Proceedings of the 2nd International Conference on Software Engineering and Machine Learning
DOI: 10.54254/2755-2721/74/20240488

Feature-preserving adaptive 3D scene multi-model simplification algorithm

Xin Li[1,2], Wenpei Huang[1]

[1]Southwest Jiaotong University, Chengdu, Sichuan, 611756

[2]1156355770@qq.com

Abstract. In a complex 3D scene, which typically contains multiple models with distinct feature differences, determining how to set appropriate simplification rates for each model so that the 3D scene maintains its characteristics well while achieving the desired simplification rate is a challenging task. To address this, this paper proposes a 3D scene simplification algorithm that integrates neural networks to accomplish this task. The algorithm is divided into two stages: the first stage extracts local and global features of each model in the 3D scene to form feature vectors, and the second stage combines neural networks to construct a multi-layer perceptron that predicts the simplification labels for each model in the 3D scene and calculates the simplification weights for each model based on these labels. Experiments have proven that this algorithm not only maintains a good appearance of the 3D scene but also has a smaller overall simplification error compared to the traditional QEM algorithm.

Keywords: Mesh Simplification, Feature-Preserving, Neural Networks, Multi-Model.

Chinese Library Classification Number: TP391 Document Identifier: A

1. Introduction

Current mesh simplification methods mainly fall into two categories. One category is based on geometric metrics, which reduces the complexity of the model by removing faces, edges, or vertices from the mesh model. The most representative algorithm of this type is the Quadric Error Metrics mesh simplification algorithm proposed by Garland [1], which achieves efficient model simplification through multiple point pair collapses. Subsequent researchers have made improvements on this basis. Reference [2] introduced the concept of absolute curvature of vertices into the calculation of quadric error, using the product of the vertex's absolute curvature and the quadric error measure as a new error metric. Reference [3] proposed considering the curvature of vertices and the length attributes of adjacent edges of vertices to maintain the model shape, and reference [4] designed an energy loss function, controlling the termination condition of simplification by setting a fixed energy loss threshold. The other category is based on surface clustering, which segments the model into different areas through a certain clustering algorithm, then replaces the mesh with simple geometric surfaces, such as planes, spheres, or cylinders, and finally triangulates to simplify the model [5].

In recent years, the development of machine learning and deep learning has brought new ideas and methods. Mesh simplification techniques based on machine learning have not yet been widely applied.

DOI: 10.54254/2755-2721/74/20240488

Current applications of neural networks in processing 3D models are mostly for segmentation and classification. Reference [6], by converting mesh models into point cloud data for local feature extraction and using neural networks to predict simplification labels for different areas, allows for graded simplification of different areas in the scene. However, this process is relatively cumbersome, and the simplified model is a whole, which cannot be interacted with or controlled for individual models. To address this issue, this paper proposes an adaptive simplification algorithm for multi-model 3D scenes that maintains the model's own topological structure, based on existing mesh simplification algorithms and integrating neural networks.

2. Algorithm Overview

This paper proposes an adaptive multi-model simplification algorithm for 3D scenes, with a focus on preserving the features of different 3D models. Traditional algorithms, in the process of feature extraction for models, only consider the local features of a single model, leading to the loss of characteristic information of the entire scene during multi-model simplification. This results in simple models appearing distorted or damaged after significant simplification, while complex models still have considerable room for simplification.

To address this issue, this paper builds on the feature extraction method described in reference [7] and proposes a feature-preserving adaptive multi-model simplification algorithm for 3D scenes, divided into four steps. First, it extracts the feature information of each model in the 3D scene, including the geometric and global features of the 3D models. Specifically, in this paper, the local geometric features of a model include normal vector deviation and vertex curvature variation, while the global features include surface density. Next, the extracted features from various dimensions are merged to form an overall feature vector for the model. Then, this overall feature vector is input into a neural network to predict the simplification rate labels for the model. Finally, based on the model's simplification rate labels, the model's simplification weight is calculated, and combined with a manually set overall simplification rate for the 3D scene, to determine the simplification rate for individual models.

