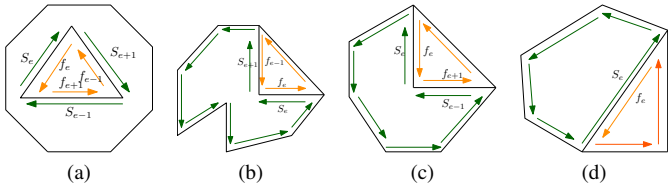


Fig. 1: Merging of a triangle with another triangle of the polygon ((a) Three common pairs of half-edges between S and f, (b) - (c) Two common pairs of half-edges between S and f, (d) One common pair of half-edges between S and f)

A. Merging of face triangles based on the similarity of norms

The DCEL data structure changes when merging a triangle with an existing set of triangles. A number of pointers are modified for the edges involved. The new triangle which is merged has one or more edge(s) adjacent to the existing set of triangles. These edges and their twins are dropped. For the next and previous edges, either old pointers are retained or new pointers are assigned for them, based on the status of the next and previous edges of the dropped edge and its twin. It is shown in Fig. 1 and explained in Sec. IV.

B. Recomputation of vertex coordinates using Averaging

After merging the ϵ-quasi-coplanar faces, we get a pseudo-polygon in 3-dimension whose boundary vertices are not exactly on the same plane as the merged faces may differ by ϵ, with respect to their norms. In order to fit these boundary vertices into a single polygon, a Least Square Method is applied which yields a single new norm, based on which, the boundary vertices are recalculated. It may be noted that a vertex may belong to more than one such merged polygon. In each instance, a vertex, when belongs to a merged polygon, it is a different incarnation. Thus, if a given vertex belongs to m different merged polygons, it has m different incarnations. Now the final coordinate values are assigned by averaging the coordinates of all these m incarnations.

IV. Algorithm 3D-Surface-Polygonization

The algorithm for 3D surface polygonization is based on the Breadth First Search (BFS) principle. It takes the triangulated 3D object, D (in the form of DCEL), and generates the output object with polygonized surfaces. The initilizations are done in Steps. 1-7. The linked list L, containing one boundary edge of the set of triangles and the set S to contain the ϵ-quasi-coplanar triangles are initialised to $\emptyset$. Color of all the faces are also initialised to NULL (Step. 2). First face from the face list of D is assigned as the base face, b, and it is added to S (Steps. 4 & 5). Color of b is initialised to $c[i]$, where c is the color array for newly constructed polygons. All the three edge adjacent faces of b are enqueued to Q (Step. 7). Then, in the **while** loop (Steps 8-24) all the faces of the face list of D which

are not yet polygonized are considered. The BFS is excuted in the loop (Steps 9-24). A face, t, is dequeued from Q (Step. 10), if t has not yet been assigned a color, i.e., it has not been polygonized, then we check if it is coplanar with b (Step 12). If yes, then in Steps 13-16, t is merged with S and an edge of S is stored in $L[i]$ as an anchor edge to find the polygon P_i later using the DCEL. Color of t is assigned as $c[i]$. The edge-adjacent faces of t are queued in Q (Step 16). If t has already been assigned a color (Step 17) and the color is same as of b (Steps (18-20)), t is merged with S and $L[i]$ stores a boundary edge of P_i. If Q is empty in Step 9, then BFS corresponding to b has been completed, i.e., no more faces are similar to b, w.r.t. norm. In Step 21, a new face from the face list is considered as the new base face, the corresponding color is assigned (Step 23), and the edge-adjacent faces are enqueued in Q for the next BFS traversal.

After all the faces are checked in Steps (8-24), in the last **while** loop, (Steps (26-29)), for each the polygon, P_i, boundary vertices are extracted (Step 27). Least square method is applied to compute the unique norm of P_i (Step 28). In Step 29, each vertex coordinate is recomputed using the unique norm. Finally, in Step 30, for all the vertices, coordinates of different incarnations are averaged to get the final coordinate. D contains polygonized object.

Procedure MERGE: While merging a face f with the polygon S, f may have one or more edges common with an edge of S. The common edges (obviously along with the twins) are deleted and the previous and the next pointers are adjusted accordingly. The different possible cases are depicted in Fig. 1 and explained below.

