

Improvements for Mesh Optimization Algorithm

Muhammet Mustafa Özdal and Uğur Güdükbay

Bilkent University,
Department of Computer Engineering,
06533 Bilkent, Ankara Turkey
{ozdal, gudukbay}@cs.bilkent.edu.tr

Abstract. To reduce the complexity of polygonal models used in computer graphics, simplification techniques are applied on these models. Mesh Optimization is one of these techniques that requires a high running time, but gives good results. This paper proposes some modifications on this method to improve the performance and presents the results of performance experiments on different meshes.

1 Introduction

The current technology in computer graphics allows the generation of complex models with millions of faces. However, the improvements in hardware performance usually can not compete with the increase in data complexity. So it is required to reduce the complexity of models by using efficient simplification methods. For this aim, data is represented with meshes and simplification techniques are applied on them.

There are different strategies used for this aim. A survey of the different techniques used for simplification of triangular meshes can be found in [1]. The analysis of these techniques show that they effectively differ in two criteria: *running times* and *data fidelity*. It is generally the case that, if the resulting simplified mesh of a method is of higher quality, this method requires more running time, and vice versa.

Mesh Optimization [2] is one of the techniques that requires a high running time, but gives good results. This paper proposes some improvements for this method and shows their effects on different meshes in terms of quality and performance.

The rest of the paper is organized as follows: First, Mesh Optimization algorithm is explained. Next, the implementation details used in the algorithm to make the program run faster are discussed. Then, some modifications to improve the performance are proposed. Then, the results obtained using the improved version and some statistics for the execution times are presented. Finally, conclusions are given.

2 Polygonal Model Simplification Using Energy Optimization

In their original article, Hoppe et al. give a method to solve the following problem [2]:

“Given a set of data points scattered in three dimensions and an initial triangular mesh M_0 , produce a mesh M , of the same topological type as M_0 , that fits the data well and has a small number of vertices.”

The approach used to solve this problem is to minimize an energy function such that the number of edges is minimized and proximity of the simplified mesh to the original data points is maximized.

2.1 Mesh Representation

A mesh is formally represented with a pair: (K, V) . K is called simplicial complex and represents the connectivity of the vertices, edges and faces; V is called the vertex set and represents the vertex positions in the set.

The simplicial complex determines the topological type of the mesh. It consists of a set of vertices and their subsets (edges and faces). To form the topological realization of a mesh, first the vertices $\{1, \dots, m\}$ are identified as the basis vectors $\{e_1, \dots, e_m\}$ of R^m . Then according to the edges and faces in the simplicial complex, vectors called the *barycentric coordinate vectors* are formed by using these basis vectors. So the topological realization of the mesh (denoted as $|K|$) is obtained.

To obtain the geometric realization, the set of vertex positions $V = \{v_1, \dots, v_m\}$ is used to form a linear map Φ , such that the i -th standard basis vector $e_i \in R^m$ is sent to $v_i \in R^3$. By using this map, the geometric realization (denoted as $\Phi(|K|)$) of the given mesh is obtained from its topological realization.

2.2 Energy Function

The mesh optimization is realized by minimizing the following energy function:

$$E(K, V) = E_{dist}(K, V) + E_{rep}(K) + E_{spring}(K, V)$$

where:

$$E_{dist}(K, V) = \sum_{i=1}^n d^2(x_i, \Phi_V(|K|))$$

$$E_{rep} = c_{rep} m$$

$$E_{spring}(K, V) = \sum_{\{j,k\} \in K} \kappa \|v_j - v_k\|^2$$

E_{dist} is the distance energy defined as the sum of squared distances from each data point to the nearest face of the mesh. The aim for using this term is to penalise the deviation of the mesh from the original data points. E_{rep} is simply the number of vertices multiplied by a parameter. This term incorporates the aim of minimizing the number of vertices while trying to minimize the energy function.

