User-Guided Simplification

Youngihn Kho* Michael Garland†

University of Illinois at Urbana–Champaign

Abstract

While many effective automatic surface simplification algorithms have been developed, they often produce poor approximations when a model is simplified to a very low level of detail. Furthermore, previous algorithms are not sensitive to semantic or high-level meanings of models. In this paper, we present a user-guided approach for mesh simplification that aims to overcome such limitations. Our proposed method allows users to selectively control the relative importance of different surface regions and preserve various features through the imposition of geometric constraints. Using our system, users can produce perceptually improved approximations with very little effort.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Surface and Object Representation;

Keywords: user-guided simplification, level of detail, quadric error metrics

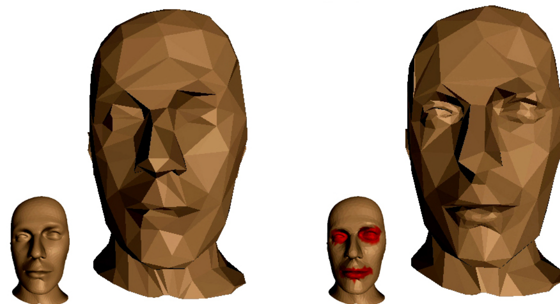

Figure 1: Simplified face models of 650 triangles by fully automatic QSlim (left) and by our system (right). Both the eyes and lips look significantly improved by user's guidance.

1 Introduction

Throughout the last decade, many automatic simplification algorithms have been developed to generate an approximation of fewer polygons from complex models. While they produce very plausible results in many cases, at very low levels of detail their approximations do not preserve the visual appearance of the original model very well. Furthermore, previous automatic algorithms ignore semantic or high-level meanings of models, since these are hard to measure by the simple geometric error metrics used by earlier methods. For example, features such as eyes in a face are semantically crucial but geometrically small (Figure 1).

Since perceptual importance is ultimately determined by a human observer, we argue that human guidance in the simplification process is a natural way to solve the problem mentioned above. Starting from this idea, we propose a new user-guided simplification system that allows a user to interactively control an automatic simplification method. Our system is built upon the *quadric based simplification algorithm* [Garland and Heckbert 1997], which is a very efficient algorithm for interactive systems. With our system, users can generate visually superior approximations with very little effort. Specifically, users can guide the computation of errors and vertex placement of edge contractions by directly interacting with the underlying algorithm.

We provide two main tools: adaptive simplification and geometric constraints. Using adaptive simplification, users can specify that some areas are to be simplified more than others. For example, a user can specify important areas such as the eyes of a face by directly painting over the surface (Figure 1). Geometric constraints preserve features by guiding the placement of vertices on the approximation. We suggest three types of geometric constraints: contour, plane, and point constraints. Users can apply appropriate constraints to accomplish various tasks. For instance, point constraints are used to preserve the positions of vertices. An example is shown in Figure 5 where a point constraint is introduced on each tooth.

2 Background

Among various simplification algorithms, we will survey the literature most relevant to our own work. Readers can refer to recent surveys [Garland 1999; Luebke et al. 2002] for more comprehensive reviews.

Many successful methods are based on *iterative edge contraction* [Hoppe 1996; Garland and Heckbert 1997; Lindstrom and Turk 1998]. These approaches iteratively collapse edges in increasing order of cost. One important feature of these methods is that they produce hierarchical structures called *vertex trees*. In the vertex tree, each vertex in an original model forms a leaf node, and each edge contraction makes a parent node of its two incident vertices. Once a vertex tree is created, we can extract various levels of detail by choosing appropriate *cuts*, by performing all the contractions below the cut. The most common application of this hierarchical structure is view-dependent level of detail control [Hoppe 1997; Xia and Varshney 1996; Luebke and Erikson 1997].

Almost all previous simplification methods concentrate on purely automatic processes. As far as we know, the only two previous semi-automatic simplification methods are Zeta [Cignoni et al. 1998] and Semisimp [Li and Watson 2001]. These two methods allow user interaction to produce improved approximations. Pojar

*e-mail: kho@uiuc.edu
†e-mail:garland@uiuc.edu

and Schmalstieg [2003] have also independently developed a user-controlled method based on weighted quadrics, adopting a similar approach to our own.

