User-Controlled Creation of Multiresolution Meshes

Erik Pojar
neo, Vienna, Austria
erik.pojar@neo.at

Dieter Schmalstieg
Vienna University of Technology, Austria
dieter@cg.tuwien.ac.at

Abstract

We present a tool for the user-controlled creation of multiresolution meshes. Several automatic mesh reduction methods of high quality have been presented in the past, but most of these methods are not able to identify mesh regions of high semantic or functional importance, for example the face of a character model or areas deformed by animation. To address this problem, we present a method allowing a user to provide importance weights for mesh regions to control the automatic simplification process. To demonstrate the usefulness of this approach in a real world setting, a Maya plug-in is presented that lets the user create multi-resolution meshes with importance weighting interactively and intuitively. The user simply paints the importance of regions directly onto the mesh. All user input like weighting, resolution change, etc. are applied in real-time to give instant feedback during the modeling process. The plug-in can handle arbitrary meshes with attributes (vertex colors, textures, normals) and attribute discontinuities. This work aims to show that an integrated editing approach with full support for mesh attributes, which lets the user exercise selective control over the simplification rather than operating fully automatic, can bring multiresolution meshes out of academic environments into widespread use in the digital content creation industry.

ACM Category and Subject Descriptor: I.3.5 [Computer Graphics] Computational Geometry and Object Modeling - *hierarchy and geometric transformations*
Additional Keywords: level of detail, model simplification, multiresolution modeling

1 Introduction

Complex polygonal meshes are ubiquitous in computer graphics. However, real-time applications such as video games executing on constrained hardware such as game consoles, or applications that stream geometric models from CDROM [19] or over the network, require control over the size and complexity of polygonal geometry.

One way to get a low polygon approximation of a large model is to create it manually [15]. An artist using a modeling package can either manually reduce the original high polygon model, or create a representation with fewer polygons from scratch. However, with growing complexity of the original model, reduction by hand becomes an unfeasible approach. To simplify complex models quickly, automatic mesh reduction tools are needed. This need was recognized long ago, and this field has received a lot of research attention in the past [2][12].

Most reduction algorithms share the common goal of completely automatic reduction of the input meshes. Automatic reduction is doubtlessly necessary and appropriate for large models from real-world sources, such as laser ranging, surveying, medical scans, FEM etc. These data sets are typically extremely large, and must be reduced to moderate size with fully automated methods and guaranteed error bounds.

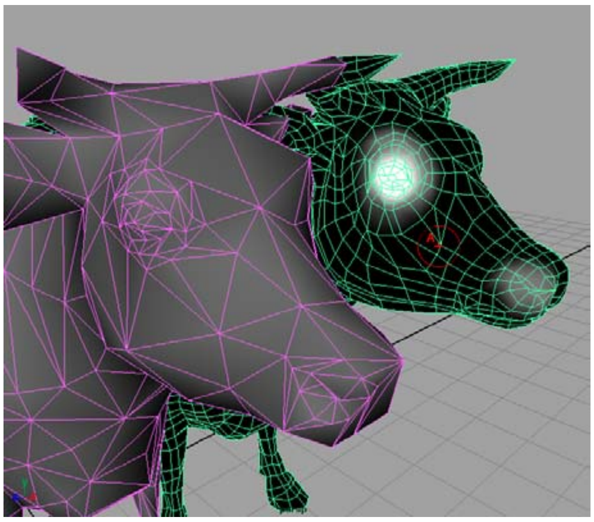


Figure 1: The „cow“ model reduced to 20% - eyes and nose marked as important

In contrast, the digital entertainment industry uses interactive modeling tools to create data sets of moderate complexity, which must then be reduced to really small sizes (a few thousand triangles). In this domain, fine-grained artistic control is required.