Note that the terms E_{dist} and E_{rep} compete with each other. If the number of vertices is decreased, E_{rep} decreases and E_{dist} increases, and vice versa. The user-specifiable parameter c_{rep} is used to control the trade-off between the degree of mesh simplification and the mesh quality (i.e., if c_{rep} is small, the aim of having a mesh of high quality dominates the aim of having a mesh with a small number of vertices).

To define E_{spring} , a spring of rest length zero and spring constant κ is placed on each edge of the mesh. E_{spring} is the sum of the energies of each such spring. The aim is to regularize the optimization process. Note that κ is not a user-specifiable parameter. As will be discussed later, the value of κ is decreased during the optimization process.

2.3 An Overview of the Algorithm

The pseudocode for the algorithm used is given in Fig. 1. The problem of mesh optimization is divided into two subproblems: inner minimization that minimizes the

energy function for a fixed simplicial complex and an outer minimization that changes the simplicial complex while minimizing the energy function.

```

OptimizeMesh (K0, V0)
{
    K = K0
    V = OptimizeVertexPositions(K0, V0)

    // the outer minimization problem
    repeat
    {
        (K', V') = GenerateLegalMove(K, V)
        V' = OptimizeVertexPositions(K', V')
        if E(K', V') < E(K, V) then
            (K, V) = (K', V')
        endif
    }
    until convergence
    return (K, V)
}

// inner minimization problem
OptimizeVertexPositions(K, V)
{
    repeat
    {
        B = ProjectPoints(K, V)
        V = ImproveVertexPositions(K, B)
    }
    until convergence
    return V
}

GenerateLegalMove(K, V)
{
    Select a legal move K → K'
    Locally modify V to obtain V' appropriate for K'
    return (K', V')
}

```

Fig. 1. Mesh Optimization Algorithm based on energy minimization [2]

2.3.1 The Inner Minimization Problem

The inner minimization is performed by the `OptimizeVertexPositions(K, V)` procedure. This procedure takes the simplicial complex and the initial vertex positions as input and optimizes the energy function by changing the vertex positions V .

Since E_{rep} depends only on the number of vertices, the problem is to minimize $E_{dist} + E_{spring}$. Calculating E_{spring} is trivial, since it uses the vertex positions and the constant κ :

$$E_{spring}(K, V) = \sum_{\{j, k\} \in K} \kappa \|v_j - v_k\|^2$$

However, calculating the distance energy E_{dist} is not so trivial. As described before, the distance energy is the sum of all the distances of the original data points to the nearest face of the mesh. To calculate E_{dist} it is required to find the nearest face for

each data point and calculate the distance between them. The sum of the distances for all data points gives E_{dist} .

To perform the minimization, two subprocedures are used in the pseudocode: `ProjectPoints(K, V)` and `ImproveVertexPositions(K, B)`. The first one finds the nearest points to each point and the corresponding barycentric vectors and the second one optimizes the positions of the vertices by using the barycentric vectors.

2.3.2 The Outer Minimization Problem

The outer minimization is performed iteratively. First, the simplicial complex is changed a little by generating a random legal move. Then, inner minimization process is performed for this new simplicial complex. After this, it is checked whether the energy of this new simplicial complex is less than the initial simplicial complex before the legal move. If it is, then the change in the simplicial complex is accepted, otherwise the change is cancelled. Afterwards, another legal move is generated and the above process is repeated. This goes on until the energy function does not decrease with any legal move.

A legal move is defined as one of the transformations: *edge collapse*, *edge split*, and *edge swap* on K that leaves the topological type of K unchanged. Fig. 2 illustrates these transformations on an example mesh. Here, the edge between i and j is the edge on which the transformations are applied. The edge collapse operation removes the vertices i and j and creates another vertex. Edge split operation puts another vertex between i and j . Edge swap operation removes the edge between i and j and creates an edge between another pair of vertices.