Zeta requires a pre-computed sequence of primitive simplifications as input. Users can selectively refine or simplify a model by locally changing error thresholds to extract different approximations that did not appear during the original simplification process. However, they are limited by the ordering constraint of the pre-computed sequence of primitive simplification operations. Semisimp takes a pre-computed sequence of edge contractions and allows users to extract new approximations such as in Zeta. This is essentially extraction of different cuts in the input vertex tree. However, unlike Zeta, Semisimp exploits the vertex tree structure to provide segmented simplification. With its hierarchy manipulation tool, users can partition an input model to match the semantics of the model, and simplify the model in a segmented fashion, in which contractions may not span partitions. By doing this, it sidesteps the ordering constraint of the pre-computed sequence of edge contractions.

In contrast, our approach starts with an input model, then creates a vertex tree by directly guiding the underlying algorithm itself, making it a superset of these previous approaches. Rather than allowing the user to directly manipulate the vertex positions of approximations, we provide geometry constraints to control the geometry of approximated models during the simplification process. We believe that these differences make our approach more general and flexible. Furthermore, since our system creates vertex trees, it can easily be coupled with the functionality provided by systems such as Semisimp and Zeta.

Some researchers have also considered error metrics based on models of human perception. Lindstrom and Turk [2000] suggested an image-driven simplification algorithm in which the error is measured in the image space. Recently Watson *et al.* [2001] analyzed the visual fidelity of various automatic error metrics in terms of human based experimental measures. Luebke and Hallen [2001] and Reddy [2001] suggested new approaches based on human vision models. Since their approaches focused on low-level human perception model, their application is somewhat limited: their perceptual model cannot recognize higher-level semantics. For example, low-level vision models cannot distinguish actual features from noisy fluctuation such as fur, which have similar spatial frequency. By allowing users to guide the simplification process, our approach can account for both low-level and high-level human perception.

2.1 Quadric Based Simplification

We selected the quadric based simplification method [Garland and Heckbert 1997] as a base algorithm for its time efficiency and relatively high quality of approximations.

Given a plane $\mathbf{n}\cdot\mathbf{v}+d=0$ with unit normal $\mathbf{n}$, and point $\mathbf{v}$, the *quadric* Q is defined

$$Q=(\mathbf{A},\mathbf{b},c)=(\mathbf{n}\mathbf{n}^{\mathrm{T}},d\mathbf{n},d^2) \tag{1}$$

Using this construction, we can calculate $Q(\mathbf{v})$, the squared distance of the point $\mathbf{v}$ to the plane, as follows:

$$Q(\mathbf{v})=\mathbf{v}^{\mathrm{T}}\mathbf{A}\mathbf{v}+2\mathbf{b}^{\mathrm{T}}\mathbf{v}+c \tag{2}$$

The error at a vertex is the sum of squared distances from a vertex $\mathbf{v}$ to a set of planes, and can be written as

$$\sum_i Q_i(\mathbf{v})=(\sum_i Q_i)(\mathbf{v}) \tag{3}$$

During initialization, each vertex is assigned a quadric which is the sum of the *fundamental quadrics*, formed by the unit normal and offset of its incident faces. After initialization, for each candidate edge contraction $(\mathbf{v}_i,\mathbf{v}_j)\rightarrow\bar{\mathbf{v}}$, the optimal position $\bar{\mathbf{v}}$ and the cost of this contraction are computed by

$$\bar{\mathbf{v}}=-(\mathbf{A}_i+\mathbf{A}_j)^{-1}(\mathbf{b}_i+\mathbf{b}_j) \tag{4}$$

$$Q(\bar{\mathbf{v}})=Q_i(\bar{\mathbf{v}})+Q_j(\bar{\mathbf{v}})=(Q_i+Q_j)(\bar{\mathbf{v}}) \tag{5}$$

Once this edge is contracted, the new vertex $\bar{\mathbf{v}}$ accumulates the planes associated with the edge end points by $Q_{\bar{v}}=Q_i+Q_j$.