• First, the triangle f is surrounded by S, i.e., all the *three* edges of f are adjacent to three edges if S. In this case, three edges of f and their twins are simply dropped. Hence, they are excluded from further consideration while visiting the boundary of the polygons, as shown in Fig. 1(a).

• The second possibility is that triangle f is attached to the ϵ-quasi-coplanar set S through *two* adjacent edges. These two edges and their twins are dropped and the next and previous of the previous and next edges respectively, are reassigned accordingly, as shown in Fig. 1(b) and 1(c).

• Similarly, if the triangle is attached to the set S through only one edge, then the half-edge and its twin are dropped and the next and previous of their previous and next edges, respectively, are reassigned according to Fig. 1(d).

Thus, the polygonized object is generated, which is represented by the modified DCEL, D. After the while loop (Steps (8-24)) we can get the boundary vertices of the polygons. To determine the unique norm of the polygon, Least Square Method is applied which is governed by

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = (A^T A)^{-1} A^T B \tag{1}$$

where, A and B are $[n_k \times 3]$ and $[n_k \times 1]$ matrices respectively, and $< a, b, c >$ is the newly calculated norm. Here, n_k is the total number of vertices on the boundary of the polygon k. Recalculated value of each vertex is stored.

Algorithm 1: 3D-SURFACE-POLYGONIZATION

Input: Triangulated object in the form of DCEL(D)
Output: Polygonized object

1 $L \leftarrow \emptyset$, $S \leftarrow \emptyset$
2 Initialise color of all faces to $\emptyset$
3 $i = 0$
4 $b \leftarrow$ first face from the face list of D
5 $S \leftarrow b$ /*add b to the set S*/
6 $color[b] \leftarrow c[i]$
7 ENQUEUE (Q, edge-adjacent faces of b)
8 **while** $(facelist \neq \emptyset)$ **do**
9 **while** $(Q \neq \emptyset)$ **do**
10 $t \leftarrow$ DEQUEUE(Q)
11 **if** $(color[t] = \emptyset)$ **then**
12 **if** (t coplanar with b) **then**
13 merge t with S
14 $L[i] \leftarrow$ boundary edge of S
15 $color[t] \leftarrow c[i]$
16 ENQUEUE(Q, edge-adjacent faces of t)
17 **else if** $(color[t] \neq \emptyset)$ **then**
18 **if** $(color[t] = c[i])$ **then**
19 merge t with S
20 $L[i] \leftarrow$ boundary edge of S
21 $t \leftarrow$ next unvisited face from face list, $b \leftarrow t$
22 $i \leftarrow i + 1$, $S \leftarrow \emptyset$
23 $S \leftarrow t$, $color[t] \leftarrow c[i]$
24 ENQUEUE (Q, edge-adjacent faces of t)
25 $i = 0$
26 **while** $L[i] \neq \emptyset$ **do**
27 Extract the boundary of the polygon P_i from the boundary edge $L[i]$
28 Find unique norm of P_i by Least Square method
29 Recalculate the vertex coordinates of P_i based on unique norm
30 For all vertices, average the coordinates of different incarnations to get the final vertex coordinates.

V. COMPLEXITY

A. Time complexity

In the algorithm for polygonization, each face triangle of the original object is enqueued a maximum of three times as the triangle is edge-adjacent to only three other triangles. In the loop Steps (8-24), the algorithm takes constant amount of time to check the coplanarity and assign colors etc. The merging operation also takes constant amount of time as it involves manipulation of edges which is linear, as explained in the method of MERGE. In Step 28, we apply the Least Square Method, whose complexity is explained as follows.

As per Eqn. 1 the order of matrix A is $[n_k \times 3]$ and so the computation of $A^T A$ requires $9n_k$ multiplications, where n_k is the number of vertices of the polygon k. Inverse operation of this constant order matrix is also of constant order and is

$O(3^2)$, i.e., $O(1)$. Now, multiplication of A^T and B requires $3n_k$ multiplications and the order of the product is $[3 \times 1]$. Hence, the order of the final resultant matrix is $[3 \times 1]$ and requires a total of $12n_k$ multiplications, where n_k is the number of vertices on the boundary of the polygon k, which is less than n. The Least Square Method is called for all polygons and thus the complexity is of the order of the sum of the vertices of the polygons. $O(\sum_1^k n_i)$ is definitely less than the total number of the triangles of the original object, i.e. $O(\sum n_k) < O(n)$, as some of the quasi-coplanar triangles are merged. Thus, the complexity is bounded by $O(n)$.

