A New Mesh Simplification Algorithm Combining Half-edge Data Structure with Modified Quadric Error Metric

Li Guangming, Tian Jie, Zhao Mingchang, He Huiguang and Zhang Xiaopeng
AI LAB, Institute of Automation, the Chinese Academy of Sciences, Beijing, 100080, China
Email: {guangming.li, jie.tian, mingchang.zhao, huiguang.he, xiaopeng.zhang} @mail.ia.ac.cn

Abstract

This paper presents a fast mesh simplification algorithm that combined the half-edge data structure with modified quadric error metric (QEM). When half-edge structure is used, the adjacency queries between components of the mesh, such as vertices, faces and edges, can be quickly achieved and thus the run time is reduced remarkably. Furthermore, with the modified quadric error metric, the quality of the simplified meshes for a certain kind of 3D medical models whose normal vectors are computed through voxel gradient during reconstruction can be greatly improved. The experimental results illustrate the efficiency of the algorithm.

1. Introduction

In computer graphics, objects are often represented by triangle mesh. With the advances in data acquisition and the development of modeling techniques, 3D models become more and more complex. Although the rendering capability of current graphic hardware has been improved considerably, it can't yet keep up with the growing of model size, which makes real-time rendering difficult. However, in many applications, the highly detailed polygonal models are not necessary and the rendering time is relatively more important. In this case, we should substitute simpler approximations of the original model, i.e., the original surface should be simplified.

In recent years, many effective techniques for automatic simplification have been developed. At Siggraph'92 Turk [1] presented an algorithm based on re-tiling. At the same time, Schroeder [2] described a simplification method called triangle decimation. Later Rossignac [3] used vertex clustering to simplify meshes. In 1996, Hoppe [4] proposed an algorithm called Progressive Meshes to construct LOD (Level-Of-Detail) model. Garland and Heckbert [5] described a new algorithm based on quadric error metric (QEM) for mesh simplification in 1997. Moreover, there exist many other methods such as coplanar facets merging [6], wavelet-based approaches [7] etc.

In this paper, we combine the half-edge data structure with modified QEM algorithm to simplify models. The advantages of half-edge structure in adjacency queries can provide significant savings in running times and by modifying the construction of vertex quadrics, we can improve the quality of the simplified surfaces for a certain kind of 3D medical models that QEM can't process well.

Section 2 introduces the details of our algorithm. The experimental results are given in section 3 and we conclude in section 4.

2. Details of the algorithm

2.1. Data structure

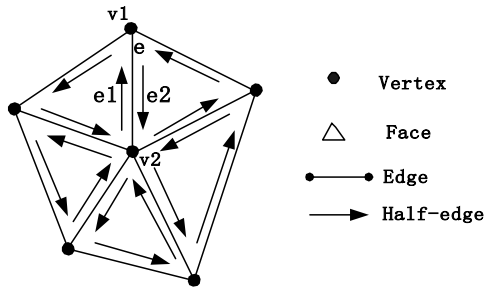


Figure 1. A mesh represented by half-edge

As the name implies, a half-edge is a half of an edge and is constructed by splitting the edge into two directed half edges. The basic concept of half-edge is illustrated in fig.1. The edge *e* is divided into two directed halves: *e1, e2*. We call *e1, e2* a pair. The vertex from which the half-edge emanates is called its *start point*, and the vertex at the end of the half-edge is called its *end point*. For example, *v2* is the start point of *e1* and *v1* is its end point. The edge *e* has two adjacent faces. Each one has three half-

Supported by the National Natural Science Foundation of China under Grant Nos. 60071002,60072007,69931010,60172057

edges. When using this structure, we store half-edges instead of edges.

According to different requirements, different vertex lists, face lists and half-edge lists can be constructed because of the flexibility of half-edge structure. In our implementation, we can get all information needed for mesh simplification by adopting several simple structures shown in fig.2 and the updates of the adjacency of vertices, faces and half-edges can be easily achieved.

As we can see in fig.1, the half-edges that border a face form a circular linked list. This list can either be oriented clockwise or counter-clockwise around the face as long as the same convention is used throughout. Each of the half-edges in the loop stores four pointers: its start point, its pair, the face it borders and its next half-edge around the face. In C it looks like fig.2 (a).

In vertex list, it is not necessary to record the adjacent faces or edges of each vertex. Besides the essential coordinates and normal vector, one vertex only needs to store a pointer to one of the half-edges whose start points are this vertex. There may be several half-edges for us to choose, but we only need one and it doesn't matter which one it is. The vertex structure looks like fig.2 (b).

