Abstract

In order to preserve structural edges and shapes while simplifying the triangle mesh of man-made 3D models, we propose a new quadric error metric based on the distances between the new vertex position and the affected edges for edge collapse operations. With a 2D view sampling method, the contour and silhouette likelihood of the edges are computed, which act as the quadric’s weights during the mesh decimation. The comparisons and analysis among our methods with different weighting schemes will be presented in this report. We also compare our method with QSlim [Garland and Heckbert 1997] and structure-aware mesh decimation method [Salinas et al. 2015] in multiple cases. Output generated by our method is better than QSlim and comparable to structure-aware method in most of the cases.

1 Introduction

Man-made 3D objects usually have a coarse level structure that can be modeled with very few triangles which still conveys the perceptual information of the original model while eliminating all the details. This kind of abstraction is very useful for rendering faraway objects which is also the reason why extreme simplification of 3D models matters. In the extreme mesh simplification scenario, traditional error driven methods often fails to preserve the coarse level structure of the original model. Although some recent works tackle the problem from a geometric approximation point of view involving high level observations on the model’s structural information, they rarely focus on preserving structural edges which is important since many structural shapes are conveyed by them.

Focusing on preserving structural edges and shapes while simplifying the triangle meshes of man-made 3D shapes, we propose a new quadric error metric based on the distances between the new vertex position and neighboring edges of the collapsed edge. We design our new method for triangle meshes in the edge collapse framework because of their conciseness and high efficiency. Given importance information of all the edges on a triangle mesh, we can use them as a weight of our quadric to guide the collapsing so that the new vertex replacing the collapsed edge could be close to all the important edges thus preserving the shape. Therefore, the importance information of edges play a vital role in our method.

In 2D drawings, contours and silhouettes are both key lines to convey the shape of a character or an object. 3D models are indeed 2D while viewing by human only that they have multiple views. In this sense, we compute contour and silhouette likelihood for each edge as a kind of edge importance using a 2D view sampling method.

In this project, we compare our method with QSlim and structure-aware method in multiple cases where the meshes are manifold CAD models, non-manifold CAD models, and models obtained by 3D scanning or some image based reconstruction method in order to investigate the ability and feature of our method. We also tested our method with two different weighting schemes, both of which make sense. Among the comparisons of multiple methods, our method generates simplified models which are better than QSlim and comparable to structure-aware method in most of the cases.

The rest of this report is organized as follows. Chapter 2 gives a literature review on mesh simplification in two aspects: error driven mesh decimation methods (§ 2.1) and high level geometric approximation methods (§ 2.2). Chapter 3 provides the derivation of our vertex-edge quadric (§ 3.2) from point-line distance formula in 3D (§ 3.1). Chapter 4 talks about contour and silhouette likelihood and the method to compute them. Chapter 5 first discuss the weighting schemes (§ 5.1), then presents and analyzes results in different cases (§ 5.2) and reports the timing statistics (§ 5.3). Finally, Chapter 6 concludes our work and gives the future works.

2 Related Works

2.1 Mesh Decimation Methods

The most efficient and popular mesh simplification method over the past twenty years is no doubt mesh decimation which collapse
or remove edges, triangles, and vertices sequentially followed by a specific error metric measuring the similarity between the simplified model and the original model.

[Garland and Heckbert 1997] proposed a quadric error metric (QSM) for edge collapse mesh simplification. This quadric measures the sum of squared distances from the new position to the incident triangles of the vertex. The quadric of triangle \( t \) lying in plane \( Ax + By + Cz + D = 0 \) is defined as

\[
Q_t = \begin{bmatrix}
A^2 & AB & AC & AD \\
AB & B^2 & BC & BD \\
AC & BC & C^2 & CD \\
AD & BD & CD & D^2
\end{bmatrix}
\]  

(1)

Then for a position \( p \), the squared distance from \( p \) to \( t \) can be calculated as \( p^TQ_t p \).