B. Space Complexity

As far as the space complexity is concerned, the object is represented by DCEL data structure for subsequent iterations; It is augmented with some fields, like the color of the face, etc., keeping the size of DCEL of $O(n)$. A new structure for polygon is added. The edge structure is modified to hold three new attributes. The vertex and face structures are also modified to hold some more necessary attributes for implementation. For running the above algorithm, the object is represented in the DCEL data structure which will take a size of $O(n)$. This DCEL data structure is accordingly modified during the merging process. After averaging the different instances of the coordinates, the final boundary vertices are computed. A separate linked list L is maintained, each node of which stores an edge (acts as the starting edge for tracing the polygon using the DCEL) of each resulting polygon. The size of L is less than $O(n)$. Hence, the final space complexity of the algorithm POLYGONIZE is $O(n)$.

Thus, we have presented an algorithm POLYGONIZE which gives a polygonal representation with linear time complexity as well as linear space complexity.

VI. RESULTS AND ANALYSIS

The proposed algorithm in this work is implemented using the programming language C. Table I shows the results for four different objects and the corresponding change in surface area, change in volume, the total number of boundary vertices, total number of polygons, and also the compression ratio for four different thresholds for each object. It may be noted here that the information in the row for $\epsilon = 1$ is essentially that of the original object. Hence, $\triangle V$ and $\triangle A$ is computed with respect to the original object information (corresponding to $\epsilon = 1$). Four different thresholds have been used, which are 1.000, 0.985, 0.945, and 0.905. The four objects in table are organized in the descending order of their size.

It is easy to see that for a triangulated object with n triangles, it will require $3 \times n$ vertex indices to describe the objects. After polygonization, as a number of neighbouring triangles with similar norms are merged into a polygon, the number of vertices on the boundary of the polygon will be more than three. Here, we define a parameter polygon space which will also relate to the compression ratio. Let the resultant polygonized object be described by a set of n polygons $P = P_1, P_2, ..., P_n$. Size of a polygon, S_i, is defined as the number of vertices on the boundary of the polygon. Then, in order to describe the object, we need a total of $N = \sum_{i=1}^{n} S_i$ vertex indices, which is called as polygon space, P_s.

Compression Ratio: The compression ratio, $\mathcal{C}$, is given by

$$\mathcal{C} = \frac{v_o \times 3 \times (size\_float) + P_s \times (size\_integer)}{v_i \times 3 \times (size\_float) + f_i \times 3 \times (size\_integer)} \quad (2)$$

where, v_i =Total number of vertices in the input object, f_i =Total number of faces in the input object, v_o =Total number of vertices in the output object, P_s =Polygon space.

The Eqn. 2 computes the compression achieved after polygonization. If the system uses 2 bytes to represent an integer and 4 bytes for a float data, then a vertex with three coordinates, each requiring 4 bytes, will require 12 bytes in total. For faces, input object always has triangulated faces. Hence the space required for face information will be *total number of faces* $\times 3$ (*for three vertices*) $\times$ *size of an integer*. But for polygonized faces, the total number of vertices on the boundary of faces are counted. Size of data to represent face information will be twice this count. The total number of polygons varies with the threshold. If the threshold is very high then the total number polygons to represent the object will be more and the surface area of the polygon will be less. The extreme case is when the threshold is 1.000, when almost each face triangle of the input object becomes a single polygon. On the other hand, as the threshold is reduced, the number of polygons also decreases and the size of the area of polygonized face increases.

For example, if the object elephant is considered, for threshold 1.000, the number of polygons is maximum, which equals the number of face triangles in the input object. The number of vertices are also same as the input object. But, as the threshold is reduced the polygons become larger and the total number of polygons and the boundary vertices reduce significantly, as seen in table I. If the threshold is reduced from 1.000 to 0.985, the total number of polygons reduces from 53506 to 6370 and the total number of boundary vertices reduces from 26755 to 18526. This gives a compression gain of 50% (tables I and IV). For camel and tiger, polygonization at threshold 0.985 achieves compression 61% and 59%. Polygonization below the threshold 0.985, e.g., at thresholds 0.945 and 0.905 results in a high rate of compression. But also, very dissimilar face triangles get included in the same polygon. So the visual quality is compromised. This can be understood from the table I and the figures given in tables II, III, IV, and V.