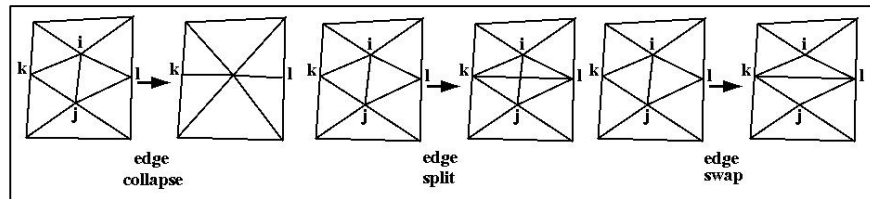


Fig. 2. Legal Transformations

In the beginning of an execution of the program, a candidate set of edges that includes all the edges of the mesh is created. Then, an edge is randomly removed from the candidate set. It is checked whether collapsing this edge would be legal and would decrease the energy. If not, edge swap and edge split operations are checked in this order. If one of these operations is legal and decreases the total energy, this operation is accepted and the candidate set is updated by adding all the neighboring edges. If none of them is accepted, another edge from the candidate set is selected and the same operations will follow. Note that these operations go on until some convergence criterion (i.e. the candidate set is empty) is satisfied.

It was mentioned that while considering a legal move, the inner minimization process (i.e. holding the simplicial complex fixed and changing the vertex positions) is applied for the new simplicial complex. To make the execution faster, the inner optimization is performed not for all the vertices of the mesh, but only for those

effected by the edge operation (i.e. the new vertex in the edge collapse and edge split operations; and the vertices of the new edge in the edge swap operation).

2.4 Setting the Spring Constant

In the beginning of the program, the spring constant κ is assigned a large value. Then, the operations in the algorithm described before are performed and a simplified mesh is obtained. Then, the spring constant is decreased and the same operations are performed for the simplified mesh. This is repeated for a small number of times and the final simplified mesh is obtained.

3 The Improvements

In the experiments that were performed by using the program, we observed that there are too many unsuccessful operations occurring. Reducing these unsuccessful operations seems to be a good improvement. The following subsections describe the methods proposed for this aim.

3.1 Limiting the Transformations to Edge Collapse Operations

The first improvement can be achieved by limiting the edge operations to edge collapse only. As stated in [3], the other two operations (edge split and edge swap) may be useful for surface reconstruction, but they are not essential for mesh simplification. The improvements in mesh quality by using these operations usually do not compensate the increase in run time.

3.2 Reducing the Number of Unsuccessful Operations in the Later Stages

Performance can be improved by reducing the number of unsuccessful operations in the later stages. For this, an array of size equal to the number of initial edges and that has two fields is used. The array index determines the edge; one field holds the change in E_{dist} (ΔE_{dist}) and the other holds the change in E_{dist} / κ if this edge is collapsed.

Initially, the array is empty. In the first iteration, that is for the first spring constant, while the unsuccessful edge operations cause the edges be removed from the candidate set, the fields of this array are filled. Most of the unsuccessful operations are due to positive energy. This means that before removing an edge from the candidate set, the energy change it would cause is calculated for almost all the edges. So filling this array brings almost no additional workload to the program.

In the later iterations for different spring constants, this array is used to determine the initial candidate set. For each edge, the following operation is performed: the first field of the array is added to the second field times the new spring constant κ . If the result of this operation is greater than zero, then this edge is added to the candidate set. In this way, most of the unsuccessful operations in the later iterations are avoided.

We do not need to update our array at each successful edge collapse because the edges that need to be updated in such an operation will be added into the candidate set. The operations for a fixed spring constant will continue until the candidate set becomes empty. So the edges that need to be updated will definitely be updated before the next iteration for which this array will be used.