A face in the face list only needs to store a pointer to one of the half-edges that borders it. It is similar to the vertex structure in that although there are three half-edges bordering each face, we only need one and it doesn't matter which one. The face structure looks like fig.2 (c).

```
Struct HalfEdge {
 Vertex *vert;
 HalfEdge *pair;
 HalfEdge *next;
 Face *face;
};
```
(a)

```
Struct Vertex {
 float vcoord[3];
 float ncoord[3];
 HalfEdge *he;
};
```
(b)

```
Struct Face {
 HalfEdge *he;
};
```
(c)

Figure 2. Structures of half-edge, vertex and face

With the structures mentioned above, the adjacency queries can be easily achieved. For example, the faces and vertices that neighbour a half-edge can be found with the codes in fig.3 (a), and the adjacent edges, points and faces of a vertex can be obtained through the simple *do…while* loop illustrated in fig.3 (b).

```
Vertex *vert1=he->vert;
Vertex *vert2=
        he->pair->vert;
Face *face1=he->face;
Face *face2=
        he->pair->face;
```
(a)

```
HalfEdge *the=Vert->he;
Vertex *Vert;Face *face;
do{
   the=the->pair->next;
   Vert=the->next->vert;
   face=the->face;
}while(the!=vert->he);
```
(b)

Figure 3. Examples of adjacency queries using half-edge structure

2.2. Half-edge collapse

The half-edge collapse is similar to edge collapse. The only difference lies in that for half-edge collapse, the start point is pulled into the end point. In other words, the new vertex after the half-edge collapse is the same as the end point of this half-edge. If fact, the half-edge collapse can be viewed as the general edge collapse without locating new vertex. It can also be considered as the vertex removal operator without re-triangulation.

2.3. The error metric

For general meshes whose normal vectors are computed through the normal vectors of their neighbouring faces, we can get acceptable results by using the QEM presented by Garland. But for a certain kind of 3D medical models whose normal vectors are computed from voxel gradient during reconstruction, the simplified mesh is unacceptable if we use Garland's QEM directly. To solve this problem, we integrated the accurate normal information to vertex quadrics by constructing a virtual plane. In section 2.3.1, we briefly describe the quadric error metric of Garland. In section 2.3.2, we introduce our improvements.

2.3.1 Overview of QEM. Given a plane that is represented by the standard form $n^Tv+d=0$ where $n=[a,b,c]^T$ is a unit normal (i.e., $a^2+b^2+c^2=1$) and d is a scalar constant, the squared distance of a vertex $v=[x,y,z]^T$ from the plane is given by the equation $D^2(v)=(n^Tv+d)^2=v^T(nn^T)v+2(dn)^Tv+d^2$. This equation is equivalent to the formula $D^2(v)=v^TQv=Q(v)$, where $v=[x,y,z,1]^T$ and the quadric Q is treated as a homogeneous matrix $Q=\begin{bmatrix} A & b \\ b^T & c \end{bmatrix}$, $A=nn^T$ is a 3×3 matrix, $b=dn$ is a 3-vector, $c=d^2$ is a scalar. Using this representation, we can easily compute the sum of squared distances to a set of planes:

$$E_Q(v)=\sum_i D_i^2(v)=\sum_i v^TQ_iv=v^T(\sum_i Q_i)v=v^TQv$$

Where $Q=\sum_i Q_i$. In other words, to compute the squared distances to a set of planes, we only need one quadric that is the sum of the quadrics defined by each of the planes in the set. When contracting the edge (v_i,v_j), the quadric of the new vertex v_{new} is merely $Q_{new}=Q_i+Q_j$ and the cost of contraction, which is defined as the sum of squared distances from v_{new} to the set of planes determined by the adjacent faces of v_i *and* v_j, can be computed through the equation $Q_{new}(v_{new})=v_{new}{}^TQ_{new}v_{new}$. The point we want to find is the one that minimizes this cost.

2.3.2 Modified quadric error metric. Fig.6 (a) is a 3D medical model of human knee reconstructed by Marching Cubes. Since we add two zero slices around the first layer

and the last layer to improve the visual quality of reconstructed model (for instance, if the original datasets have 58 slices, when reconstructing, we will process 60 slices. The pixel values in the first and last slices are all zero), large amounts of flat region will occur in these two layers. If the vertex normal is the weighted sum of the normals of its adjacent faces, the visual quality of the reconstructed model will be unacceptable. We employ the method of computing voxel gradient to get more accurate normal vectors, which makes these two layers look uneven. In fact, through this approach, the information of vertex normals can be expressed more accurately and the quality of the 3D medical models can be improved greatly. But when being simplified, because QEM method selects the sum of squared distances from the point to a set of planes as its cost criterion, the flat regions mentioned above will be simplified very soon while the vertex normals remain unchanged. This results in the great difference between the simplified model and the original model as illustrated in fig.6 (b). To solve this problem, we do some modification on QEM.