A comparison of the compressions achieved for the relevant objects for different thresholds is depicted in Fig. 2. It is evident from this graph, that the more the threshold is reduced, the less size is needed to represent the polygonized object, compared to the original input object. Also, there are Fig. 3 and Fig. 4, containing the plots of change in area and volume of the original objects with respect to varying thresholds. Fig. 2 reveals another characteristic of the polygonization process, which is the improvement of the compression rate with the increasing size of the object. For example, more number of face triangles are combined to make a single polygon of bunny,

than that of camel. Hence, the compression is more. In fact, the rate of compression improves gradually with increasing size, for the mentioned objects. It may also be noted that the rate of compression also depends on the structural shape of the object. An object, with bigger stretches of flat surface areas, has a better compression rate compared to an object which has smaller stretches and fast-changing norm of face triangles. Due to this behaviour, with varying threshold, the area or volume may both increase or decrease for a particular object. If some faces, belonging to a concave area, combine to a single polygon, the volume may increase. On the contrary, the volume may decrease for the faces pertaining to a convex area. Hence, the total volume of the polygonized object may increase or decrease with varying threshold. Similarly, it is true for the area too. Fig. 3 and Fig. 4 shows a different trend of the plot for bunny, compared to the other objects. It happens because of bunny's shape, which has many small concave or convex regions with sharp change in norms.

In general, a threshold of 0.985 results in an acceptable optimization of the compression and the visual quality loss. Hence, Fig. 5 is the graph for threshold 0.985, where the runtimes of different objects are plotted with respect to the size(number of face triangles in 1000s) of the objects. This graph tends to linearity, which is acceptable for the complexity $O(n)$, of the algorithm 1.

The process polygonization combines the ϵ-quasi-coplanar face triangles to a single polygon. Therefore, the finer details of the original input object are compromised to some extent. The more the threshold is reduced, the more the original object is distorted. So, an optimized threshold is chosen for which a trade-off can be achieved when the polygonization yields minimum visual distortion with maximum compression. Hence, there is minimum difficulty to interrogate the surface of the object.

This technique of polygonization can be used for shape characterization by storing the polygons in decreasing order starting with the largest polygon for a given object. The shape of the polygons as well as its size can identify a particular object. A shape similarity measure can be devised by matching the polygons of a given a query object with the polygons corresponding to a given object in a database with respect to the shape and size. The same can be used for shape retrieval purpose also.

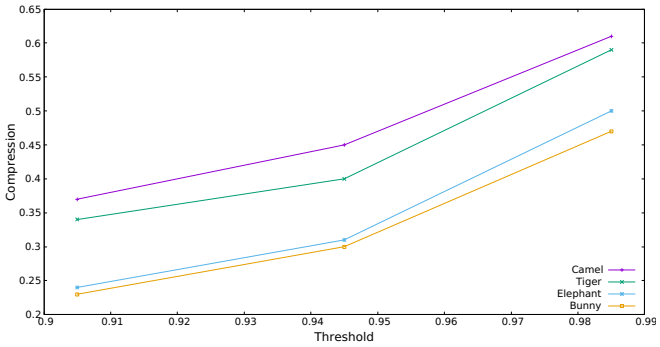


Fig. 2: Compressions for varying thresholds

TABLE I:

Results corresponding to different objects. The different columns shows the following: ϵ:threshold, A:area of the polygonized object, $\%\triangle A$: change in area w.r.t. the input object, V:volume of the object, $\%\triangle V$: change in volume after polygonization, $\#v$:total number of vertices, $\#P$:total number of polygonal faces, P_s:Polygon Space, $\mathcal{C}$:compression ratio. Note that for $\epsilon = 1$, it is the original triangulated input object.