For a vertex \( v \) with \( n_{it} \) incident triangles \( t_i, i = 1, 2, \ldots n_{it} \), its quadric will be

\[
Q_v = \sum_{i=1}^{n_{it}} (r_{t_i}Q_{t_i})
\]  

(2)

where \( r_{t_i} \) is the area of \( t_i \) serving as a weight.

To collapse edge \( e_i \) with endpoints \( v_1 \) and \( v_2 \), the best new vertex position is obtained by solving the optimization problem:

\[
\arg\min_p f(p), f(p) = p^T(Q_{v_1} + Q_{v_2})p
\]  

(3)

Here the minimum value of \( f(p) \) is the minimum error introduced by collapsing \( e_i \). This minimum error is used as a sort key to maintain all the edges in a min heap for decimation. After each edge collapse operation, the quadric of the new vertex is simply \( Q_{v_1} + Q_{v_2} \). Finally, the minimum error of all the affected edges (incident edges of \( v_1 \) and \( v_2 \)) are all recomputed to reorder the heap before conducting the next edge collapse operation.

In order to work adaptively on non-manifold meshes, [Garland and Heckbert 1997] defined virtual planes across the boundary edges and the plane is perpendicular to the only incident triangle of the boundary edges. The quadric of these virtual planes will be involved in the quadric of the boundary edges so that they could be protected as well. But this strategy doesn’t work well in extreme cases where there are a lot of independent mesh components, boundary edges, and even non-manifold edges.

[Lindstrom and Turk 1998] put forward a different error metric which selects the position of the new vertex so that the original volume of the model is maintained. They minimize the per-triangle change in volume of the tetrahedra swept out by those triangles that are moved. Other than only considering geometric similarities between the original model and a single simplified result, [Hoppe 1996] introduced progressive mesh (PM), which is an efficient, lossless, continuous-resolution representation addresses several practical problems in graphics: smooth geometry preserving of level-of-detail approximations, progressive transmission, mesh compression, and selective refinement.

There are also works [Cohen et al. 1998] [Sander et al. 2001] [Garland and Heckbert 1998] paying attention to the appearance of the model during mesh decimation. They applied mesh parameterization schemes to add material metrics or constraints into the process to ensure appearance similarity at the same time.

### 2.2 Geometric Approximation Methods

However, to deal with non-manifold meshes and extreme simplification, methods exploring high level description of geometries can achieve much better results than mesh decimation but with a relatively larger time cost.

[Cohen-Steiner et al. 2004] cast shape approximation as a variational geometric partitioning problem. Using the concept of geometric proxies, they drive the distortion error down through repeated clustering of faces into best-fitting regions without parameterization or local estimations of differential quantities. [Mehra et al. 2009] introduced a novel algorithm for abstracting 3D models using characteristic curves or contours as building blocks for the abstraction. Their method robustly handles models with poor connectivity, including the extreme cases of polygon soups. [Yumer and Kara 2012] presented a co-abstraction method that takes as input a collection of 3D objects, and produces a mutually consistent and individually identity-preserving abstraction of each object.

Combining local operations and high level geometry constraints, [Salinas et al. 2015] proposed a structure-aware mesh decimation method which performs edge collapse decimation design to approximate the local mesh geometry as well as the geometry and structure of planar proxies detected in a pre-processing analysis step. Such structure-preserving approach is well suited to planar abstraction, i.e. extreme decimation approximating well the planar parts while filtering out the others.

### 3 Vertex-Edge Quadric

#### 3.1 Point-Line Distance in 3D

Let a line in 3D be specified by two points \( x_1 \) and \( x_2 \) lying on it as illustrated in Fig.2, so the distance between the line and a point \( x_0 \) is given by the minimum of the distances between \( x_0 \) and a point on the line:

\[
d = \frac{|(x_2 - x_1) \times (x_1 - x_0)|}{|x_2 - x_1|}
\]  

(4)

Here, the numerator is simply twice the area of the triangle formed by points \( x_0 \), \( x_1 \), and \( x_2 \), and the denominator is the length of one of the bases of the triangle, which follows since, from the usual triangle area formula, \( \Delta = bd/2 \). [Weisstein 2016]