3.3 Choosing the Better Edge Collapse Operations

The original algorithm uses random descent as the selection strategy. That is, an edge is selected randomly from the candidate set. If collapsing this edge is legal and decreases the total energy, then it is collapsed immediately. Instead of using this method, it is possible to use steepest descent strategy for selection. Lindstrom and Turk have used this strategy for selection in [4]. They define a cost function and calculate the cost of each edge collapse operation. They use a priority queue and always select the operation that has the minimum cost. Although this strategy is expected to give better results, the usage of a priority queue increases the execution time considerably. So, we use another technique for this aim. It is to use multiple iterations for a spring constant instead of one iteration and not to add the neighboring edges into the candidate set when an edge collapse occurs in the current iteration.

In the original algorithm, when an edge is collapsed, the neighboring edges are added into the candidate set and the iteration continues until the candidate set is empty. Assume that an edge collapse is tried and it is unsuccessful. Assume further that an edge collapse is performed in the neighborhood and this edge is to be added to the candidate set again. By intuition, we can say that the probability that the collapse of this edge will decrease the energy function is less than that of other edges that have not been tried yet. So it is logical to try those other edges before this edge. This can be achieved by not adding this edge into the candidate set in the current iteration. The technique used is as follows:

- After an edge collapse, mark the edges that would be added to the candidate set.
- Then in the next iteration (for the same spring constant) add the marked edges into the initial candidate set and perform the same operations.
- Perform new iterations until there is no marked edge.

This technique resembles the steepest descent technique a bit. However, instead of choosing the “best” edge, it helps choosing a “better” edge.

4 Experimental Results

The three improvements explained were implemented and the results were observed. The first improvement (removing edge split and edge swap operations) degraded the mesh quality by a very small factor, but improved the performance. The second and third techniques improved performance without degrading the mesh quality.

The results obtained by using the original program and the improved version are given in Tables 1 and 2 respectively. Here E_{dist} represents the distance energy term. Note that, a small value of E_{dist} indicates that the quality of the simplified mesh is high. E_{total} represents the total energy that the program tries to minimize. The original program and the improved version have been executed for different models by using the same parameter values. A Sun Ultra Enterprise 4000 computer with eight 256MHz processors and 1Gbyte memory has been used. The qualities of the simplified models for the two versions of the program are very close to each other. However, the improved version performs about twice faster compared to the original program. Simplification examples obtained by using the original program and the improved version are given in Figs. 3 and 4.

Table 1. Experimental results obtained by using the original code

Model	Original Model				Simplified Model				Proc. Time (secs)
	No. vertices	No. faces	E_{dist}	E_{total}	No. vertices	No. faces	E_{dist}	E_{total}	
Hypersheet	2032	3832	0.0	0.2999	288	531	0.0119	0.0513	224
Teapot	7704	15382	0.0	0.8124	245	487	0.0100	0.0355	470
Mannequin	7834	15543	0.0	0.0636	1532	3018	0.0007	0.0023	433
Oilpump	22741	45478	0.0	2.3526	505	1006	0.0257	0.0762	1107
Budha	543652	1087716	0.0	54.4111	2560	5532	0.1026	0.3586	25125

Table 2. Experimental results obtained by using the optimized code

Model	Original Model				Simplified Model				Proc. Time (secs)
	No. vertices	No. faces	E_{dist}	E_{total}	No. vertices	No. faces	E_{dist}	E_{total}	
Hypersheet	2032	3832	0.0	0.2999	279	513	0.0134	0.0519	103
Teapot	7704	15382	0.0	0.8124	227	452	0.0121	0.0356	235
Mannequin	7834	15543	0.0	0.0636	1518	2991	0.0008	0.0024	253
Oilpump	22741	45478	0.0	2.3526	508	1012	0.0281	0.0789	530
Budha	543652	1087716	0.0	54.4111	2810	6032	0.1186	0.3996	13509