Object	ϵ	A	$\%\triangle A$	V	$\%\triangle V$	#v	#P	P_s	$\mathcal{C}$
Camel	1.000	7574.37	0.00	20 664.85	0.00	12460	24916	74748	1.00
	0.985	7533.37	0.54	20 252.94	1.99	10725	2974	27382	0.61
	0.945	7507.09	0.89	19 473.88	5.76	8170	1152	18570	0.45
	0.905	7446.47	1.69	19 534.10	5.47	6841	723	14857	0.37
Tiger	1.000	9044.63	0.00	34 098.76	0.00	12898	25800	38694	1.00
	0.985	9019.00	0.28	34 348.58	0.73	10191	4807	29896	0.59
	0.945	8980.55	0.71	32 504.17	4.68	7469	1555	17725	0.40
	0.905	8826.66	2.41	32 948.22	3.37	6409	933	14294	0.34
Elephant	1.000	8109.67	0.00	25 167.53	0.00	26755	53506	160518	1.00
	0.985	8119.35	0.12	24 892.66	1.09	18526	6370	49679	0.50
	0.945	8138.97	0.36	25 348.09	0.72	11838	2144	27672	0.31
	0.905	8064.14	0.56	25 049.25	0.47	9469	1261	21168	0.24
Bunny	1.000	58 229.13	0.00	754 903.01	0.00	34834	69664	104502	1.00
	0.985	58 627.60	0.68	746 858.36	1.07	22814	7657	60304	0.47
	0.945	58 651.74	0.73	807 102.17	6.91	15040	2575	33616	0.30
	0.905	60 637.81	4.14	691 028.41	8.46	11924	1571	25240	0.23

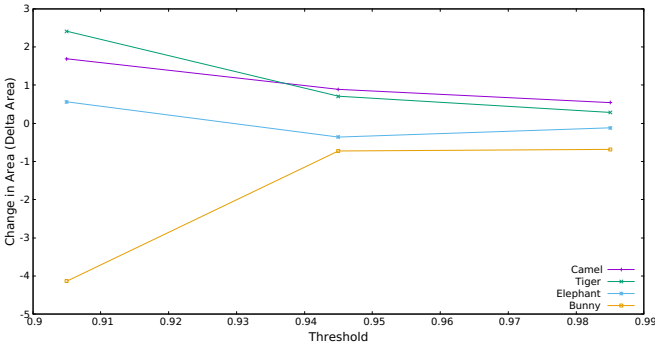
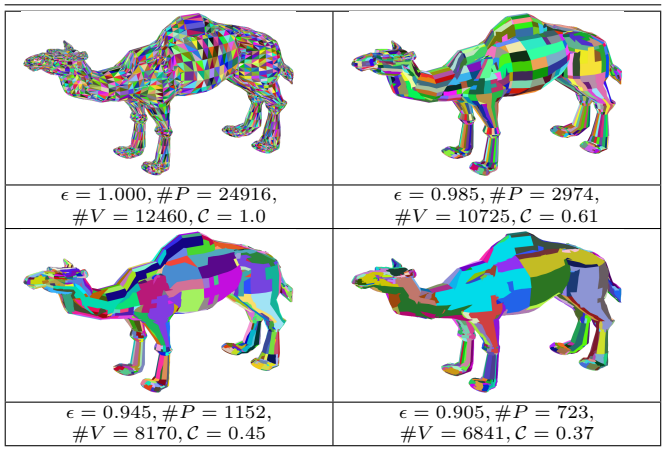


Fig. 3: Change in surface area for varying thresholds

VII. Conclusion

We have presented an algorithm which polygonized a given 3D digital object. By polygonization we mean that a set of similar neighbouring triangular faces are coalesced together to get a larger polygonal face. These faces are further refined using least square method. The resultant polygons give the approximate polygonized object. As an immediate outcome, we achieve compression which is dependent upon the threshold. Also, the size and nature of these surface polygons characterize the given 3D object. This work can be useful in 3D object transmission in view of its ability to compress the objects. It can be further used for shape identification and retrieval which

