Geometry Compression for 3D Polygonal Models using a Neural Network

Nadine Abu Rumman
Instructor – CGs Dept.
PSUT- Jordan

Samir Abou El-Seoud
Professor – CS Dept.
PSUT- Jordan

Khalaf F. Khatatneh
Asso. Prof. - IT-School
Al-Balqa Appl. Univ.
Jordan

Christain Gütl
Key Research- IT-School
IICM – TU Graz, Austria

ABSTRACT

Three dimensional models are commonly used in computer graphics and 3D modeling characters in animation movies and games. 3D objects are more complex to handle than other multimedia data due to the fact that various representations exist for the same object, yielding a number of difficulties, among of which are the distinct sources of 3D data. Research work in the field of three dimensional environments is represented by a broad spectrum of applications. In this paper we restrict ourselves only on how to do compression using a neural network in order to minimize the size of 3D models for making transmission over networks much faster. The main objective behind this compression is to simplify the 3D model and make handling the large size of 3d objects much easier for other processes. Even the process of rendering**,** digital watermarking, etc., will be faster and more efficient.

Keywords

Geometry Compression, Artificial Intelligent, Genetic Algorithm, Neural Network, Multilayer feed forward.

1. INTRODUCTION

Multimedia plays an increasingly important role in various domains, including Web applications, movies, video games, and medical visualizations. The rapid growth of digital media data over the Internet makes it easy for everybody to access, copy, edit, and distribute digital contents such as electronic documents, images, sounds and videos. Motivated by this, much research work has been dedicated to develop methods for digital data copyright protection, tracing the ownership, and preventing illegal duplication or tampering. One of the most effective technique for the copyright protection of digital media data is the process, in which hidden a specified signal (watermark) is embedded in digital data while minimizing the size of the data media. As we mentioned before, we will restrict ourselves in this paper to the process of compression of 3D models. On one hand, we are looking for an effective technique for copyright protection by embedding a hidden specified signal in the digital data and at the same time trying not to increase the size of the digital data using effective compression technique. The existing efforts in the literature on compression have been concentrated on media data such as audio, images, and video, but there are no effective methods for the compression of three dimensional (3D) models, especially for easy distribution of the models over the Internet or using other media. There are three aims behind any compression technique: efficient rendering, progressive transmission, and maximum compression [1].

Efficient rendering: Encoding for efficient rendering tries to reduce the amount of data that needs to be sent to the graphics card. When each triangle of the mesh is rendered individually the card must process every mesh vertex an average of six times.

Progressive transmission: Encoding for progressive transmission uses incremental refinements of mesh connectivity and geometry so that partial data already represents the entire mesh at a lower resolution and decoding starts with a small base mesh and expands the collapsed edges in reverse order.

Maximum compression: Encoding for maximum compression tries to squeeze a given mesh into as few bits as possible for faster network transmission and more compact storage.

This paper provides the necessary information related to the compression of 3D objects. First we will introduce the Artificial Intelligent (AI) technique, namely the artificial neural network (ANN), that we have chosen for applying our compression. Then, we will explain why the Genetic Algorithm (GA) is not good enough. Thereafter, we will start with our compression methodology and algorithm. Toward this end, we will try to give satisfactory answers to many important questions related to the compression process of 3D objects. For example:

1. What are the main input parameters in a neural network in a given input layer?
2. How many hidden layers exist?
3. What is the proposed architecture of the neural network (NN)?
4. What is the error function for the neural network?
5. What is the tool of the neural network we used?
6. What are the training and testing data for the neural network?

Finally we will present our results on three 3D models as samples for our algorithm and functions in this domain. The main objective from this compression stage is re-meshing 3D objects for faster transmission, efficient rendering operations and any other process.

2. BACK GROUND

The polygonal meshes representation of a 3D object is defined by a set of plane polygons, which contain three kinds of elements: faces, edges and vertices. Each vertex is defined by three coordinates x, y, and z, each of which requires 4 bytes to be represented [2]. Those meshes contain two kinds of information: the geometry and connectivity. The geometry describes the vertices' coordinates in the 3D space, and the connectivity describes how to connect these positions for better appearance and high precision. This process of representing 3D object requires a huge number of vertices and therefore a rapid increase in the amount of data used. This in turn requires high bandwidth transmission and high memory storage [3]. For this reason, the proposed algorithm should compress the model without much noise or loss of details, but with a smaller amount of data that is easier to work with. After that, a robust watermark is inserted into the model. The process of providing a robust digital watermarking algorithm for polygonal meshes will be discussed in another paper.

Our main target here is to produce a compression algorithm which could be applied to 3D-based polygonal representations and would be able to solve the problem of large size data in polygonal representation. Since 3D polygonal meshes representations are frequently used in computer graphics and modeling, our compression algorithm should find a concise representation of the 3D model and remove redundancy, with minimal bit rate, distortion, and noise.

When the compressed 3D polygonal meshes become ready, a robust digital watermark should be applied on these meshes after selecting a suitable scheme to insert a watermark that covers the entire model.

We will choose one of the AI techniques to solve the problem of compression for 3D model. It is known that AI tools help to find most critical vertices or points in a 3D model which in turn will help us in all the processes to be executed.