Given a vertex $v=[x,y,z]^T$ with a unit normal $n=[a,b,c]^T$ that is computed through voxel gradient, we can imagine that there exists a virtual plane P on which the vertex v lies and its normal is n. In other words, P is the tangential plane of the vertex v. When constructing the quadrics of v, we not only consider the contribution of the adjacent faces of v but also consider the contribution of the virtual plane P. The following equation illustrates this idea:

$$Q_v = \sum_i Q_i + wQ_P$$

Where Q_i is the quadric of the *i-th* face around vertex v, Q_P is the quadric of the virtual plane P and w is the weight.

From section 2.3.1 we know that the quadric of a plane is determined completely by its standard representation form. For the virtual plane P, its unit normal and one of the vertices on it are known, so we can easily compute the scalar constant of the plane equation: $d= -ax-by-cz$. Then the quadric of P can be constructed and thus the quadric of vertex v can be obtained. Although this approach is very simple, it can indeed improve the visual quality of this kind of 3D medical models as fig.6 (c) shows.

In our approach, the weight w is defined by users. Our tests have proved that if w is too small, it has little influence on simplification procedure while the global results may suffer if w is too large. In our implementation, we select the number of the adjacent faces of vertex v as the weight. Experimental results show that the effect is relatively good in this case.

2.4. Boundary constraints

We partition the edges of the mesh into two classes: the *interior* edges and the *boundary* edges. The interior edge has two adjacent faces and is represented by two half-edges that form a pair. The boundary edge has only one adjacent face and is represented by one half-edge whose pair is null. Accordingly, the vertices of the model are also classified into two categories: the *interior* vertices and the *boundary* vertices. All adjacent edges of the interior vertex are interior edges while at least one of adjacent edges of the boundary vertex is boundary edge.

There are two cases that will change the boundary shape. One is that a boundary vertex is collapsed to an interior vertex, the other is that a boundary vertex is collapsed to another boundary vertex, i.e., a boundary edge is collapsed. It can be obviously seen that the former may considerably change the boundary, so we don't process it. For the second case, however, we allow them to be simplified. The reason is that when most triangles have been removed, the proportion of boundary edges will increase. If we forbid them from being simplified, we can only contract other edges of the mesh, thus although the boundary is preserved fairly well, the global result will drop back. We, during initialization, first compute a plane perpendicular to the face through the boundary edge, then form a quadric weighted with a large penalty factor for this plane and add it into the initial quadric for each of the endpoints just as Garland did. This can guarantee the boundary edges against being collapsed soon while when most triangles have been simplified, some boundary edges will be contracted properly. Since this is only the merging of boundary vertices and doesn't occur very often, the boundary can also be preserved fairly well.

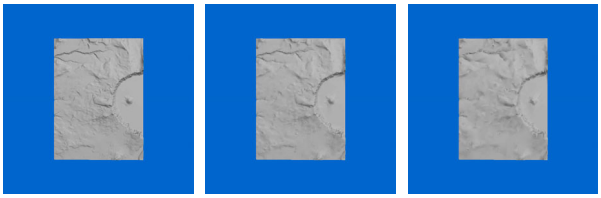
2.5. Implementation

Here it is not necessary for us to describe the details of the basic steps of our approach, for it is similar to those of methods based on general edge collapse. What we would say is that when using half-edge structure, we don't need to update the face list after every contraction and the updates of the vertex and half-edge lists are also very simple. Therefore, the computational time for decimation can be greatly reduced. Moreover, when a half-edge is collapsed, its pair will be removed too, so it is not necessary to put all half-edges into the heap that keyed on cost with the minimum at the top. We can simply allocate the size of the heap according to the number of the edges instead of half-edges. The cost of each edge is the smaller value of the costs of its two half-edges and the new vertex is the end point of the half-edge which has the smaller cost.

3. Experimental results

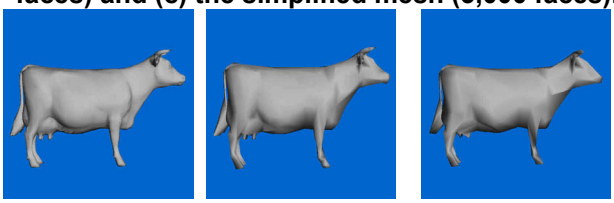
We implement our method with C++ and test it in several different datasets. All of the tests are completed on a PIII 800MHz PC with 128MB of main memory and Windows 2000. The rendering is left to standard OpenGL API using a GeForce 2 MX-400/32MB video adapter.