3 Guiding Simplification

The quality of the approximations produced by edge contraction algorithms is fundamentally dependent on the order of contractions and the repositioning of vertices following contraction. The key to our approach is to guide simplification by manipulating the quadric associated with each vertex. By manipulating quadrics, we can implicitly re-order the edge contraction sequence and control the selection of optimal positions, producing vertex trees of varying structures. Specifically, we propose two fundamental mechanisms: adaptive weighting of quadrics and additional constraint quadrics. In this section, we assume that the weights and constraint quadrics are assigned at initialization. In Section 4.1, we will relax this restriction.

3.1 Adaptive Weighting of Quadrics

We can adaptively simplify a model by controlling the order of contractions selected by the underlying algorithm. Since edge contractions are selected in increasing order of cost, and since this cost is computed by $Q(\bar{\mathbf{v}})$, we can manipulate their order by adaptively *weighting* quadrics, multiplying the quadric of each vertex by some scalar factor. Conceptually, we assign a weight for each input vertex $\mathbf{v}_i$, that is $Q_i\leftarrow w_iQ_i$, at the initialization step.

During simplification, the areas with heavy weights have higher contraction costs. Therefore, the edge contractions in those areas are delayed and the regions maintain higher levels of detail. As an example, users can apply heavy weights around eyes and lips in a face model, which is indicated in red paint (Figure 1). The eyes and lips show a higher level of detail than before.

Weighted quadrics also change the optimal position. In weighted quadrics, the optimal position is chosen to minimize the weighted sum of squared distances: it is biased towards planes with heavier weights. This is a desirable property, since heavily weighted vertices, which the user has designated as important features, will be more likely to preserve their positions.

It is left to the user to select appropriate weights. However, we use $w\log|V|$ for the actual weight, which scales the user-given weight w by the log of the number of vertices $|V|$ in the original model. In order to get similar results of a feature from two different scales of models, we need a roughly similar ratio of triangles in proportion to the size of the models (or the vertex tree). Since errors are accumulated exponentially, the $\log|V|$ term is used to compensate for the different scales.

3.2 Constraint Quadrics

Applying heavy weights is very useful for preserving features, but it is not always sufficiently effective, since heavily weighted regions preserve features by leaving more polygons in those areas. However, we can preserve features without using too many polygons in many cases. To do this, we extend the boundary constraints of the quadric simplification algorithm.

In the original quadric simplification algorithm, each vertex is assigned a *geometry quadric* derived from the geometry of its incident faces. Optionally, it can add boundary constraints to preserve the boundary of models. In fact, we can create virtual quadrics from arbitrary planes to impose geometric constraints by simply adding

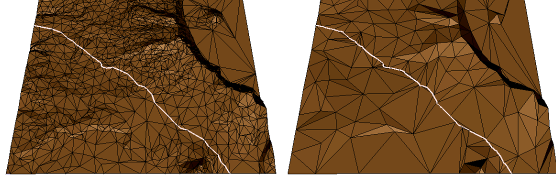

Figure 2: (Left) Contour constraints are applied along a feature indicated in a highlighted path. (Right) Contour constraints preserve the feature in the approximation.

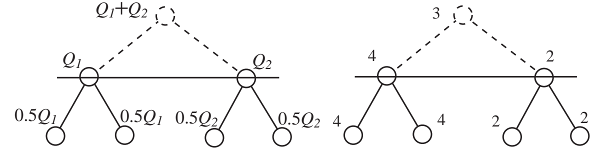

Figure 3: Illustration of (left) constraint quadric and (right) representative weight propagation. User input is given along the current cut. (Note that the dashed part does not exist at the cut.)

them to the appropriate vertices. We call those additional quadrics *constraint quadrics*.