Among all the AI tools available, the most suitable tools for our case are the artificial neural network (ANN), and the genetic algorithm (GA). Fuzzy logic is more capable for controller systems than for our case, where there are no explicit rules found to build compression models. Moreover, a fuzzy logic classifier is simple to design but the computational speed is very low. Membership functions, if-then rules, and logical operators provide robust classifiers. However, tuning membership functions with a large volume of features proved to be its only disadvantage [4].

3. RELATED WORK

The first compression methods proposed for polygonal meshes were particularly tailored to the problem of in-memory storage and rendering. Many techniques exist for lossy and lossless compression of 2D pixel images, while few techniques work with 3D objects. Among the first researchers who worked in this domain is Michael Deering from Sun Microsystems [5], who explained the concept of geometry compression working with 3D triangles using quantization of coordinate values from 32 bits to 16 bits, a 9- bit index into a global list of values for vertex, and a 15 bit representation of color value, and provided a new technique for lossy compression of 3D geometric data with ratio of 6 and 3 to 1 achievable with little loss in displayed object quality and with fast rendering, with geometry decompression hardware for more limited local storage requirement.

The approach of Martin Isenburg and Stefan Gumhold [6] is quite different; where they proposed an out-of-core mesh compression technique that converts gigantic meshes into streamable, highly compressed representation. During decompression, only a small portion of mesh needs to be kept in memory at any time. As full connectivity information is available, along the decompression boundaries, which provides seamless mesh access for incremental in-core processing on gigantic meshes. Therefore, Isenburg *et al*. [7] suggest cutting large meshes into smaller pieces that can be dealt with in-core. They process each piece separately by first constructing explicit connectivity, which is then compressed with two-pass coder, before compressing the vertex positions with parallelogram predictor in third pass. The pioneer in this field, Martin Isenberg, introduces also, with Snoeyink, a simple schema for encoding the connectivity of polygon mesh that is based on assigning a code to each mesh edge and the mesh step encoding is directly applied on polygonal representation avoiding the triangulation step. This compression algorithm is called facefixer [1].

The spectral compression effort of Karni and Gotsman [8] was based on the construction of a set of basis function for the decomposition of a triangle mesh into signals. The low frequency components in the signals correspond to smooth features and high frequency component corresponds to discontinuities such as creases, folds and corners.

Compression of Large engineering 3D models using automatic discovery [9] was provided by applying the algorithm for automatic discovery of repeating feature in 3D polygon mesh models, compression of geometry of 3D polygon mesh models by removing the redundancy in the representation of repeating geometric feature patterns, and new scheme that can incorporate the best results achieved in the area of connectivity of compressed polygonal meshes.

Chow [10] proposed an algorithm that locally constructs the generalized triangle mesh by starting with triangle strips adjacent to edges and covers the object's mesh in a spiraling pattern, taking care not to overflow the vertex queue. Using this algorithm, an average ratio of 0.67 is obtained between the number of explicitly coded vertices and the total number of triangles. As reference, this ratio is 3 for independent triangles, 1 for infinitely long generalized triangle strips and ≈ 0.5 for a generalized triangle mesh on an infinite regular mesh.

Rossignac proposed the Edgebreaker [11] connectivity coding scheme that has a guaranteed worst case compressed rate of 2 bits per triangle for genus zero meshes with no more than one boundary. One important point about Edgebreaker is that no index needs to be coded with the split operation, which is unlike that used in triangle meshes compression. King *et al*. [12] extend the original Edgebreaker algorithm to handle pure quadrilateral mesh, or Q-meshes and mixed quadrilateral and triangles mesh, or QT-meshes. The proposed algorithm encodes quads by splitting them into two triangles through one of their diagonals.

More recently, Khodakovsky *et al*. [13] extended valence based coding to polygonal meshes, the entropies of the vertex graph and its dual, the face graph, are obviously the same as the information content does not change. Based on this remark, the authors proposed that an optimal polygon mesh coder should be dual.

Finally, mesh compression was applied using Multi-layer Feed Forward Neural Network that was proposed by Emmanouial Piperakis *et al* [14], where the applied neural network includes the vertex coordinates, the connectivity and normal information in one compact form, converting the discrete and polygon mesh representation into an analytical form. The neural compression representation is viable to 3D transformations without the need for disrupting the accuracy of the geometry. This basic method works for compression stage in this paper. More details will be given in section 5.

4. GENETIC ALGORITHM EXPERIMENT

We can use a genetic algorithm (GA) to solve this problem. When we try to use GA on a data set of 3D models, like the sample of input data for the Cow model and Beethoven model shown on figure 4.1 and figure 4.2 respectively, we see that they are displayed in two forms:

- shaded form in Maya software and
- Point cloud in MATLAB, where the genetic algorithm has been applied on this data.

We notice that the shaded form looks like 3D more than the point cloud form because MATLAB doesn't have shading (Shading addresses how different types of scattering or reflection are distributed across the surface on an object). Shading does not exist in the matlab environment and therefore the model appears as point cloud in matlab. The package that used in this Experiment is the Genetic Algorithm and Direct Search Toolbox™ in matlab with the following characteristics:

- Goal:

The main goal is to reduce the number of vertices and minimize the Mean Square Error (MSE) value:

$$MSE = \sum_{i=1}^{N} \sum_{j=1}^{M} [X_{i,j} - X'_{i,j}] \qquad (4.1)$$

where V are the coordination vertices (3D point) in original mesh V' the coordination vertices (3D point) in original mesh, N denotes the number of rows and M the number of columns in the array of vertices coordinates, respectively.