Figure 4, 5 and 6 demonstrate the results of our method on three models. In fig.4, the simplified model whose triangle number is 99% less than the original can still keep good features. The cow in fig.5 is a manifold model without boundaries. The major details are kept even for the approximation with 600 triangles. Fig.6 (a) is a 3D medical model of human knee with 95,936 faces reconstructed by Marching Cubes. The two simplified models with 15,000 triangles using QEM and our method are shown in (b) and (c) respectively. It can be seen that the visual quality is improved remarkably with our modified QEM described in section 2.3.2. Table 1 gives the running times of the two methods. It shows that our method is faster than QEM.



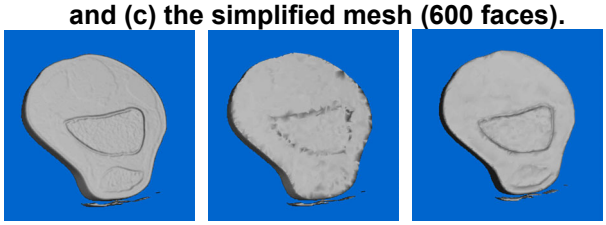
(a) (b) (c)

Figure 4. Crater model: (a) the original mesh (199,114 faces), (b) the simplified mesh (20,000 faces) and (c) the simplified mesh (3,000 faces).



(a) (b) (c)

Figure 5. Cow model: (a) the original mesh (5,804 faces), (b) the simplified mesh (1,000 faces) and (c) the simplified mesh (600 faces).



(a) (b) (c)

Figure 6. Knee model: (a) the original mesh (95,936 faces), (b) the simplified mesh (15,000 faces) using QEM, (c) the simplified mesh (15,000 faces) using our method.

4. Conclusion and future work

In this paper we presented an efficient simplification algorithm combining modified QEM with half-edge data structure. The experimental results show that our method is faster than QEM for general meshes and at the same time, it can produce better visual quality for a certain kind of 3D medical models whose accurate normal vectors are known. Moreover, since the simplified model is the subset of the original meshes, it can be easily used to the constructing and editing of multi-resolution models.

The half-edge is restricted to represent manifold surfaces, so the work on how to expand it to non-manifold surfaces must be done in the future. We can consider the following three methods to solve this problem. The first is converting the non-manifold surfaces to manifold surfaces before simplification. The second is allocating an additional memory to record the information of non-manifolds. The third is using the more complicated data structure such as winged-edge or radial-edge data structure to simplify meshes. Which to use depends on practical applications.

References

[1] Greg Turk. Re-tiling Polygonal Surfaces. In: Computer Graphics Proceedings, Annual Conference Series, SIGGRAPH, Chicago, Zllinois, 1992, 55-64

[2] W. Schroeder, J. Zarge, and W. Lorensen, Decimation of triangle meshes, Computer Graphics, 26(2): 65-70, July 1992

[3] Jarek Rossignac, Paul Borrel, Multi-resolution 3D approximations for rendering complex scenes, In B.Falcidieno and T.Kunii, editors, Modeling in Computer Graphics:Methods and Applications, pages 455-465, 1993

[4] Hugues Hoppe, Progressive Meshes, Proc. of SIGGRAPH'96, pp. 99-108

[5] Michael Garland, Paul S. Heckbert, Surface simplification using quadric error metric, Proc. of SIGGRAPH'97, pp. 209-216

[6] Kalvin A. D., Taylor R. H., Surperfaces: Polygonal Mesh Simplification with Bounded Error, IEEE Computer Graphics and Applications, 1996, May, 64-77

[7] Michael Lounsbery, Tony DeRose. Multiresolution Analysis for Surfaces of Arbitrary Topological Type [Report]. Washington: University of Washington, January 1994

Original Model (Faces)	Simplified Model (Faces)	QEM			Our Method		
		Init. (Sec)	Simp. (Sec)	Total(Sec)	Init. (Sec)	Simp. (Sec)	Total(Sec)
Crater (199,114)	3,000	11.837	21.992	33.829	5.257	5.998	11.255
Cow (5,804)	600	0.25	0.65	0.9	0.12	0.09	0.21
Knee (95,936)	15,000	3.675	8.863	12.538	2.003	2.083	4.086

Table 1. Running times for QEM and our method