Comparison of Multiple Meta-heuristic 3D Mesh Optimization Algorithms

Agha Syed Nasir Mehmood Azeemi
Department of Computer Science
DSSE, Habib University
Karachi, Pakistan
aa04377@st.habib.edu.pk

Khubaib Naeem Kasbati
Department of Computer Science
DSSE, Habib University
Karachi, Pakistan
kk04333@st.habib.edu.pk

Moid Ul Huda
Department of Computer Science
DSSE, Habib University
Karachi, Pakistan
mh05205@st.habib.edu.pk

***Abstract*—In computer graphics, models are represented in space by polygonal meshes that define the shapes of objects by connecting vertices and edges to form surfaces. These surfaces can be very complex hence there is a need to simplify them if sufficient computing power isn't available for rendering. Mesh simplification is an NP-Hard Problem. This paper looks at several meta-heuristic mesh optimization algorithms and performs a comparative analysis on the results**

***Index Terms*—meta-heuristic, mesh optimization, genetic algorithms, multi-objective optimization, Computer Graphics**

I. Introduction

3-Dimensional meshes are an important aspect of 3D visualizations and simulations. Their real-world applications range from the fields of Computer Aided Design (CAD), Computer Vision (CV) to Topography and Medicine. A mesh model comprises of thousands of vertices in 3D space, which are connected with thousands of edges to form surfaces. Processing these surfaces (e.g. Rendering) requires substantial amount of time, energy and computational resources which leads to problems in time-sensitive applications (e.g. medicine). To cut down on our costs (time, energy and computational power) it is necessary to simplify our meshes.

Mesh Simplification/Optimization is a process where we can reduce the total number of vertices while trying to preserve the topology, shape, volume, structure and boundaries of the original mesh as best as possible. n vertices from the original mesh are carefully extracted such that more vertices from curved regions are chosen with respect to flat regions, to maximize accuracy mesh approximation. This allows for faster processing and helps cut down our costs.

However, mesh simplification is a very hard problem. Classical algorithms such as decimation of vertices, vertex clustering, edge contraction or energy function optimization have been used but have either failed to preserve the topology of the object or are too expensive (computationally). In contrast, nature-inspired meta-heuristic algorithms have fared better.

In this paper we will look at some nature-inspired meta-heuristic algorithms that, given no prior knowledge about the smoothness of a mesh, would simplify the mesh accordingly. We will look at one selection-reproduction algorithm, a genetic algorithm and a multi-objective optimization algorithm. All of these algorithms are implemented in Python.

In Section III, we will look into detail about each of these algorithms, In Section IV, we dive deep into the results, seeing the parameters and perform comparison between how these algorithms performed on different meshes. In Section V, we will summarize our findings.

II. Literature Review

Mesh simplification is an important problem and many state-of the art solutions have been proposed in literature over the years. Classical solutions have been developed which are based in the iterative method. Notable examples are of [1] [2] [3] which implement 3 different decimation approaches namely: removal of edges, removal of triangle, and removal of vertices, respectively. Another example is of [4] which uses energy function optimization approach where edges are iteratively swapped, split, or collapsed. [2] uses an enhanced version of the splitting technique. However, Classical algorithms such as decimation of vertices, vertex clustering, edge contraction or energy function optimization either do not preserve the topology of the model and hence the surface is not preserved in the original state or the algorithm is too computationally expensive that the applications that it is required in becomes useless as time constraints are important in applications outside of research.

In contrast, nature-inspired meta-heuristic algorithms have a good trade off between computational complexity and quality of the results. [5] tackled the problem of mesh simplification of a face by constructing triangular meshes via Delaunay Triangulation with selected number of vertices using a single-objective genetic algorithm. [6] proposed an Evolutionary Algorithm (EA) based on [5] using Orthogonal Array Crossover (OAX). Finally, [7] proposed a genetic algorithm based on super-face that performs triangular-mesh reduction. So far, nature-inspired meta-heuristic algorithms such as those discussed in the papers mentioned above, have produced satisfactory results.