- Input Parameter:

Input data: vertices coordination's x, y and z {Matrix = number of vertices * 3}, all of type double vector.

Number of variables: 3 (Best x, y and z point in model).

Number of generation: specifies the maximum number of iterations for the genetic algorithm to be performed. It will be 100 generations.

Size of population: specifies how many individuals are in each generation. With a large population size, the genetic algorithm searches the solution space more thoroughly, thereby reducing the chance that the algorithm will return a local minimum. However, a large population size also causes the algorithm to run more slowly. In our case, it will be 3000. (Depending on the compression ratio for the model).

- Fitness Function:

It is the objective function that quantifies the optimality of a solution (i.e. chromosome) in a genetic algorithm. A particular chromosome may be ranked against all the other chromosomes. Optimal chromosomes, or at least chromosomes which are more optimal, are allowed to breed and mix their datasets by any of several techniques (arithmetic cross ones, mutation, and ranking selecting function). They produce a new generation that will (hopefully) be even better. The fitness function here, based on MSE, is used to limit the noise that happens in the compressed object based on the equation given in 4.1.

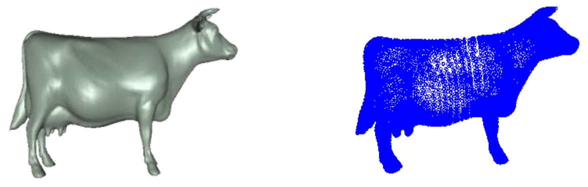
Figure 4.1 represents the code written in MATLAB for fitness.

N_Fitness.m

```
---------------------------------------------        function
[val,sol]=N_Fitness(sol)
load vertices;
[M,N]=size(vertices);
[V,F]=size(sol);
hat_v=zeros(M,N);
for i=1:V
   for j=1:F
    hat_v(i,j)=sol(i,j);
  end
end
tmp1=0;
tmp2=0;
for i=1:M
   for j=1:N
      tmp1=tmp1+(double(vertices(i,j))-  double(hat_v(i,j)))^2;
     tmp2=tmp2+double(vertices(i,j))^2;
   end
end
val=tmp1/tmp2
save all_output_data;
--------------------------------------------------------------------------------
```

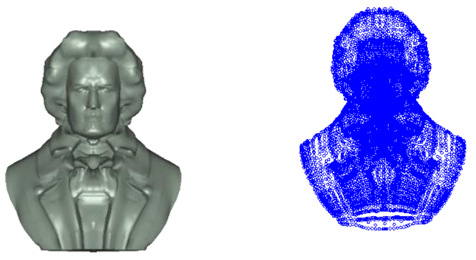
Figure 4.1: Fitness Function.

- Sample of Input Data



Number of vertices =17414

Figure 4.2: (a) Shaded Cow model and (b) Cloud point Cow model



Number of vertices = 12409

Figure 4.3: (a) Shaded Beethoven model and (b) Cloud point Beethoven model.

This data set is entered into the genetic algorithm as matrix of double value for x, y and z vertices coordinated.

- **The Result :**

Because it cannot be controlled how a genetic algorithm distributes the chromosomes in the search area and chooses the best vertices, the fitness function tries to determinate and limit the noise, but the general-form of the 3D model is destroyed and loses a lot of detailed. This makes compression, using the genetic algorithm with MSE as the fitness function, not suitable for our case. Even when a try is made to assign a given block of specific range of vertices' coordinates (for example [from 15 vertices choose 3 vertices]) to make genetic algorithm choose the best vertices from each block, it was found that the compression is not good enough for our case.

Below is the result for the sample Cow and Beethoven models. Figure 4.4 shows the best-so-far optimal values for the individual of existing variables, which are 3 variables. Figure 4.5 shows the relationship between fitness value and number of generation.

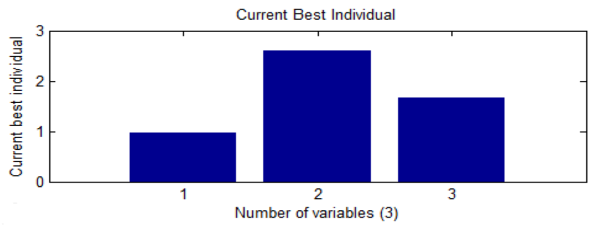


Figure 4.4: Relationship between number of variables and current best individual.