![Figure 2: Illustration of point-line distance in 3D, from Wolfram MathWorld.](image)

#### 3.2 Quadric Derivation

Now we derive our vertex-edge quadric based on Eq.4. Since the cross product between two vectors \( a \) and \( b \) can also be expressed as the product of a skew-symmetric matrix (constructed using \( a \)) and vector \( b \): [Liu and Trenkler 2008]

\[
a \times b = \begin{bmatrix}
0 & -a_z & a_y \\
a_z & 0 & -a_x \\
a_y & a_x & 0
\end{bmatrix} b
\]  

(5)
we can express the squared point-line distance $d^2$ with matrix multiplications:

$$d^2 = (x_1 - x_0)^T E (x_1 - x_0)$$  \hspace{1cm} (6)

where

$$E = \frac{1}{e^T e} \begin{bmatrix} 0 & -e_z & e_y \\ e_z & 0 & -e_x \\ -e_y & e_x & 0 \end{bmatrix}^T \begin{bmatrix} 0 & -e_z & e_y \\ e_z & 0 & -e_x \\ -e_y & e_x & 0 \end{bmatrix}$$

$$e = x_2 - x_1$$

To rewrite Eq.6 using homogeneous coordinates, we need to solve for $c$ and $h$ from

$$\begin{bmatrix} x_0^T \\ 1 \end{bmatrix} \begin{bmatrix} E & c \\ e^T & h \end{bmatrix} \begin{bmatrix} x_0 \\ 1 \end{bmatrix} = (x_1 - x_0)^T E (x_1 - x_0)$$  \hspace{1cm} (7)

This gives us

$$c = -Ex_1$$  \hspace{1cm} (8)

$$h = x_1^T Ex_1$$  \hspace{1cm} (9)

Then we can define the vertex-edge quadric for a vertex $v$ on a triangle mesh with neighboring vertices $v_i$, $i = 1, 2, \ldots$ and incident edges $e_i = v_i - v$ as

$$Q_v = \sum_{i} w_i Q_{e_i}$$  \hspace{1cm} (10)

where $w_i$ is the weight of $e_i$, and

$$Q_{e_i} = \begin{bmatrix} E_{e_i}^T & -E_{e_i} \\ -v^T E_{e_i} & v^T E_{e_i} \end{bmatrix}$$

Similar to the triangle area weight in QSlim, here $w_i$ could be set to $|e_i|$ or $|e_i|^2$ intuitively. Moreover, structural information of edges is also needed for computing $w_i$ because long edges are not necessarily important. Since our vertex-edge quadric directly measures distances to the edges, it can perform much better than QSlim as for protecting important edges.

### 4 Contour and Silhouette Likelihood

An edge of a triangle mesh in a certain view is a contour edge if it is visible and only one of its two incident triangles is visible. In our 2D view sampling program, we decide whether an edge $e_i$ is a contour edge by first examining whether it satisfies

$$(n_1 \cdot m)(n_2 \cdot m) \leq 0$$  \hspace{1cm} (11)

where $n_1$ and $n_2$ are the normals of the two incident triangles of $e_i$, and $m$ is a vector from camera to the midpoint of $e_i$. We collect edges satisfying Eq.11 and then do the visibility test within OpenGL pipeline.

Silhouette edges of a triangle mesh is just a subset of contour edges where they lie on the visual hull of the model in the 2D view. Given these descriptions, contour and silhouette edges could be easily rendered on the screen, so that we can just extract the IDs of those edges from the frame buffer.

We used an ID rendering and sampling method to extract the edges. The color of an edge is computed from its edge ID, $i$, which is unique:

$$R = (i \gg 16) \& 255$$  \hspace{1cm} (12)

$$G = (i \gg 8) \& 255$$  \hspace{1cm} (13)

$$B = i \& 255$$  \hspace{1cm} (14)

This will provide up to $2^{24} = 16,777,216$ different color and edge IDs, which is enough for almost all the models nowadays. Then by getting the color of each pixel in the frame buffer and computing the ID reversely:

$$i = (R \ll 16) + (G \ll 8) + B$$  \hspace{1cm} (15)

we can know exactly which edges are rendered in the frame buffer, thus deriving the set of contour or silhouette edges in the current view as depicted in Fig.3.