TABLE II: Polygonization of Camel for different thresholds



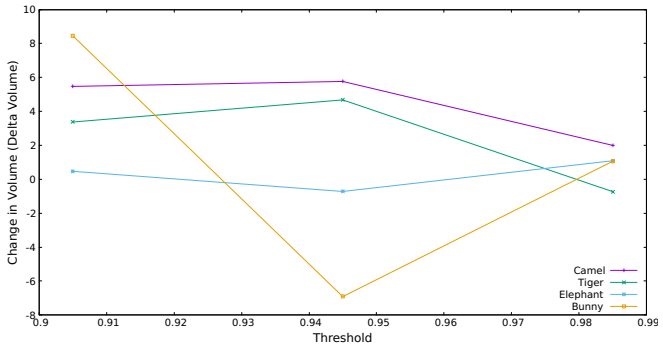


Fig. 4: Change in volume for varying thresholds

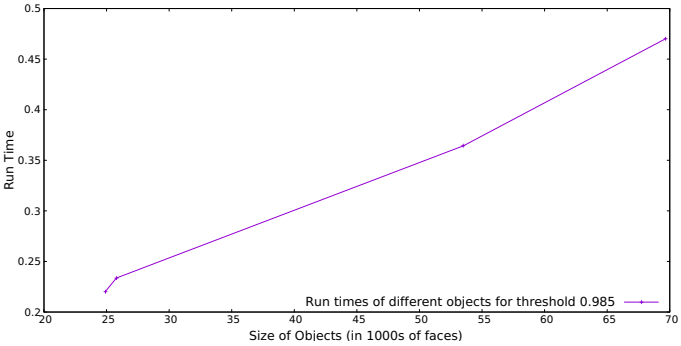


Fig. 5: Process Run Times for threshold 0.985

TABLE III: Polygonization of Tiger for different thresholds

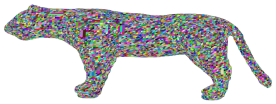
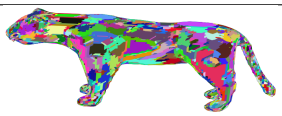
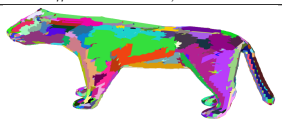
	
$\epsilon = 1.000, \#P = 25800,$ $\#V = 12898, \mathcal{C} = 1.0$	$\epsilon = 0.985, \#P = 4807,$ $\#V = 10191, \mathcal{C} = 0.59$
	
$\epsilon = 0.945, \#P = 1555,$ $\#V = 7469, \mathcal{C} = 0.40$	$\epsilon = 0.905, \#P = 933,$ $\#V = 6409, \mathcal{C} = 0.34$

TABLE IV: Polygonization of Elephant for different thresholds

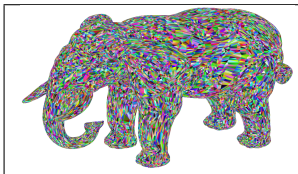
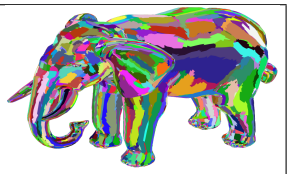
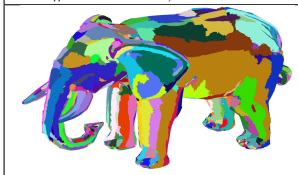
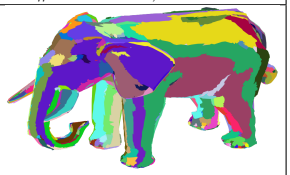
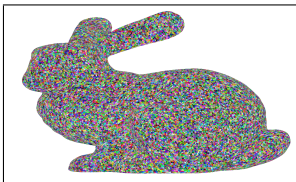
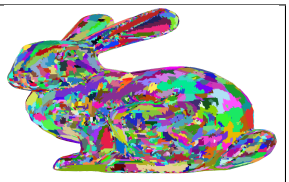
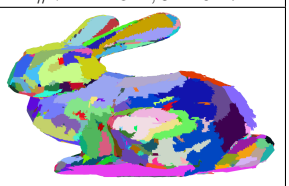
	
$\epsilon = 1.000, \#P = 53506,$ $\#V = 26755, \mathcal{C} = 1.00$	$\epsilon = 0.985, \#P = 6370,$ $\#V = 18526, \mathcal{C} = 0.50$
	
$\epsilon = 0.945, \#P = 2144,$ $\#V = 11838, \mathcal{C} = 0.31$	$\epsilon = 0.905, \#P = 1261,$ $\#V = 9469, \mathcal{C} = 0.24$