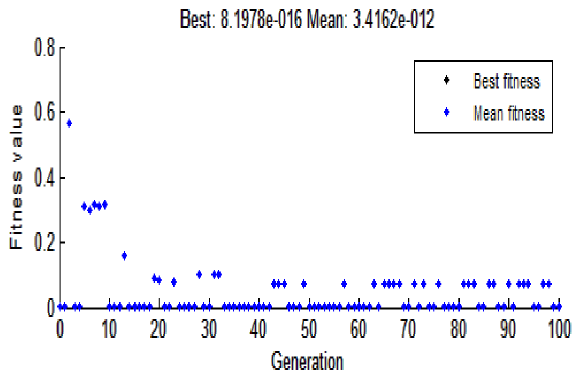
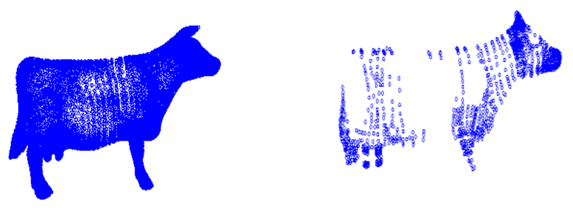


Figure 4.5: Relationship between fitness value and number of generation.

Although the evolution in GA is inductive; in nature life, it does not evolve all the time towards a good solution, it might evolve away from bad circumstances. This may cause a species to evolve into an evolutionary dead end. The GAs risk is in finding a sub-optimal solution. If there is more than one solution, GA will not work effectively. Moreover, the number of generations that mutate in each stage makes the number of experiments very large which takes a long time to get the solution. That happened in our case, large compression ratio with huge noise destroying in model, although figure 4.5 indicates that most the values of fitness function are zeros, which is consider good but for visual eye it not good enough because noise destroying in model. It makes GA fail for the compression of 3D model. Figures 4.6 and 4.7 show the huge destruction that happened for the general form of cow and Beethoven models.

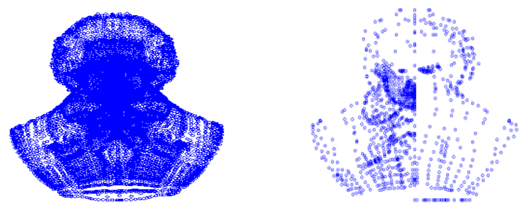


(a) (b)

Number of vertices =17414 Number of vertices =1500

Figure 4.6: (a) Original Cow model

(b) Compressed Cow model.



(a) (b)

Number of vertices =17414 Number of vertices = 1500

Figure 4.7: (a) Original Beethoven model

(b) Compressed Beethoven model.

5. 3D COMPARISON USING A NEURAL NETWORK

An artificial neural network consists of a number of very simple and highly interconnected processors, called neurons, which are analogous to the biological neurons in the brain. The neurons are connected by weighted links passing signals from one neuron to another. Each neuron receives a number of input signals through its connections. However, it never produces more than a single output signal [15].

The neurons are connected by links, and each link has a numerical weight associated with it. Weights are the basic means of long-term memory in ANNs. They express the length. A neural network learns through repeated adjustments of these weights see figure 5.1.

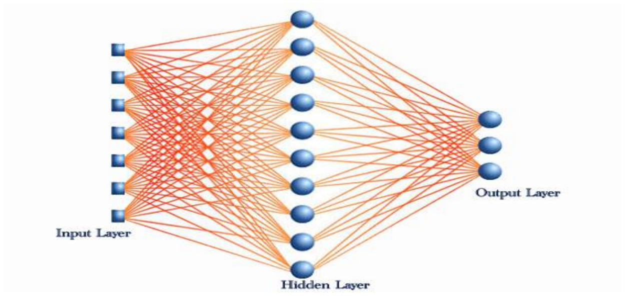


Figure 5.1: Standard Architecture for Artificial Neural Network.

Each neuron receives several input links, computes a new activation level and sends it as an output. The output is either the final solution to the problem or an input to other neurons. The neuron computes the weight sum of the input signals and compares the result with a threshold value, θ. If the net input is less than the threshold, the neuron output is -1. But if the net input is greater than or equal to the threshold, the neuron becomes activated and its output attains a value of +1.

In other words, the neuron uses the following transfer or activation function:

$$X = \sum_{j=1}^{N} x_i \ w_j$$

$$Y = \begin{cases} +1, & x \geq \theta \\ -1, & x < \theta \end{cases}$$

Where X is net weighted input to the neuron, $\boldsymbol{x_i}$ is value of input $\boldsymbol{i}$, $\boldsymbol{w_i}$ is the weight of input $\boldsymbol{i}$, N is number of neuron inputs, and Y is the output of neuron.

The sigmoid function (see equation 5.1) transforms the input, which can have any value between plus and minus infinity, into a reasonable value in the range between 0 and 1, Neurons with this function are used for forward networks.

One of the neural network types is the multilayer feed-forward neural network (MLFF). MLFF is the basic type that will be used in the proposed compression algorithm. A multilayer perception is a feed forward neural network with one or more hidden layers. Typically, the network consists of an input layer of source neurons, and at least one hidden layer of computational neurons.

In this case, the 3D object representation can depend completely on the neural network, and the learning procedure contains three main steps:

1. The presentation of the input sample.

2. The calculation of the output.
3. The modification of the weights by specified training rules.

The diagram shown in figure 5.2 presents the flow of data in the compression stage of the proposed methodology for a 3D model compression using a neural network.