Figure 3: View sampling illustration of a chair model. Note that partly visible edges are also counted as visible.

To compute each edge’s contour and silhouette likelihood, we sample view points on a bounding sphere of the model and do the ID rendering and sampling for each view, counting the times of each edge being contour or silhouette. Then the counted times of all the edges are mapped to $[0, 1]$ linearly as the likelihood. Fig.4 shows a visualization of the contour and silhouette likelihood of two models.

Some model may contain edges with one or more than two incident triangles. For these boundary and non-manifold edges, we don’t count them as either contour or silhouette, but we will set their likelihood to 1 since they are apparently important structural edges. This strategy makes our statistics more adaptive to non-manifold triangle meshes.

### 5 Experiment and Results

#### 5.1 Weighting Schemes

Weighting our vertex-edge quadric by edge length, we get a QSlim-like quadric error metric. But it will not be an appropriate error metric for edge collapse operations since edge importance does not only depend on edge length. If that is the case, all the edges will have comparable importance, then our method will produce a seemingly optimized resulting vertex trying to get close to each edge but
actualy far away from all the edges, thus introducing big changes to all the edges (only the shape of near planar regions can be preserved as it’s similar to QSub in this case). On the contrary, if only several edges are with comparable weights and others are with negligible weights, our method will try to make the new vertex close to those important edges while probably far from those less important edges, producing results that preserves the important edges under the sacrifice of less important edges. In this case, both the local shape and the important edges are preserved because the important edges define the local shape.

Consequently, we compute our weight for each incident edge of a vertex using both the edge length or squared edge length and the biased contour likelihood \( P_c(i) = P_e(i) + b \):

\[
\begin{align*}
  w_i &= |e_i| P_e(i) + b \quad (16) \\
  w_i &= |e_i|^2 P_e(i) \quad (17)
\end{align*}
\]

Here \( b \) is a tiny value which prevents from multiplying by zero while not influencing the likelihood distribution. In this way, patches containing edges with larger likelihood and edge length will be considered more expensive to collapse, while patches with all zero-likelihood edges will be considered very cheap to collapse. The offset \( b \) ensures that the latter case patches still has non-zero quadric error metric to follow.

5.2 Results and Analysis

Fig.5 shows results of simplifying a train model with 57,244 faces originally to only 1,142 (2%) and 570 (1%) faces with QSub and our methods. As you can see, our methods preserve the shape of the big horizontal cylinder, the two side windows, the back of the tank, and the stripe of the wheels better than QSub in both of the two LoD’s. It also shows that using edge length or squared edge length to compute edge weight both make sense in our method. The difference is that weights computed using squared edge length have larger differences between short edges and long edges, so that short edges are more likely to be sacrificed (e.g., the wheels) and long edges (e.g., the big horizontal cylinder) are more likely to be preserved. However, our methods failed to preserve shapes of those concave regions, e.g., shape under the big horizontal cylinder. This is because our likelihood doesn’t count edges lying in concave regions. The reason is when those edges satisfy Eq.11, they must be invisible. This is reasonable since concave shapes are less important than convex shapes when viewing far away objects.

Now we will show that our method also works for triangle meshes of man-made 3D models containing lots of boundary or non-manifold edges.

Fig.1 shows the simplification of a tank model containing 60,000 faces originally with 492 non-manifold edges. The model is simplified to 2,000 faces using QSub, structure-aware method and our method with \( w_i = |e_i|^2 P_e(i) \). Our method preserves the shape of the tank’s top while the other two failed and they produced holes. Besides, our method also preserves the two bulges in the front and the thin cylinders on the top better.

Fig.6 shows the results of simplifying two models containing lots of boundary edges. As we can see, QSub failed to preserve the topology, shape, and the continuity of the model, while our method and structure-aware method succeeded. However, structure-aware method still gives discontinuous pillar feet while our method does not. But our method can not preserve the shape of the top due to the complicated surface details. These details will make the likelihood very dense on the top, thus affecting our vertex-edge quadric as has been described previously. But structure-aware method is really good at this case since it fits plane proxies to the model and then follows them as prior knowledge and constraints during decimation.