III. Algorithms

A. Selection-Reproduction

The algorithm begins by converting a 3D mesh in Cylindrical coordinate system (ρ, θ, z), centered at the axis $\rho = 0$, into a 3D Cartesian space (x, y, z). The mapping is done such

that $x = \theta$, $y = z$ and $z = \rho$. To implement the algorithm the the mesh is required to be 2D. Hence, the mesh is flattened using the function $f(x, y, z) \in R^3 \mapsto f(x, y) \in R^2$ and the depth information in dimension z, for each grid-point (x, y), is stored separately for later use. This creates the grid-point configuration for the first iteration P_0.
Delaunay Triangulation was then used to create triangles T in the 2D mesh to approximate the 3D facial surface. For an iteration i, Delaunay Triangulation of a grid-points configuration is denoted by $D(P_i)$. To accurately approximate 3D surfaces in 2D, we calculate the error e using Euclidean Distance (1) between the original depth values (z) of a grid-point (x, y), contained inside a Delaunay Triangle $T \in D(P_i)$, and the new linearly interpolated depth values z_i at (x, y).

$$d_i = |z(x, y) - z_i(x, y)| \tag{1}$$

The total error $\bar{e}$ for a Delaunay Triangle T is the sum of errors e for all grid-points $(x, y) \in P_i$ within T.
The algorithm is described bellow:

1) Distribute **N** vertices in R^2 space to create grid-point configuration P_0.
2) generate a population of Delaunay Triangles $D(P_i)$.
3) Calculate the total error $\bar{e}$ for each triangle $T \in D(P_i)$.
4) Select $\lambda \times N$ triangles T with smallest errors e, where $0 < \lambda < 1$
5) Randomly delete a vertex v for each selected triangle from the list of vertices.
6) Select $\lambda \times N$ triangles T with the largest errors e, where $0 < \lambda < 1$
7) Add the centroids c of each selected triangle to the list of vertices.
8) Repeat steps 2 to 6 for i iterations

The Basic idea of the algorithm is that curved regions produce greater errors when interpolated. So we add vertices in such regions and to preserve them. We delete vertices from flatter regions, which are detected because they produce smaller errors. Therefore, this iterative algorithm tries to minimize the total error produced in the approximation.

B. Genetic Evolution

In the genetic algorithm, we define the chromosome P_n where n represents the number of points stored. We store point using the mapping described in the above Section III-A. Formally a chromosome is defined as follows

$$P_n = \{\mathbf{x}_i \mid i = 1, \dots n\}$$

where $\mathbf{x}_i$ represents each point (x_i, y_i) such that $\forall i, j \in \{1, 2, \dots, n\}, x_i \neq x_j$. x_i and y_i are stored as integers which represent position of our coordinates on a rectangular grid such that $0 \leq x_i \leq N_x, 0 \leq x_i \leq N_y$ where N_x, N_y is the number of points in the x plane and y plane respectively. In order to prevent deformation of shape, we store the boundary points of our rectangle.
The gene's phenotype(visual representation) is given by using Delaunay triangulation $D(P_n)$. We use the error function as described in Section III-A.

We denote our population size as M. The algorithm is as follows: Our algorithm is described below:

1) Initialize Population: We randomly choose n points in order to generate each individual's chromosome. In order to prevent an abnormal boundary, for each boundary we allocate $\frac{\sqrt{n}}{2}$ out of the n points for the boundary randomly.
2) Selection Scheme: In order to introduce selective pressure and ensure that good solutions exist, we use the truncation selection scheme. We rank all our individuals in order of their fitness and then choose the top m individuals, We then apply our genetic operations (Mutation and crossover). We perform elitism and the best individual is sent to the next generation without any genetic operator performed on them.
3) We then perform crossover from two randomly chosen parents. Once we have chosen the parents, we establish a horizontal line randomly such that $x = x_c$ where $0 \leq x_c \leq N_x$. For one child, we will choose all points $x_i^1 \geq x_c$ where x_i^1 is any arbitrary point in Parent 1's chromosome, and all points $x_i^2 < x_c$ where x_i^2 where x_i^2 is any arbitrary point in Parent 2's chromosome. Similarly for the other child we will choose all points $x_i^1 < c$ and $x_i^2 \geq c$.
   It is possible that the number of points in the new child might exceed n, if it is does then we randomly delete some points (we need to keep the 4 boundary points). If the number of points in the new child is less, then we dd more points randomly.
4) We then apply mutation to every single point (except boundaries) to each individual (except the best) with a mutation rate p_m. If we want to mutate a point, we move it to one of the nearest neighbors on the grid.
5) Termination: Throughout the algorithm, we preserve the best fitness. We will choose to terminate if we reach a specific number of iterations or the value can no longer be improved.