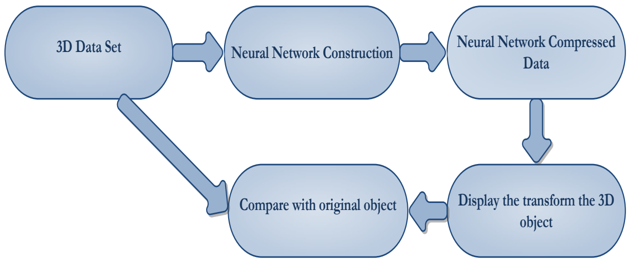

3D Data Set
Neural Network Construction
Neural Network Compressed Data
Display the transform the 3D object
Compare with original object

Figure 5.2: Data flow in proposed 3D Compression with ANN.

5.1 THE ALGORITHM

The proposed algorithm is based on the geometric and connectivity data of the 3D object. The proposed algorithm passes the following four stages:

As previously mentioned, the polygonal meshes representation of the 3D object contains two kinds of information: the geometry, and the connectivity, where compression is applied on each separately.

The neural network employed in this thesis is an MLFF neural network as a lossy compression method, where the neural network tool used for this algorithm is Mathworks tool (Neural Network Toolbox's with Multi-layer Feed Forward Architecture) the object is created manually (modeling them using Autodesk Maya 2008) and before entering the data in neural, the pre-processing should be applied on these data.

5.1.1 THE PRE-PROCESS DATA SET

Before the inputs are presented to the MLFF, the data should be pre-processed. Accuracy of the outputs of the neural network depends on the data pre-processing step. Following are the steps that should be done in the data pre-processing stage.

- Normalization
- Extract main features of the dataset

The supervised learning problem is divided into a parametric and nonparametric model. The problem here is in the nonparametric model because there is no a priori knowledge of the form of the function being estimated. Therefore, we need neural network learning by example. The learning process will be performed by a learning algorithm. The objective of this algorithm is to change the synaptic weight of the network to attain a desired design objective, which is the compressed object. Once the network has been trained, it is capable of generalization.

The examples that we used in the neural network as input are ten different 3D models that are created manually. Our target is to create desired outputs. These outputs are generated by external package for geometry compression using the Java 3D package.

Standard representations of a polygon mesh uses:

- an array of floats to specify the positions and
- an array of integers containing indices into the position array to specify the polygons.

A similar scheme is used to specify the various properties and how they are attached to the mesh[16].

For large and detailed models this representation results in large of substantial size, which makes their storage expensive and their transmission very slow.

All the positions of vertices should be normalized between 0.0 and 7.0. This step helps the neural network to focus on the aim of reducing the number of vertices and reconstructing the faces. All vertices values should be normalized between [0-7] to make all neural network work concentrate on compression. The minimum value for all vertices of all ten created models should be 0 and the maximum values should be 7.

5.1.2 JAVA 3D GEOMETRY COMPRESSION

The geometry compression using the Java 3D package can achieve (lossy) compression ratios of between 6 and 10 to 1, depending on the original representation format and the desired quality of the final level. The compression proceeds in four stages. The first stage is the conversion of triangle data into a generalized triangle mesh form. The second is the quantization of individual positions, colors, and normals. Quantization of normals includes a novel translation to non-rectilinear representation. In the third stage the quantized values are delta encoded between neighbors. The final stage performs a Huffman tag-based variable-length encoding of these deltas. Decompression is the reverse of this process. The decompressed stream of triangle data is then passed to a traditional rendering pipeline, where it is processed in full floating point accuracy. The improvement in this package by adding optimization compression makes the loss in detail of the 3D object much smaller. Figure 5.7 displays the pseudo code that describes part of this work. Also, there are some definitions that have been added to identify the critical vertices so that removing those critical vertices can be controlled such that the number of vertices remains correspondent to these edges which are never used by the compression algorithm. The following are the definitions of those vertices depending on invariant vertex identification that is provided by [18].

1. Boundary vertices of the 3D model are the vertices that cannot be used by the compression algorithms because these are critical vertices. These are defined as vertices which influence the shape of the 3D model. These vertices can be used for watermarking later on.
2. Neighboring vertices that split a vertex will never be used by the compression algorithms.
3. Vertices of edges which do not form a simple triangle, cannot be collapsed. This can be calculated from the data of 3D models by storing all the vertices and faces according to the label of vertices, and then checking every two consecutive faces. If any two consecutive triangles have two of its vertices common so that two vertices form the complex triangle, this pair of vertices cannot be used by the compression algorithm.

The complexity of invariant vertex selection is analyzed as follows according to [18]:

1. The complexity of selecting boundary vertices of the 3D object by computing convex hull takes O (*n log n*) using quick hull algorithm [19].
2. The neighboring vertices to split, which are computed after each refinement. If p is the number of split vertices in a refinement and d is the maximum degree for a vertex, then the complexity for processing these set of vertices is O (*p*d).*
3. Computing the vertices of edges which are not simple triangles. First, sort all the faces according to the label of vertices which takes O (*n log n*). Then, checking between two consecutive faces takes O (*n*) time. Therefore, we have altogether

$T(n) = n \log n + n \log n + n = 2n \log n + n$

Where *T(n)* is the time complexity and n is the number of vertices

So, the overall worst time complexity for computing this set of vertices is O (*n log n*).