Simple addition of the constraint quadric of a virtual plane will generally (1) increase the contraction cost, and (2) bias the optimal position towards the plane. In order to avoid unnecessary increase of cost, we separate constraint quadrics from the geometric quadrics and add them only when we compute the optimal positions. Note that the change of optimal positions still increases contraction costs (Equation 5).

We propose three types of constraint: contour, plane, and point constraints. These constraints are scaled both by the user-given importance of the regions and by the area of faces surrounding the regions where constraint quadrics are applied.

Contour Constraint A contour constraint can be applied to feature edges or color or texture discontinuity edges that the user wishes to preserve. The contour constraint is very similar to the boundary or discontinuity constraints suggested by the authors of the original QSlim method [Garland and Heckbert 1997].

For each edge in the selected contour, we generate two planes running through the edge and perpendicular to each other, then add the quadrics of these planes to the endpoints of the edge. Any pair of perpendicular planes running through the edge produce the same result. In our implementation, we simply pick a face incident to the feature edge as a first plane and its perpendicular plane as the other plane. Since the contour constraints measure the squared distance from the edge, those constraint quadrics will pull the optimal position onto the contour edges. Consequently, it preserves the shape of contours. An example of a contour constraint is shown in Figure 2.

Plane Constraint A plane constraint is designed to be applied to areas that the user wants to preserve as flat regions. For example, suppose we have a model of a cow on the ground. We can apply plane constraints on the feet of the cow so that the feet will remain in contact with the floor.

When the user selects a set of vertices, our system computes a least squares best fit plane to this set, then adds the quadric of the plane to the selected vertices. Note that usually the planes of plane constraints are more or less parallel to the surface, while those of contour constraints are perpendicular to the surface. The effect of this quadric is to pull the vertices' positions onto the plane. This helps the user keep the selected area flat, or can be used to remove noise in flat regions.

Point Constraint The quadric based simplification algorithm is based on the squared distance from *planes*. But quadrics can be directly used to measure the squared distance to *points* as well. For points $\mathbf{p}$ and $\mathbf{v}$, the squared distance between them is

$$\|\mathbf{v}-\mathbf{p}\|^2 = \mathbf{v}^{\mathrm{T}}\mathbf{v} + 2(-\mathbf{p}^{\mathrm{T}}\mathbf{v}) + \mathbf{p}^{\mathrm{T}}\mathbf{p} \tag{6}$$

Now we define a quadric for a point $\mathbf{p}$

$$Q = (\mathbf{A}, \mathbf{b}, c) = (\mathbf{I}, -\mathbf{p}, \mathbf{p}^{\mathrm{T}}\mathbf{p}) \tag{7}$$

where $\mathbf{I}$ is the 3x3 identity matrix. Together with Equations (2) and (6), the distance from a point $\mathbf{v}$ to $\mathbf{p}$ is simply $Q(\mathbf{v})$, which can be treated in the same way as the ordinary plane quadrics. In fact, point constraints are equivalent to placing three plane constraints through the point, where the planes are parallel to the coordinate planes. We can apply point constraints on vertices that are meant to maintain their positions (Figure 5).

4 Interactive Control

Our goal in interface design is to provide user friendly input methods and intuitive ways to get feedback. In our system, users can guide the simplification process at any time. This user guidance affects the structure of vertex trees of which approximations might be better. If a user refines an approximation, gives some guidance, and re-simplifies it again, then the new approximation will usually have a different vertex tree structure.

Figure 4 shows an illustration of a user interaction. A user simplifies and refines a model back and forth as in *(a)* and *(b)*. At some point, he re-weights parts of the current approximation *(c)*, then backtracks and re-simplifies for improved eyes *(d)*. To improve the teeth, the user refines the dragon and imposes point constraints on the teeth *(e)*. Finally, we get a much improved approximation *(f)*. Users directly paint over the surface with varying parameters to adaptively weight quadrics of each area. Constraints are specified by direct selection of vertices and edges.