If a 3D model comes from 3D scanning or some image based reconstruction methods, it will have many noisy edges gathered together to convey the shape of a straight line. Then, similarly, our method might probably fail in this case. Fig.7 provides an example. In fact, this is also the reason why our method does not work for creature models or shapes composed of lots of curves.

5.3 Timing

All the simplifications were conducted on a laptop with Intel(R) Core(TM) i7-4710HQ CPU @ 2.50GHz and 64-bit Windows 10. QSub only need less than 1 second per example. Structure-aware method needs around 20 to 30 seconds per example. Our method needs around 20 seconds to compute the likelihood and around 2 seconds to do the edge collapse operations.

6 Conclusion and Future Works

We designed a vertex-edge quadric measuring the distances between the new vertex position and the important edges in edge collapse operations in order to preserve structural edges and shapes of the original model. This is achieved by assigning structural information, the contour likelihood, as weights to the neighboring edges of the collapsed edges. Aiming at man-made 3D shapes, our method performs better than QSub and structure-aware method in interactive time no matter whether the mesh is manifold or non-manifold. However, our method failed to preserve the shape and volume of concave regions since the contour likelihood of concave regions are always 0. Models with noisy edges representing a straight line will also make our method fail. What’s more, our method does not work for man-made shapes with lots of curves either. To sum up, our vertex-edge quadric is promising, and the weighting scheme and the feature of the method still needs to be explored.

As for future works, there are several directions worth investigating. We didn’t apply silhouette likelihood while computing the weights. This is worth trying since silhouette does not involve those detailed...
Figure 5: Simplification of a train model (only 13 non-manifold edges) using QSlim and our methods with $w_i = |e_i| P_{c}(i)$ and $w_i = |e_i|^2 P_{c}(i)$. Our methods outperform QSlim as for preserving the shape of the big horizontal cylinder, the two side windows, the back of the tank, and the stripe of the wheels in both of the two LoD’s. However, our methods failed to preserve shapes of concave regions, e.g., shape under the big horizontal cylinder.

geometry on the surfaces, which will probably produce better top shape for the Triumph Arch model and in cases where coarse level structural surfaces are decorated with lots of details.

Besides, since our method does not perform well in preserving shape and volume of concave regions, we might involve dihedral angle [Paillè et al. 2015] of edges while computing weights. This might be helpful because concave regions will get comparable importance to convex regions. Or we can blend our quadric with QSlim or [Lindstrom and Turk 1998]. But the balance of the quadrics is hard to keep, and the robustness of the combined error metric will also be a big issue.

Acknowledgements

Thanks I-Chao for the book Polygon Mesh Processing [Botsch et al. 2010]. Thanks Nicholas for the discussion. Thanks Alla for the supervision and support. This project is based on El Topo¹ [Brochu and Bridson 2009].

References


¹El Topo: a public domain C++ package for tracking dynamic surfaces represented as triangle meshes in 3D.


Figure 6: Simplification of a sofa model (400 boundary edges) and a Triumph Arch model (844 boundary edges) using QSLim, structure-aware method, and our method with $w_i = |e_i|^2 P_c(i)$. For the sofa model, QSLim failed to preserve the topology of the arms and the shape of the back, while our method and structure-aware method produced fine results only that the shape of the feet are affected. Our method also provide better shape of the back. For the Triumph Arch model, QSLim failed to preserve the continuity of the pillars and feet and structure-aware method only preserved the continuity of the pillars but still not the feet. Our method gives continuous shape both on the pillars and the feet. As for the top shape, structure-aware method performed well, while our method failed.

Figure 7: Simplification of a church model (322 boundary edges) using QSLim, structure-aware method, and our method with $w_i = |e_i|^2 P_c(i)$. Only structure-aware method succeeded in preserving the coarse level structure of the original mode. Our method performed equally bad to QSLim.