TABLE V: Polygonization of Bunny for different thresholds

	
$\epsilon = 1.000, \#P = 69664,$ $\#V = 34838, \mathcal{C} = 1.0$	$\epsilon = 0.985, \#P = 7657,$ $\#V = 22814, \mathcal{C} = 0.47$
	
$\epsilon = 0.945, \#P = 2575,$ $\#V = 15040, \mathcal{C} = 0.30$	$\epsilon = 0.905, \#P = 1571,$ $\#V = 11924, \mathcal{C} = 0.23$

we intend to explore in future.

REFERENCES

[1] G. Turk, Re-tiling polygonal surfaces., in: Siggraph, Vol. 92, 1992, p. 2.

[2] J. Rossignac, P. Borrel, Multi-resolution 3d approximations for rendering complex scenes, in: Modeling in computer graphics, Springer, 1993, pp. 455–465.

[3] W. J. Schroeder, J. A. Zarge, W. E. Lorensen, et al., Decimation of triangle meshes., in: Siggraph, Vol. 92, 1992, pp. 65–70.

[4] A. D. Kalvin, R. H. Taylor, Superfaces: Polygonal mesh simplification with bounded error, IEEE Computer Graphics and Applications 16(3) ('96) 64–77.

[5] R. Ronfard, J. Rossignac, Full-range approximation of triangulated polyhedra., in: Computer Graphics Forum, Vol. 15, Wiley, '96, pp. 67–76.

[6] C. L. Bajaj, D. R. Schikore, Error-bounded reduction of triangle meshes with multivariate data, in: Visual Data Exploration and Analysis III, Vol. 2656, pp. 34–45.

[7] H. Hoppe, Progressive meshes, in: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, ACM, 1996, pp. 99–108.

[8] T. S. Gieng, B. Hamann, K. I. Joy, G. L. Schussman, I. J. Trotts, Smooth hierarchical surface triangulations, Visualization'97, IEEE, pp. 379–386.

[9] M. Garland, P. S. Heckbert, Surface simplification using quadric error metrics, in: Proc. of the 24th annual conf on Computer graphics and interactive techniques, ACM Press, 1997, pp. 209–216.

[10] J. Zhu, T. Tanaka, Y. Saito, A new mesh simplification algorithm for the application of surface sculpture, in: The First JTU-TIT Joint Workshop on Creative Engineering—Mechanics, Control and Advanced Robotics, Shanghai/Xi'an, China, no. 11-15, '05, p. 9.

[11] L. Wang, J. Li, I. Hagiwara, A topology preserved mesh simplification algorithm, in: 2009 WRI World Congress on Computer Science and Information Engineering, Vol. 1, 2009, pp. 698–702.

[12] H. Wang, G. Yin, J. Zhang, D. Wang, J. Wang, An efficient mesh simplification algorithm, in: 2009 Fourth International Conference on Internet Computing for Science and Engineering, 2009, pp. 60–63.

[13] X. Ma, J. Zheng, Y. Shui, H. Zhou, L. Shen, A novel method of mesh simplification using hausdoff distance, in: 2012 Third World Congress on Software Engineering, 2012, pp. 136–139.

[14] W. Jian, W. Hai-Ling, Z. Bo, Q. Fu, J. Ni, An efficient mesh simplification method in 3d graphic model rendering, in: 7th International Conf. on Internet Computing for Engineering and Science, '13, pp. 55–59.

[15] K.-W. Ng, Z.-W. Low, Simplification of 3d triangular mesh for level of detail computation, in: 11th Intl. Conf. on Computer Graphics, Imaging and Visualization, IEEE, '14, pp. 11–16.

[16] G. An, T. Watanabe, M. Kakimoto, Mesh simplification using hybrid saliency, in: International Conference on Cyberworlds, '16, pp. 231–234.

[17] Z. Shi, K. Qian, K. Yu, X. Luo, A new triangle mesh simplification method with sharp feature, 7th Intl Conf on Digital Home, '18, 324–328.

[18] Least square method. https://www.ilikebigbits.com/2015_03_04_plane_from_points.html. Accessed: 2019-10-27.