Local_meshify_algorithm

```
-----------------------------------------------------------------------------
Local_meshify_algorithm (MeshRegion *region)
{CurrGtmesh = gtmeshCreate0;
MeshInfoAddMesh (region, currGmesh);
% Step 1: Find meshes boundary edges.
FindMeshBoundaryEdges (&boundaryEdges, MAXJIBUFF-SIZE);
% Step 2: Chain together triangles of the first's trip.
BoundaryEdgesGetStripFacets (boundaryEdges, &currStripFacets);
While (more triangles in the mesh region) {
% Step 3: Find boundary edges of the previous strip.
FindNextBoundaryEdges (curSttipFacets, &boundaryEdges);
% Step 4: Chain together triangles of the next strip.
change = boundaryEdgesGetStripFacets (boundary Edge,
&n extStripFacets);
if (change)
{
% Mark the new boundaries of the current strip facets.
StripFacetsMarkBoundary (currStripFacets);
MarkFacetsAsDiscovered (currStripFacets);
% Step 5: Assign vertex replacement codes to form a gtristrip.
GtriStrip = findGeneralTrianglesStrip
-------------------------------------------------------------
```

Figure 5.7: Pseudo code for the remesh algorithm.

The overall complexity of remesh algorithm using Java 3D geometry compression, in addition to invariant vertex selection algorithm, is as follows:

1. The invariant vertex selection algorithm complexity is O(*n log n*).
2. The remesh algorithm complexity is T (n) =15n+4 which is equal O (*n*).

Therefore the calculation for the overall time complexity for the compression algorithm is:

$T(n) = (2n \log n + n) * (15n + 4)$
$T(n) = 30 n^2 \log n + 15 n^2 + 8 n \log n + 4n = O(n^2 \log n)$

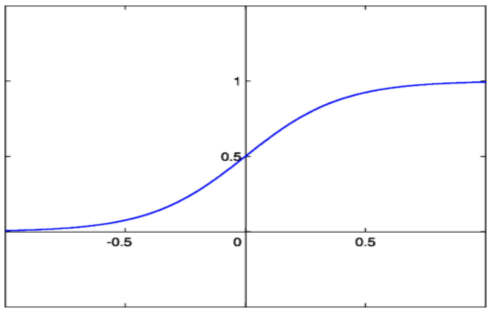
where *T (n)* is time complexity and *n* is the number of vertices.

5.1.3 THE STRUCTURE OF a MLFF NEURAL NETWORK

The neural network structure contains an input layer, one hidden layer, and an output layer. All nodes are fully connected. The network takes x, y and z coordinates of the vertices as input. The activation function used is sigmoid logistic function.

A log-sigmoid function, also known as a logistic function, is given by the relationship:

$$\sigma(t) = \frac{1}{1+e^{-\beta t}} \text{ ----------------- } (5.1.2)$$



where β is a slope parameter. The sigmoid has the property of being similar to the step function, but with the addition of a region of uncertainty. Sigmoid functions in this respect are very similar to the input-output relationships of biological neurons, although not exactly the same. Below is the graph of a sigmoid function.

Sigmoid functions are also prized because their derivatives are easy to calculate, which is helpful for calculating the weight updates in certain training algorithms. The derivative is given by:

$$\frac{d\sigma(t)}{dt} = \sigma(t)[1-\sigma(t)] \text{ ------------}(5.1.3)$$

The number of neurons in input layer is 4, where the first three input vectors are the x, y and z vertices coordinates and the

Fourth input is the max face ratio which indicates that the maximum face must remain as it is. The number of neurons in the hidden layer is between 3 and 4 because the compression

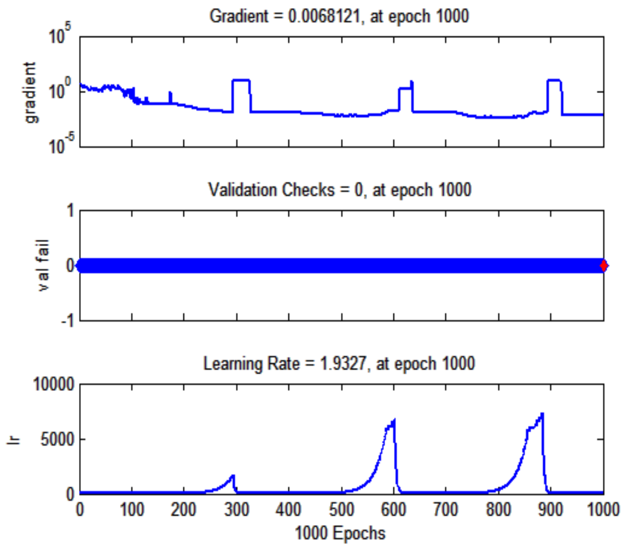


Figure 5.10: The best performance of network retch in 857 epochs.

process overall depends on the hidden layer, so the number of neurons should be absolutely less than the number of neurons in the input layer in order to do compression. For higher accuracy, the number of neurons in the hidden layer should be increased but this reduces the compression process.

A two-layer feed-forward network with sigmoid hidden neurons and linear output neurons can fit multi-dimensional mapping problems arbitrarily well, given consistent data and enough neurons in its hidden layer.

Figure 5.5 displays the neural network structure with a given 3D model object sample for input object and target object.