3. Model Feature Extraction

3.1. Extraction of Normal Vector Angle Difference

The normal vector angle difference in a 3D model refers to the angle difference between the normal vectors of adjacent vertices or faces within the model. It can be used to analyze the degree of curvature and surface changes of the model. To extract the normal vector angle difference, one must first calculate the corresponding normal vector for each triangular face in the 3D model. Given a model M, this paper defines the vertices of model M as $T(M) = \{t_i; i = 1,2, \dots, n\}$, where n is the number of triangular faces in model M. The edge from t_i to t_d is defined as $S(t_i|t_d) = \{s_{i,d} = t_i - t_d; t_i, t_d \in T\}$. The operation object for constructing normal vectors in this paper is the triangular face, and the normal vector of each triangular face is defined as a unit vector perpendicular to that face. Therefore, given a triangular face $Face_i$, its normal vector is as follows.

$$NV_{Face_i} = \frac{(t_2 - t_1) \times (t_3 - t_1)}{\|(t_2 - t_1) \times (t_3 - t_1)\|} = \frac{S(t_2|t_1) \times S(t_3|t_1)}{\|S(t_2|t_1) \times S(t_3|t_1)\|} \quad (1)$$

Where, t_1,t_2,t_3 are the three vertices of the triangular face.

After calculating the normal vector for each triangular face of the model, the next step is to calculate the angle difference between the normal vectors of each triangular face. This paper uses the cosine function to calculate the angle difference, which is the dot product of the normal vectors of two faces divided by the product of the magnitudes of the two normal vectors, as shown in the following formula.

$$cos\left(NV_{Face_1} \middle| NV_{Face_2}\right) = \frac{dot(NV_{Face_1}, NV_{Face_2})}{(|NV_{Face_1}| \times |NV_{Face_2}|)} \quad (2)$$

DOI: 10.54254/2755-2721/74/20240488

Where $cos(NV_{Face_1}|NV_{Face_2})$ represents the angle difference between the normal vectors NV_{Face_1} and NV_{Face_1}, $dot(NV_{Face_1}, NV_{Face_2})$ represents the dot product of the two normal vectors, and $|NV_{Face_1}|$ represents the magnitude of the normal vector NV_{Face_1}.

After obtaining the angle differences of the normal vectors, the final step is to calculate the statistical characteristics of these normal vector angle differences, ultimately resulting in a 200-dimensional feature vector containing statistical information about the model's normal vector angle differences, completing the extraction of this feature.

3.2. Extraction of Model Curvature Variation Feature

The curvature variation feature of a 3D model describes the curvature changes at different positions on the model's surface. This paper extracts this feature by calculating the Gaussian curvature for each vertex of the 3D model, namely, by computing the Gaussian curvature of a vertex through the weighted average of the curvatures of triangular faces within the neighborhood of the vertex. For each triangular face in the neighborhood, the Gaussian curvature of the face is calculated based on the sum of the interior angles of the triangle, as shown in the following formula.

$$K_i = \frac{(2\pi - \sum \theta_i)}{A_i} \quad i = 1,2,\dots,n \tag{3}$$

Where θ_i is the sum of the internal angles of the triangular face, A_i is the area of the triangular face, and n is the number of faces in the vertex's neighborhood.

After obtaining the curvature of each triangular face in the vertex's neighborhood, the Gaussian curvature of the target vertex can be calculated through the weighted average method, as shown in the following formula.

$$K_V = \frac{\sum(K_i \times A_i)}{A_V} \quad i = 1,2,\dots,n \tag{4}$$

Where K_V represents the Gaussian curvature of vertex V, A_V represents the total area of the neighborhood around vertex V, K_i is the curvature of a certain triangular face within the neighborhood, A_i is the area of that triangular face, and n is the number of triangular faces in the neighborhood.

After obtaining the Gaussian curvature of all vertices, calculate the statistical characteristics of these Gaussian curvatures to describe the variation in vertex curvature. Ultimately, a 200-dimensional feature vector containing statistical information about the model's vertex curvature variation is obtained, completing the extraction of this feature.

3.3. Extraction of Model Face Density

The extraction of the face density feature considers the potential contribution of the model to the rendering results. Models with a higher density of faces are more refined and can better display feature details, but they can reduce rendering efficiency. Therefore, when simplifying the scene, models with a higher face density should have a higher simplification weight.