This is especially true if parts of the mesh are of high semantic or functional importance. For example, consider the potential semantic and functional meaning for a model of a human. The face may be of higher visual importance than the body; we would like to spend more polygons for the face than for the rest of the geometry. Depending on the application, other regions of the mesh may have similar high semantic importance to the user. If the human model is animated through e.g., skeletal animation, functional importance arises. We would like to keep more polygons for the deformable regions around the joints than for the rigid mesh areas.

To some extent, visual importance can be deduced from the geometry. Several simplification algorithms take mesh borders and (attribute) discontinuities into account because these features are of high perceptual importance. But ultimately, only the creator knows the intended use of a model, therefore completely automatic reduction can never infer all the constraints of a model.

However, we believe that it is possible to get the best of both worlds: To enable user-controlled creation of multiresolution meshes, we introduce a weighted error metric that lets the user specify the importance of mesh regions. The reduction of the mesh still happens automatically, but the user input is taken into account: Regions of high importance are reduced less, while regions of low importance are reduced more aggressively. By embedding importance weights into the simplification metric, the user gains a new level of control over the model reduction process, enabling users to create simplified models of higher functional, semantic and visual quality.

Another goal of our work was to create a real-world tool that is easy to use. Our solution was therefore implemented as a

Maya plug-in in order to offer an integrated workflow to the user. Using Maya's rich Artisan interface, a user can specify the importance of a mesh region, by directly painting weights onto the mesh. While editing, all user input is always applied in real-time to give instant feedback during the modeling process. The plug-in handles arbitrary meshes with arbitrary mesh attributes (vertex colors, textures, normals) and attribute discontinuities.

In the remainder of this paper, we review related work (section 2), discuss the quadric error metric (section 3), followed by the user interface (section 4), and show some results (section 5).

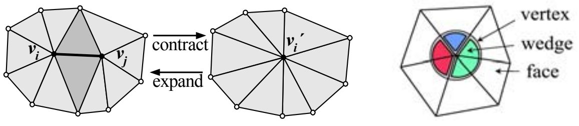


Figure 2: (left) Contraction of the edge ($\mathbf{v}_i$, $\mathbf{v}_j$) into a single vertex $\mathbf{v}_i'$, (right) Wedge-based representation - each wedge is associated with a separate quadric

2 Related Work

A large number of methods for automatic surface simplification have been presented in the past. For a recent review of different simplification schemes, see [2][12]. The approach presented here falls into the category of iterative decimation through edge contraction (Figure 2, left). A key problem in selecting an edge or vertex pair contraction is a suitable error metric, e.g., [6][16][9].

Garland's and Heckbert's quadric error metric [11] is of particular relevance for us because we based our implementation on the QSlim software package [13]. Later, in [10] Garland and Heckbert extended their quadric metric to handle surface attributes (texture coordinates, normals, etc.) when reducing the geometry. For a mesh with m scalar attributes, they generalized their distance-to-plane metric in $\mathbf{R}^3$ to a distance-to-hyperplane metric in $\mathbf{R}^{3+m}$. Hoppe presented an improved attribute-preserving quadric metric in [5]. He maintains the attribute-to-plane correspondence in $\mathbf{R}^3$, resulting in lower storage needs, better results and higher performance. He also demonstrated that *attribute wedges* are an effective way to handle attribute discontinuities. In [4] he showed how to further speed up the evaluation of the metric. Again, Hoppe's work [5] is of particular relevance for us, because we combined his improved attribute handling metric and the concept of attribute wedges with the original QSlim source code for our implementation.

There has not been as much research in the field of user-controlled mesh reduction as on automatic mesh reduction. SemiSimp [3] by Li and Watson aims at improving the quality of aggressively simplified models. They mainly focus on changing the order of the simplification hierarchy to change the detail distribution over the models surface. While doing so, the partial ordering of the simplification hierarchy has to be carefully maintained, which restricts the editing operations. SemiSimp also allows simple geometric manipulations like re-positioning of vertices and segmented simplification to further improve the final results.