C. Multi-Objective Optimization

This is an Evolutionary Algorithm inspired by algorithm A. A population of N_{pop} individuals is randomly initialized. Each individual in the population represents a candidate solution, that is each individual in a simplified mesh. A global list of original mesh coordinates N, is also maintained. Each individual is defined by a binary chromosome of n genes, where $n = |N|$. A True value in the i-th gene in the means that the i-th coordinate in N is present in the simplified mesh represented by the chromosome and vice versa. The fitness of each individual is calculated as the sum of all errors $\Sigma\bar{e}$ for all $T \in D(P_i)$, where $D(P_i)$ denotes the Delaney Triangulation of the i-th individual's grid-point configuration. The error $\bar{e}$ is calculated in the same way as described in algorithm A. The goal is to minimize the total error and the number of vertices in the simplified mesh.

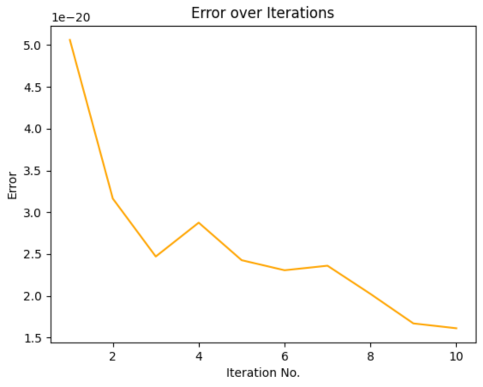

Fig. 1. Error over generations for the mulltiobjective algorithm on 10 generations with size of population 10 on the laurana.ply dataset

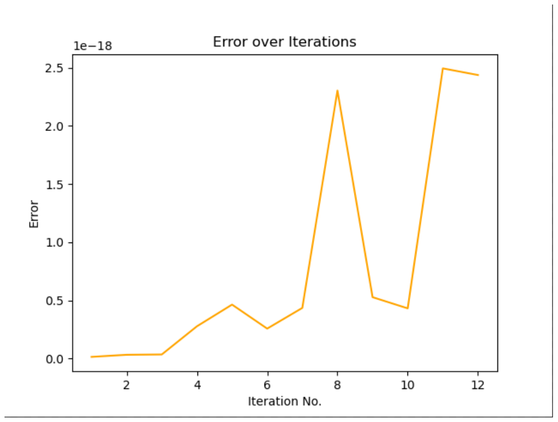

Fig. 2. Error over iteration for the evolutionary/iterative algorithm on 0.15 time the original dataset of laurana.ply

The genetic operations include mutation and crossover. Mutation on an individual is perform by randomly changing assigned gene values. The crossover operation takes two parents and produces 2 off-springs, each with a portion of both parents gene values.

1) Randomly Initialize Population: Create N_{pop} binary chromosomes with n genes and randomly assign each gene True or False values.
2) For each individual in the population, generate Delaunay Triangles $D(P_i)$.
3) Calculate $\bar{e}$ for each triangle $T \in D(P_i)$ and find the fitness of each individual $\Sigma\bar{e}$.
4) Perform genetic operations as described above.
5) Eliminate 2 chromosomes with the lowest fitness from the population.
6) Repeat steps 2-3 for each generation.

IV. Results and Discussion

In doing our analysis, we had some interesting observations. Due to Python, the time taken to optimize the mesh was astronomical, up to minutes. With Selection-Reproduction taking 15-20 mins, Genetic Algorithm takes an extremely long amount of time ,10 mins for 5 generations with population of 10 and Multi-Objective Optimization taking up to 50 minutes for 10 generations with a population size of 10.
We can generally see that the results were converging very quickly and with minimal error showing the accuracy of the algorithm.