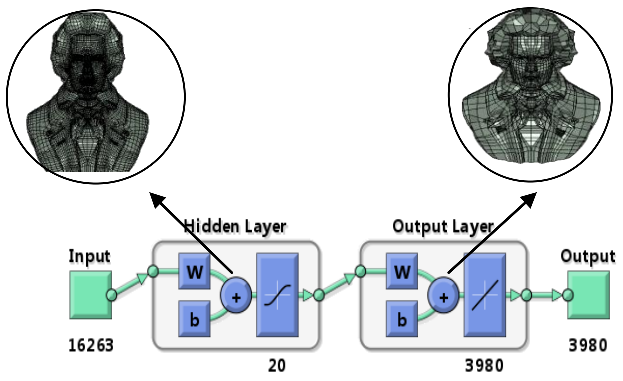
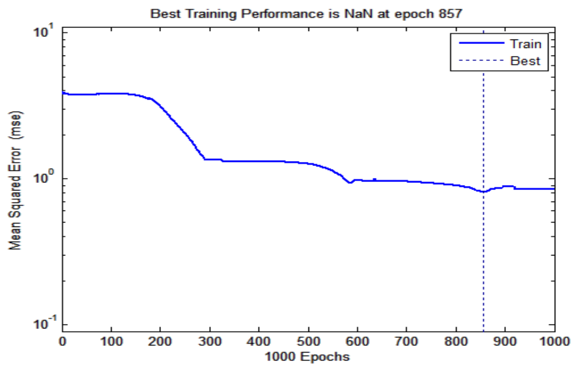


Figure 5.8: Feed Forward Neural Network Structure.

5.1.4 THE TRAINING SAMPLE

The network trains 1000 times with the training set until the MSE is small; this MSE is the difference between the output objects and desired objects, Training automatically stops when generalization stops improving, as indicated by an increase in the mean square error of the validation samples.

The network will be trained with the gradient-Descent back propagation algorithm with adaptive learning rate. Training time for each model takes approximately 2 hours and 30 minutes; for all ten models takes 25 hours and 12 minutes.



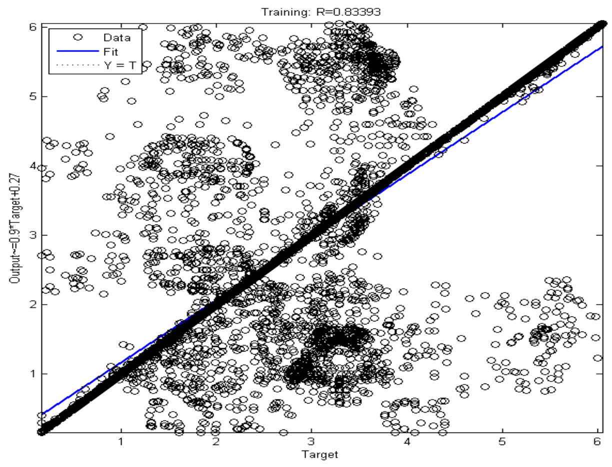


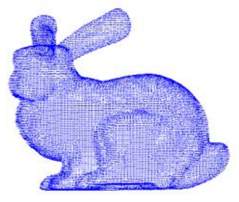
Figure 5.11: Progressive function

5.2 THE RESULT

By using the MLFF neural network algorithm, the performance of the 3D Java geometry compression increases. The compression ratio is between 5.5 and 5:10 of the original object. The noise ratio depends on the MSE, which provides minimum noise for the visual eye. Figures bellow illustrates the result for the two 3D test models before and after compression.

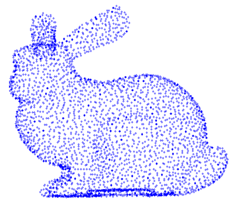
Compression using the MLFF algorithm is an adaptive one. This means that it can iteratively change the values of its parameters (i.e., the synaptic weights). These changes are made according to the learning rules. By inserting new models different than the 10 models used to train the MLFF neural network, the execution time and accuracy for any new model becomes very fast and more precise, respectively. Below are the figures and tables that show the result.

In the bunny model the vertex signal to noise ratio is 0.013 as losing visual miter. For the horse model it is 0.004 losing which is very small compared with the huge size that we are losing.



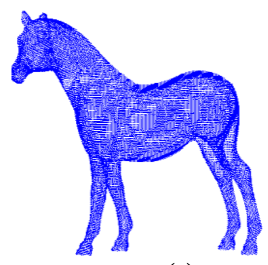
(a)

Number **of vertices** =35947



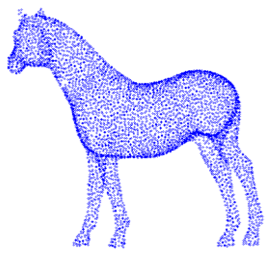
(b)

Number of vertices =3547



(a)

Number of vertices =48485

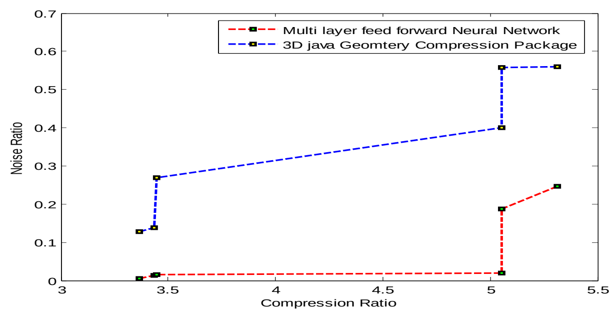


(b)

Number of vertices =4850

Figure 5.11: (a) Original model
(b) Compressed model.