The method adopted in this paper to calculate face density is to use the ratio of the number of model faces to the sum of the number of orthographic projection pixels of the model in the three directions of its bounding box, as specified in the following formula:

$$\rho = Sum/(\rho_1 + \rho_2 + \rho_3) \tag{7}$$

In the formula, Sum represents the total number of model faces, ρ_1, ρ_2, ρ_3 represent the number of orthographic projection pixels in the three directions, respectively, and ρ is the face density.

DOI: 10.54254/2755-2721/74/20240488

4. MLP (Multi-Layer Perceptron)

4.1. Definition of Simplification Labels

After extracting the feature vectors of 3D models, this paper uses supervised learning to train a neural network with these feature vectors to predict the simplification labels of models and calculate the corresponding simplification weights. The categories of labels defined in this paper, along with their corresponding simplification weights and the volume of training data, are shown in Table 1.

Table 1. Training Data Overview

Initial Simplification Weight Range	Simplification Label	Volume of Training Data
[0.0.5)	4	10478
[0.5.0.6)	5	9321
[0.6.0.7)	6	10052
[0.7.0.8)	7	10137
[0.8.0.9)	8	10982
[0.9.1.0]	9	10662

It's important to note that the initial simplification weights corresponding to the simplification labels in the table above do not represent the final simplification rate of individual models. The simplification rate for individual models in this paper is calculated based on the simplification weight labels.

4.2. Design of the Classification Network

With the training data organized, this paper employs an MLP network model, which includes 5 hidden layers with neuron counts of 1000, 750, 500, 200, and 50, respectively. The input layer neuron count is the same as the feature vector dimension, which is 402. The output layer neuron count matches the number of categories, which is 6. The model uses the ReLU activation function. The loss function utilizes the multi-class average cross-entropy loss function, as shown in the following formula.

$$Loss = -\sum_{i=0}^{m} y_{true}^{i} \times \log\left(y_{prob}^{i}\right)/m \tag{10}$$

Where m represents the number of models to be classified inputted into the network, y_{true}^{i} and y_{prob}^{i} respectively represent the true label and the predicted label of model i.

4.3. Calculation of Simplification Weight

The simplification weight for an individual model is calculated based on the simplification rate labels provided by the MLP, as follows.

First, sum all the individual model simplification rate labels obtained from the MLP, then calculate the ratio of this sum to the simplification rate label of the current model. The result of this ratio is used as the current model's simplification weight. This weight is then multiplied by the total number of points that need to be simplified in the entire model, resulting in the number of points that need to be simplified in the current model. This is formalized in the following formula.

$$Q = A \times Sim \times \frac{label_0}{\sum label_i} \tag{11}$$

Where A represents the total number of points in the model, Sim represents the overall simplification rate, $\sum label_i$ represents the sum of all individual model simplification rate labels, and $label_0$ represents the simplification rate label of the current model.

DOI: 10.54254/2755-2721/74/20240488

5. Experimental Results and Analysis

5.1. Simplification Error Statistics Comparison

To verify the effectiveness of the algorithm proposed in this paper in actual 3D model simplification tasks, 3D models in batches of 50, 100, and 200 were randomly selected. These models were simplified using the algorithm from reference [7], the QEM algorithm, and the algorithm proposed in this paper, respectively. The simplification errors of the 3D models simplified by the three methods were calculated in comparison to their original models to measure the loss of features during the simplification process. The paper uses the Euclidean distance method to calculate the simplification error before and after simplification, and the experimental results are shown in Table 2.

Table 2. Comparison of Simplification Error Statistics

Number of Models	Simplification Rate	Average Simplification Error		
		QEM Algorithm	Algorithm from [7]	Algorithm from This Paper
50	50%	0.005867	0.004346	0.004339
	80%	0.083277	0.037823	0.021345
	95%	0.654871	0.845123	0.154841
100	50%	0.006157	0.006003	0.005834
	80%	0.153687	0.098524	0.054875
	95%	0.854214	0.832154	0.548714
200	50%	0.006985	0.006433	0.006100
	80%	0.254871	0.156487	0.189452
	95%	0.915421	0.845712	0.548712

5.2. Simplification Effect Display

To more intuitively demonstrate the effect of the algorithm proposed in this paper, a 3D scene model as shown in Figure 1 was selected. Different simplification rates were set, and after simplification using the QEM algorithm, the algorithm from [7], and the algorithm from this paper, the effects were displayed. As shown in Figure 2, at a 50% simplification rate, it is apparent that the simplification degree of the algorithm from this paper is greater compared to other algorithms. Figures 3 and 4 show that at 80% and 95% simplification rates, models simplified by other algorithms experienced distortion and damage issues, whereas the algorithm from this paper maintained the overall detail of the model more effectively.