The experiments on the algorithms were carried out on 2 different datasets (laurana dataset and Stanford bunny dataset). Selection-Reproduction algorithm was set to a simplification of 15% of the data and $\lambda = 0.2$ for both the dataset. As the algorithm is fixed to work on data where the data can be mapped on the 2d plane to a rectangular space, the datasets used here were not helpful as they had lots of slim and thick features that did not map as a rectangle to the 2d plane and hence boundaries were not retained in the outputs. In the generated results in figure 7, it can be seen that the points got clustered to the places where

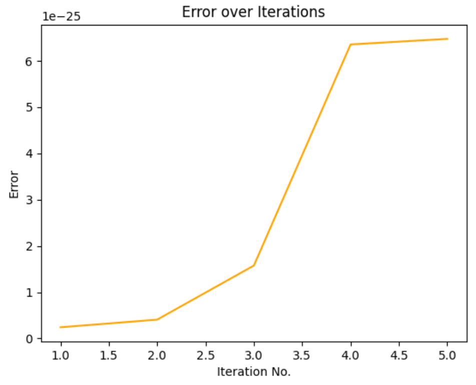

Fig. 3. Error over iteration for the evolutionary/iterative algorithm on 0.15 time the original dataset of bunny.ply

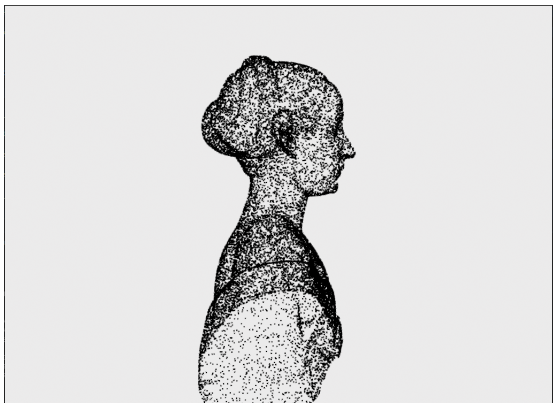

Fig. 4. Laurana point cloud genertaed by the Multi Objective algorithm

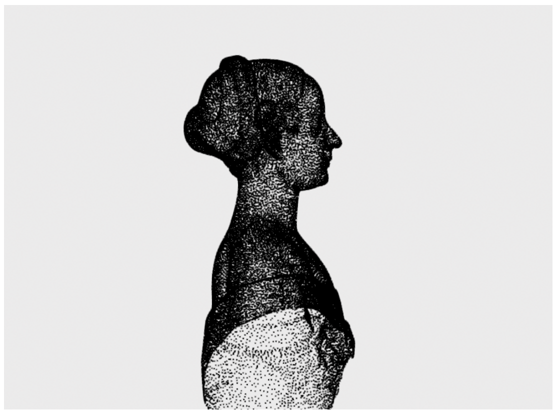

Fig. 5. Original Laurana point cloud

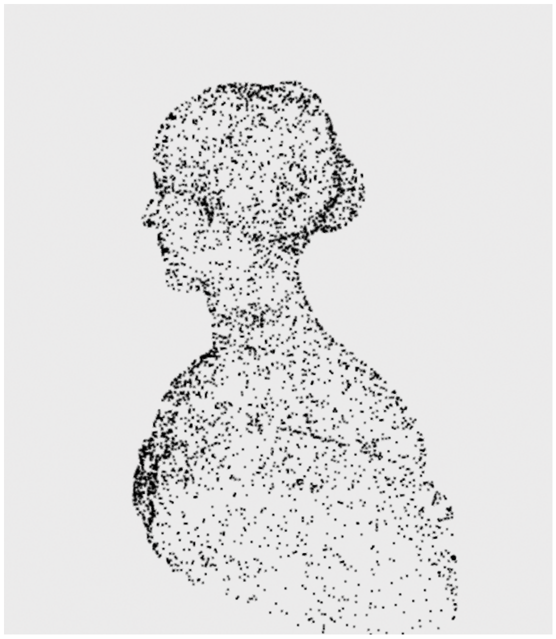

Fig. 6. Laurana point cloud genertaed by the Genetic algorithm

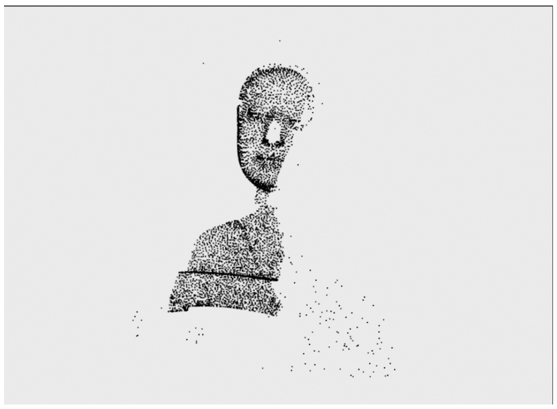

Fig. 7. Laurana point cloud genertaed by the Selection-Reproduction algorithm

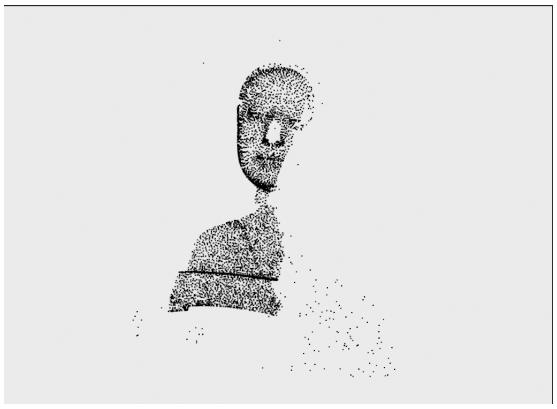

Fig. 8. Bunny point cloud genertaed by the Genetic algorithm

there was a lot of curves and the smooth surfaces lost most of the points. This was also due to the fact that the lambda value was big and it changed the orignal sampled data set dramatically. In the graph as well in figure 2 it can be seen that that point cloud got away from the original dataset

In the genetic algorithm the results from the figure 6 it can be seen that the results are good as the structure of the 3d model can be easily recognized even with 15% of original data points.

In the multiobjective algorithm, the results from figure 4 was optimized to minimize both the number of points in the dataset as well as the error from the original data set. The results can also be seen in figure 1 shows that error was reduced gradually and a better mesh was formed in the end.