4.1 Propagation of Weights and Constraints

In the previous section, we assumed that the weights and constraint quadrics are applied during the initialization phase, that is, at the original vertices. However, we want to let users freely interact at any level (cut) during the simplification or refinement process, and the user's guidance affect both the direction of simplification and refinement. This is important since users may need to guide the process several times depending on the feedback of their inputs.

The user's interaction with an input model only changes the quadrics of those vertices which exist at that time (or equivalently lie on the current cut in the vertex tree). If we directly multiply geometric quadrics by weights and add constraint quadrics upon each user input, then we would not be able to separate quadrics correctly into two children when we split a node. Our key approach to this problem is to maintain weighted (geometric) quadrics separated from constraint quadrics in each node of the vertex hierarchy tree. In fact, we have already seen a reason to separate constraint quadrics in each node: we can compute costs disregarding constraint quadrics. Constraint quadrics are added only when computing the optimal positions of contractions. However, we need new rules for propagation of quadrics.

The first rule is for the constraint quadrics. We only propagate half of each parent's constraint quadric to its children when we refine a node. On the other hand, we just add up children's constraint quadrics for their parent's constraint quadric when we simplify a node. This ensures that the quadric of each node is the sum of quadrics of leaf nodes in its subtree. This also guarantees the same result after we repeatedly simplify and refine without interaction during a series of simplification and refinement steps(Figure 3 left).

Propagation of weighted quadrics is somewhat tricky. If a user weights an intermediate node other than leaf nodes, then the quadric of a parent node might not be the sum of the quadrics of its two children. Although we can avoid this by multiplying the quadrics of all its descendent by the given weight everytime, we defer this until edge splits occur, since it is expensive. However, there is a scalar factor α such that $\overline{Q} = \alpha(Q_1 + Q_2)$, where $\overline{Q}, Q_1, Q_2$ are the quadrics of a parent and its two children. Therefore, when we split the node $\overline{Q}$, we first compute α by comparing the quadrics, then set $Q_1 \leftarrow \alpha Q_1$, $Q_2 \leftarrow \alpha Q_2$ for the quadrics of its children.

We still need a mechanism to visualize the weights for current vertices. This visualization is important since the user should know how a model is weighted from area to area. For this, we introduce a *representative weight*, a scalar value for each node to visualize an approximate weight for the node. In fact, exact representative weight in a node is generally undefined unless the node is directly weighted by a user. So we define a reasonable propagation rule as follows (Figure 3 right): (1) for an edge contraction, the weight of a parent node is the average of its children's, (2) for an edge split, the weights of two children are the same as the parent's. This rule is reasonable, since a parent node should represent both its two children, and a child inherits its parent's weight.

5 Results

Our implementation is built on the publicly available QSlim software [Garland and Heckbert 1997]. QSlim can optionally build the vertex tree during simplification to keep track of the quadric of each vertex, though it is not a required part of the core algorithm. For each edge contraction, QSlim adds a node in the vertex tree, for which the quadric is the sum of the quadrics of its two children. For vertex splits, the process is the reverse of edge contractions.

In order to add our new features, each node in the vertex tree should have two more components: a representative weight and a constraint quadric. For each edge contraction, geometric quadrics are treated the same way except they are weighted. The representative weights and the constraint quadrics follow the propagation rules outlined in the previous section. Constraint quadrics should be added when optimal positions are computed.

Our experiements do not show any noticeable difference in the simplification processing time between the original and our user-guided approach. Furthermore, the user interaction time of our system is also very short. It took only 2–3 minutes of user-guidance time to produce the approximations presented in this work.

In Figure 1, we simplified a face model with both fully automatic QSlim (left) and our program (right) to 650 triangles. We applied heavy weights around eyes and lips, shown as red paint in the small figure of the right side. The fully automatic method removes details of the eyes and lips while our system preserves much of them. The higher definition of the eyes and lips delivers a much clearer impression of the face.

The left picture in Figure 5 is the original dragon model with 50,000 faces. The middle approximation is generated without user guidance, while we applied heavy weights around the eyes, and added point constraints on the peak of the tooth in the right approximation. The right model preserves the scary teeth and eyes, while the left model loses most of these features. Furthermore, as shown in the thumbnails, our system does not degrade the overall shape.