DOI: 10.54254/2755-2721/74/20240488

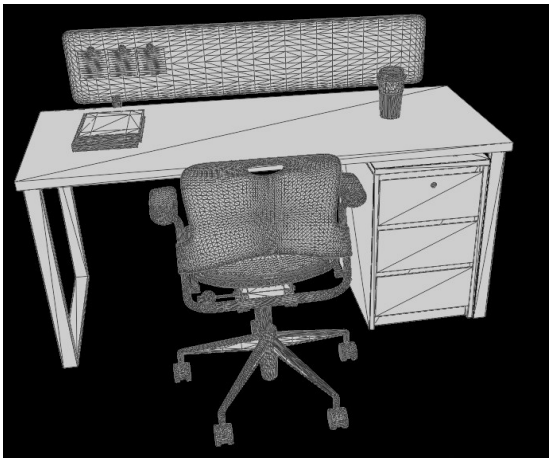
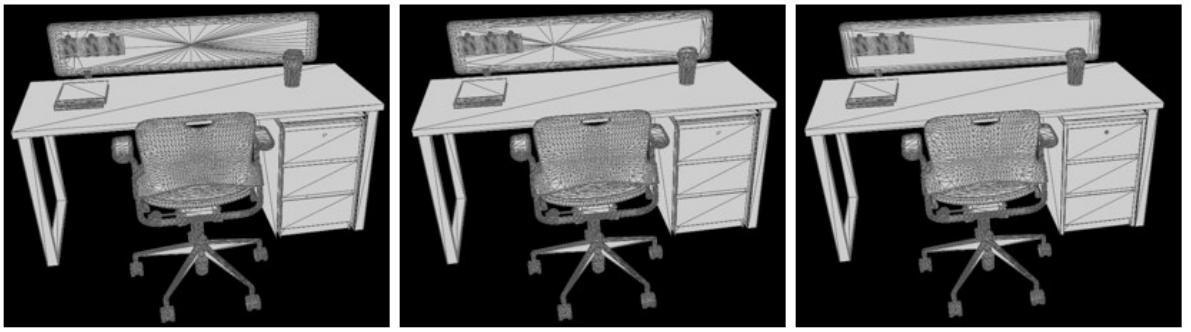
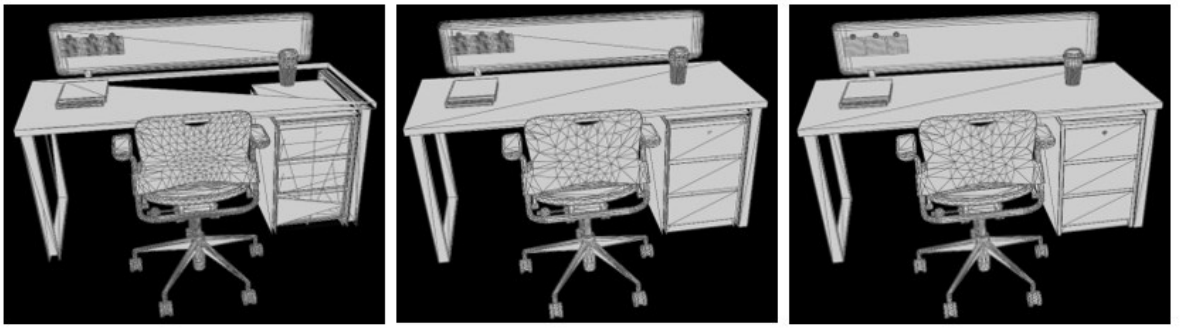