In the Selection-Reproduction algorithm, the time taken was extremely large, we suspect that this is since the value of λ is too high leading to a lot of triangles having their points removed at the end and being allocated elsewhere, leading to a high amount of time taken throughout the algorithm, this may also be one of the reasons why error is increasing. By making λ a function of time, we can prevent too many points being removed at the later stages of the algorithm.
We suspect some of the large portion of time taken is due to the use of Python. By adding optimizations of a compiled language like C++, the time taken for the algorithm could be drastically reduced.

V. Conclusion

To conclude, we observed that all 3 algorithms give a good simplification of the meshes they operated on with the multi-objective optimization function performing the best. These methods give output that is suitable visually however they also do take some time for computation. These methods can be parallelized in order to increase speed of results. Our error function, particularly Delaunay Triangulation used the most amount of time. Future research can be done to find a suitable less computationally intensive error function with similar or higher accuracy.

References

[1] B. Hamann. A data reduction scheme for triangulated surfaces. *Comput. Aided Geom. Des.*, 11:197–214, 1994.

[2] Hugues Hoppe. Progressive meshes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, page 99–108, New York, NY, USA, 1996. Association for Computing Machinery.

[3] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of triangle meshes. *SIGGRAPH Comput. Graph.*, 26(2):65–70, July 1992.

[4] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, page 19–26, New York, NY, USA, 1993. Association for Computing Machinery.

[5] Y. Fujiwara and H. Sawai. Evolutionary computation applied to mesh optimization of a 3-d facial image. *IEEE Transactions on Evolutionary Computation*, 3(2):113–123, 1999.

[6] Hui Ling Huang and Shinn-Ying Ho. Mesh optimization for surface approximation using an efficient coarse-to-fine evolutionary algorithm. *Pattern Recognition*, 36(5):1065–1081, May 2003.

[7] B. Rosario Campomanes-Álvarez, Oscar Cordón, and Sergio Damas. Evolutionary multi-objective optimization for mesh simplification of 3d open models. *Integr. Comput.-Aided Eng.*, 20(4):375–390, October 2013.