Figure 6 visualizes error measured by the closest Euclidean distances from the original to the approximations. While the average errors are roughly the same, the user-guided approximations shows less error around perceptually important regions such as eyes with sacrifice of other parts like chin.

6 Conclusion and Future Work

We have proposed an interactive simplification system with user guidance. In our system, users can interact directly with the underlying quadric-based simplification algorithm to create an improved version of approximations with very little effort. Using our system, users can preserve feature areas or semantic meanings of input models and can produce significantly better results than the fully automatic method at low levels of detail. Since our new features can be put into the original algorithm without significant changes or overhead, our system still provides the advantages of the underlying algorithm, such as time efficiency.

While our experimental results show that our system works well, there are a number of areas to be improved further. One important weakness of our system is that the user may need to repeat the simplification and refinement process changing weights several times before he is satisfied with the approximation. Our current system requires users to directly specify weights. We suggest two future directions to improve this weakness. The first one is to consider a sophisticated error metric that can quantify perceptual importance. The second direction is to incorporate simplification with learning algorithms: a small set of user guidance trains the algorithm, then the algorithm automatically guides the rest of the process for similar kinds of models which are not from the example set.

References

CIGNONI, P., MONTANI, C., ROCCHINI, C., AND R.SCOPIGNO. 1998. Zeta: A resolution modeling system. *GMIP: Graphical Models and Image Processing 60*, 5, 305–329.

GARLAND, M., AND HECKBERT, P. S. 1997. Surface simplification using quadric error metrics. In *Proceedings of ACM SIGGRAPH 97*, 209–216.

GARLAND, M. 1999. Multiresolution modeling: Survey & future opportunities. In *State of the Art Report*, Eurographics, 111–131.

HOPPE, H. 1996. Progressive meshes. In *Proceedings of ACM SIGGRAPH 96*, 99–108.

HOPPE, H. 1997. View-dependent refinement of progressive meshes. In *Proceedings of ACM SIGGRAPH 97*, 189–198.

LI, G., AND WATSON, B. 2001. Semiautomatic simplification. In *ACM Symposium on Interactive 3D Graphics 2001*, ACM, 43–48.

LINDSTROM, P., AND TURK, G. 1998. Fast and memory efficient polygonal simplification. In *IEEE Visualization 98*, 279–286,544.

LINDSTROM, P., AND TURK, G. 2000. Image-driven simplification. *ACM Transactions on Graphics 19*, 3 (July), 204–241.

LUEBKE, D., AND ERIKSON, C. 1997. View-dependent simplification of arbitrary polygonal environments. In *Proceedings of ACM SIGGRAPH 97*, 199–208.

LUEBKE, D., AND HALLEN, B. 2001. Perceptually-driven simplification for interactive rendering. In *Rendering Techniques 2001: 12th Eurographics Workshop on Rendering*, Eurographics, 223–234. ISBN 3-211-83709-4.

LUEBKE, D., REDDY, M., COHEN, J., VARSHNEY, A., WATSON, B., AND HUEBNER, R. 2002. *Level of Detail for 3D Graphics*. Morgan Kaufmann.

POJAR, E., AND SCHMALSTIEG, D. 2003. User-controlled creation of multiresolution meshes. In *ACM Symposium on Interactive 3D Graphics 2003*.

REDDY, M. 2001. Perceptually optimized 3D graphics. *IEEE computer Graphics and Applications 21*, 5, 68–75.

WATSON, B., FRIEDMAN, A., AND MCGAFFEY, A. 2001. Measuring and predicting visual fidelity. In *Proceedings of ACM SIGGRAPH 2001*, 213–220.

XIA, J. C., AND VARSHNEY, A. 1996. Dynamic view-dependent simplification for polygonal models. In *IEEE Visualization 96*, 327–334.