Figure 1. Comparative experiment 3D model diagram



(a) QEM Algorithm (b) Algorithm from [7] (c) Algorithm from This Paper

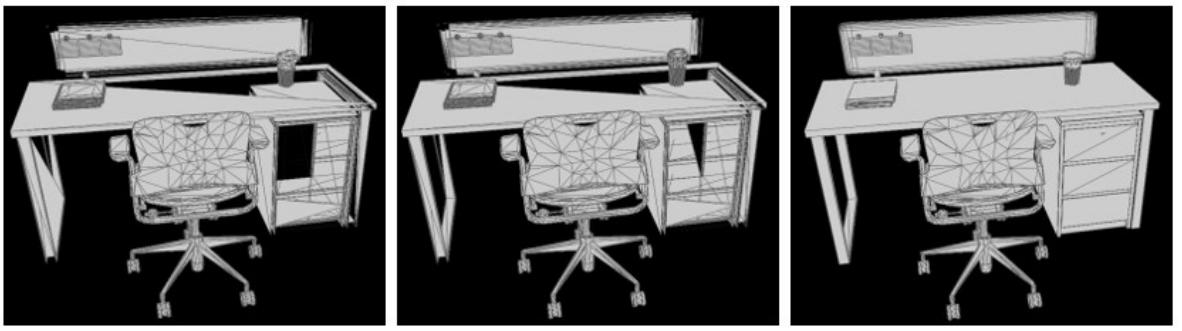
Figure 2. Comparison of the effects of three algorithms at a total simplification rate of 50%



(a) QEM Algorithm (b) Algorithm from [7] (c) Algorithm from This Paper

Figure 3. Comparison of the effects of three algorithms at a total simplification rate of 80%

DOI: 10.54254/2755-2721/74/20240488



(a) QEM Algorithm (b) Algorithm from [7] (c) Algorithm from This Paper

Figure 4. Comparison of the effects of three algorithms at a total simplification rate of 95%

6. Conclusion

This paper proposes a feature-preserving adaptive multi-model simplification algorithm for 3D scenes that ensures a smaller simplification error for the overall model by calculating the simplification weights of each individual model within the total model. Initially, the feature vector of a single model is extracted, including the features of normal vector deviation, curvature variation, and face density, and these three-dimensional features are merged into a comprehensive feature vector for the individual model. Subsequently, this feature vector is input into an MLP for classification of simplification rate labels. Finally, based on these labels, the simplification weights of different individual models are calculated, representing their proportion of simplification within the overall simplification rate. Experimental results demonstrate that, compared to previous simplification algorithms, the algorithm presented in this paper can effectively preserve the global features of 3D scenes. While ensuring the simplification rate of multi-model 3D scenes, it can reduce damage and distortion of 3D models, exhibiting advanced and applicable characteristics compared to previous algorithms.

References

[1] Garland, Michael, Heckbert, Paul S. Surface Simplification Using Quadric Error Metrics[C]// Conference on Computer Graphics & Interactive Techniques. 1997:209-216.

[2] Li, L., & He, M. Y. (2010). Mesh simplification based on feature preservation and quadratic error metrics. Journal of Northwest University: Natural Science Edition, 40(03), 419-422.

[3] Li Y B, Zhu Q. A new mesh simplification algorithm based on quadric error metrics[C] //Proceedings of International Conference on Advanced Computer Theory and Engineering. Los Alamitos: IEEE Computer Society Press, 2008.

[4] Pan Z G, Xian C H, Jin S, et al. Progressive furniture model decimation with texture preservation[J]. Journal of Computer Science and Technology, 2019, 34(6): 1258-1268.

[5] Liang, C. P., Yin, J., Wu, J., et al. (2020). A survey of clustering analysis techniques in 3D mesh segmentation. Journal of Computer-Aided Design and Computer Graphics, 32(4), 680-692.

[6] Yang, Y., Xian, C. H., & Li, G. Q. (2020). An adaptive simplification rate mesh simplification algorithm combining local area features. Journal of Computer-Aided Design and Computer Graphics, 32(6), 857-864.

[7] Zhu, T. X., Yan, F. T., & Shi, Z. C. (2023). Feature-preserving region-based hierarchical mesh simplification algorithm. Journal of Graphics, 44(3), 570-578.