Figure 5.8 shows the difference between our work with the MLFF neural network and the Java 3D geometry compression package. By fixing the compression ratio, figure 5.8 shows the relation between compression ratio and noise ratio for the same sample as that mentioned in figure 5.7. The figure indicates clearly that our compression produces low noise ratio compared with the Java 3D geometry compression package



5.3 CONCLUSION

After the Java 3D geometry compression package has been applied on the 10 created 3D objects, our MLFF algorithm has trained by example. Hence, the performance of our MLFF algorithm is much better than 3D Java package. The noise ratio that we get is very low which in turn means only a few details have been lost. The good performance of our MLFF algorithm is shown very clearly in figure 5.8 and in table 5.1 below.

Models Samples /Performance Metrics	Angel Model	Happy Model	Horse Model	Cow
Max face ratio	0.30000	0.20000	0.30000	
Edges collapsed	165917	435087	33939	
No of final edges	213321	326313	43632	
Compression ratio	3.33304	5.05457	3.33343	
Mean Square Error	0.69465	0.82077	0.79666	
signal to noise ratio	0.24736	0.20456	0.00527	
*Execution Time As CPU Time	76.74	191.65	15.35	

TABLE 5.1

REFERENCES

[1]. M. Isenburg and J. Snoeyink, " Face Fixer Compressing Polygon Meshes with Properties" , ACM Siggraph Conference Proc, pp. 263-270,2001.

[2]. 3D Object Processing: Compression, Indexing and Watermarking. Edited by J.-L. Dugelay, A. Baskurt and M. Daoudi, John Wiley & Sons, Ltd. ISBN: 978-0-470-06542-6, 2008.

[3]. H. Donald and M. Pauline Barker. 1996, Computer Graphics, C Version (2nd Edition), Publisher: Prentice Hall: ISBN-13: 978-0135309247.

[4]. A. M. Chang, and L. O. Hall 1992, The validation of fuzzy knowledge-based systems, Fuzzy Logic for the Management

of Uncertainty, L.A. Zadeh and J. Kacprzyk, eds, John Wiley, New York, pp. 589-604.

[5]. M.Deering. 1995, Geometry compression. ACM SIGGRAPH, pp. 13–20

[6]. M. Isenburg, S.Gumhold. 2003, Out-of-core compression for gigantic polygon meshes. ACM Trans. Graph. 22(3): 935-942.

[7]. M. Isenburg, P. Alliez. 2002, Compressing Polygon Mesh Geometry with Parallelogram Prediction, in Proceedings of Visualization 2002, pages 141-146.

[8]. Z.Karni and C.Gotsman. 2000, Spectral compression of mesh geometry, ACM SIGGRAPH, pp. 279–286.

[9]. D.Luebke and B.Hallen. 2001, Perceptually driven simplification for interactive rendering. Eurographics Workshop on Rendering Techniques, pp. 223–234.

[10]. M.Chow, Optimizel geometry compression for real-time rendering, In Proceedinngs of IEEE Visualization '97, Phowenix AZ, pp. 347-354, 1997

[11]. J. Rossignac. Edgebreaker: Connectivity compression for triangle meshes. IEEE Transactions on Visualization and Computer Graphics, 5(1), 1999.

[12]. D. King, J. Rossignac, and A. Szymczak. 1999, Connectivity compression for irregular quadrilateral meshes, Technical Report TR–99–36, GVU, Georgia Tech.

[13]. A.Khodakovsky, P.Alliez, M. Desbrun and P.Schröder. 2002, Near-Optimal Connectivity Encoding of 2-Manifold Polygon Meshes. Graphical Models 64(3-4): 147-168.

[14]. E. Piperakis, I. Kumazawa. 2001,3D Polygon Mesh Compression with Multi Layer Feed Forward Neural Networks, SYSTEMICS, CYBERNETICS AND INFORMATICS VOLUME 1 - NUMBER 3.

[15]. S. Haykin. 1994, Neural Networks, A Comprehensive Foundation. Macmillan College Publishing Company.

[16]. P.Maheshwari, P.Agarwal, and B. Prabhakaran. 2007, Progressive compression invariant semi-fragile watermarks for 3D meshes, in Proceedings of ACM Multimedia and Security Workshop 2007 (MM&Sec 2007), Dallas , TX , USA , pp. 245-25.

[17]. D.Luebke and B.Hallen. 2001, Perceptually driven simplification for interactive rendering. Eurographics Workshop on Rendering Techniques, pp. 223–234.

[18]. Cox, M. Miller, J.Bloom. 2001 , Digital Watermarking: Principle & Practice (The Morgan Series im Multimedia and Information Systems), ISBN-1558607145.

[19]. R.Ohbuchi, M.Nakazawa and T.Takei, Retrieving. 2003, 3D shapes based on their appearance, ACM SIGMM Workshop on Multimedia Information Retrieval, Berkeley, California, pp. 39–46.