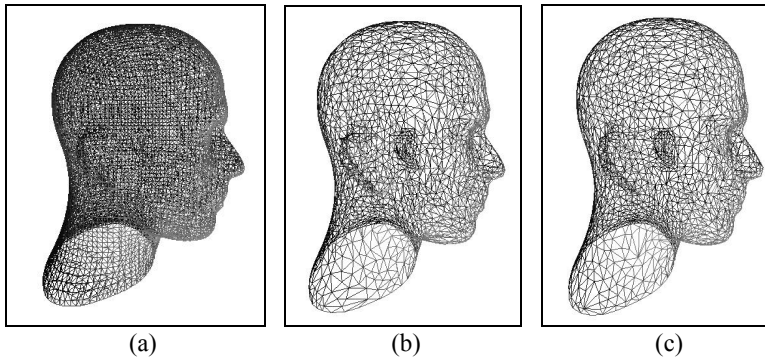


Fig. 3. Simplification of Mannequin Model. (a) Original (b) Simplified with original algorithm (c) Simplified with optimized algorithm

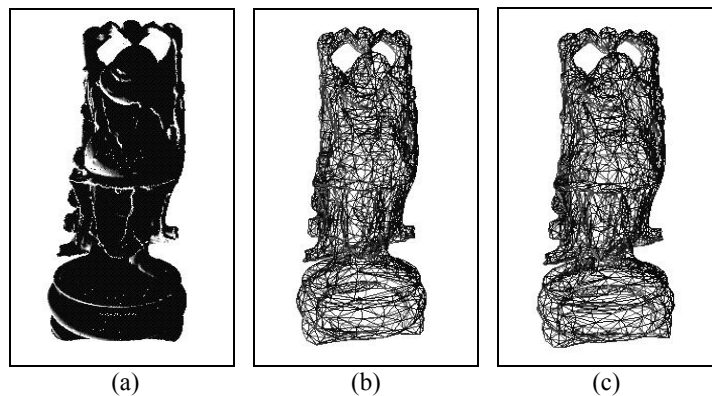


Fig. 4. Simplification of Happy Budha Model. (a) Original (b) Simplified with original algorithm (c) Simplified with optimized algorithm

5 Conclusions

This paper proposes some improvements to Mesh Optimization algorithm [2]. The method used is simply trying to minimize an energy function so that the number of vertices are reduced while providing fidelity to the original data as much as possible. The problem of mesh minimization is divided into two subproblems: inner minimization and outer minimization. In this paper, we have proposed improvements for the outer minimization subproblem.

The first improvement proposed is to eliminate the edge operations: edge swap and edge split. As indicated in [3], these two operations are especially useful for surface reconstruction applications, but have small effects on mesh simplification.

The second improvement is to decrease the size of candidate set in the beginning of later iterations. The idea is to store the calculated value of energy function for each edge and use this information to eliminate some elements of the candidate set for the next spring constant iteration.

The third improvement we proposed is to use multiple iterations for a spring constant, and to add the edges in the neighborhood of a collapsed edge to the candidate set of the next iteration instead of current one. This technique resembles the steepest descent technique, but it chooses better edges instead of the best edge.

After implementing these improvements, the new version of the program was tested on several models. The results of the experiments show that these improvements make the program run about twice faster without degrading the mesh quality.

Acknowledgments

This project is supported by an equipment grant from Turkish Scientific and Technical Research Council (TÜBİTAK) with grant no. EEEAG 198E018. Happy Buddha model is obtained from Stanford University, Computer Graphics Laboratory. Other models and the original simplification software are obtained from Hugues Hoppe.

References

1. P. Cignoni, C. Montani, and R. Scopigno, "A Comparison of Mesh Simplification Algorithms", *Computers & Graphics*, Vol. 22, No.1, pp. 37-54, 1998.
2. H. Hoppe, T. DeRose, T. Duchamp, J. McDonald and W. Stuetzle, "Mesh Optimization", *Proc. of SIGGRAPH'93*, pp. 19-26, 1993.
3. Hoppe, H., "Progressive Meshes", *Proc. of SIGGRAPH'96*, pp. 99-108, August 1996.
4. P. Lindstrom and G. Turk, "Fast and Memory Efficient Polygonal Simplification", *Proc. of IEEE Visualization'98*, pp. 279-286, October 1998.