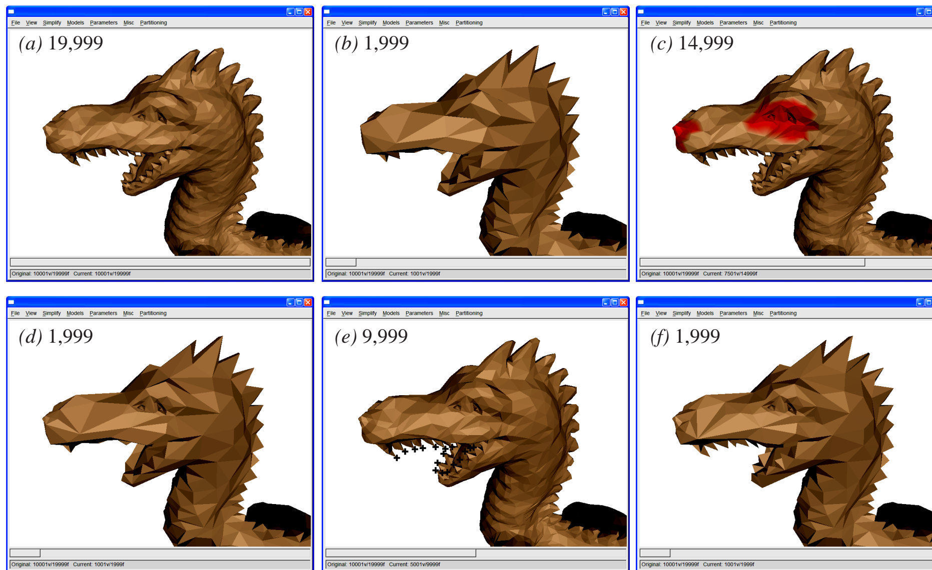

Figure 4: An illustration of user's interaction. The number of triangles is indicated in each figure.

Figure 5: (Left) A dragon model of 50,000 faces. (Middle) 1,500 faces fully automatically simplified by QSlim, and (Right) simplified by our system. Note that our dragon still has scary teeth and threatening eyes.

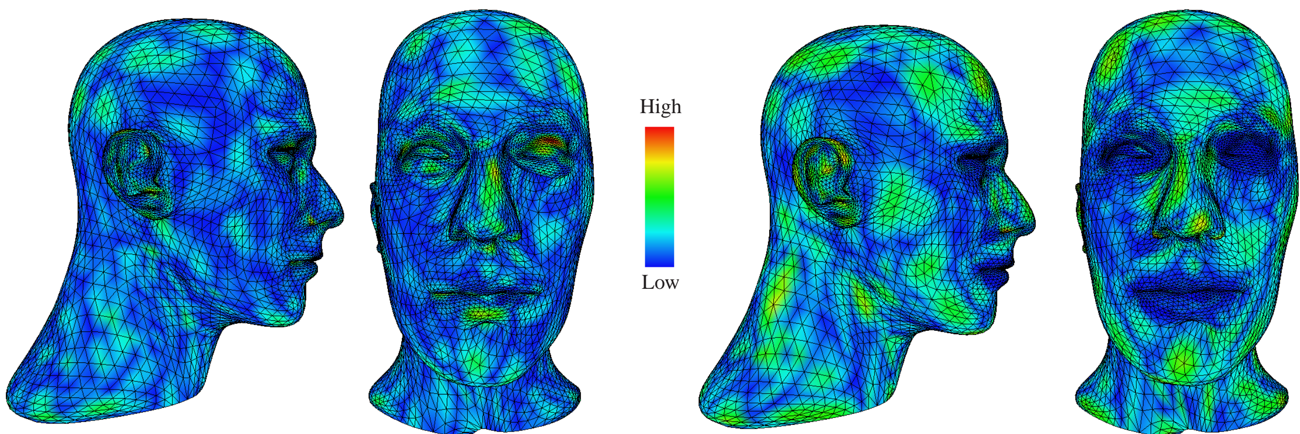

Figure 6: Visualization of relative error distribution of approximations from QSlim (left) and our system (right). While the left figure shows roughly uniform error distribution over the surface, our result indicates less error for perceptually important areas.