A similar tool, Zeta [14], from Cignoni, Montani, Rocchini and Scopigno also allows the user to manipulate the initial order of the simplification steps. Their approach does not rely on a simplification hierarchy but on what they call a *hyper-triangulation model*. In effect, they can reorder the simplification operations more freely. However, the hyper-triangulation model only supports 2-manifold meshes. Zeta does not allow propagated geometry manipulation or segmented simplification.

SemiSimp and Zeta use local manual editing operations for mesh editing. Especially with SemiSimp the user has to work on the extremely fine-grained level of manipulating single vertices and edges. We chose a different approach: By embedding additional information into the model, our simplification still executes automatically, while taking semantic and functional importance into account. With respect to exercising control over the mesh on a higher level than simple vertices, our approach has some aspects in common with recent work on remeshing [17].

Also, both SemiSimp and Zeta are demonstrated as stand-alone tools. In contrast, our approach for user-controlled creation of multiresolution meshes has been implemented as a Maya plug-in, offering improved workflow integration.

3 Weighted Error Metric

3.1 Basic Quadric Metric

Our approach is based on iterative edge contractions using Garland's and Heckbert's *quadric error metric* [11], including Hoppe's enhancements for appearance attributes to the quadric metric [5]. As a foundation for the weighted error metric, we will briefly discuss the basics of quadric-based mesh simplification. The original model gets reduced by repeatedly applying edge contractions. Here is an outline of the basic algorithm:

```
1. Extract all edges from the source model
2. Assign a cost of contraction to each edge
3. Put the edges pairs in a priority queue, keyed on cost of contraction
4. Repeat until the desired approximation is reached:
    a. Remove the edge (i, j) with the least cost from the queue
    b. Contract this edge into the single vertex $\mathbf{v}_i'$, update the mesh neighborhood
    c. Update costs for all edges connected to $\mathbf{v}_i$ or $\mathbf{v}_j$
```

The quadric error metric is used to calculate both the cost of a contraction and the target position of the unified vertex $\mathbf{v}_i'$. It is a compact representation of the planes associated with a vertex through a 4x4 matrix. Because the matrix is symmetric, 10 floating-point values are sufficient to store a quadric.

A quadric $\mathbf{Q}$ is constructed from a plane. $\mathbf{Q}$ then represents that plane and $\mathbf{Q}(\mathbf{v})$ computes the squared distance of a vertex $\mathbf{v}$ to the plane represented by $\mathbf{Q}$. Quadrics define addition in a natural way: $\mathbf{Q}_1(\mathbf{v})+\mathbf{Q}_2(\mathbf{v}) = (\mathbf{Q}_1+\mathbf{Q}_2)(\mathbf{v})$, where $(\mathbf{Q}_1+\mathbf{Q}_2)$ is the component wise sum of the two quadrics.

The quadric error metric was enhanced by Garland and Heckbert [10] and Hoppe [5] to handle appearance attributes. Because Hoppe's method needs less storage space and produces results of higher quality, we chose to use his approach. He also handles attribute discontinuities like creases or discontinuous texture coordinates with *attribute wedges* (Figure 2, right) to represent the different attributes associated with a single vertex. For us, the utilization of attribute wedges was absolutely necessary, because production tools like Maya impose no restriction on the usage of attributes in a mesh. When using attribute wedges, instead of associating a quadric with each vertex, a separate quadric for each attribute wedges of a vertex is kept. A vertex is partitioned into $k \geq 1$ wedges, each wedge having its own attribute vector and its own quadric representing the faces together with the attribute vector of that wedge. Using wedge-based quadrics, the cost for contracting an edge (i, j), connecting the vertices $\mathbf{v}_i$ and $\mathbf{v}_j$ into a single vertex $\mathbf{v}_i'$ is calculated as:

- Compute the unified quadric $\mathbf{Q}'$ representing all wedge quadrics associated with $\mathbf{v}_i$ and $\mathbf{v}_j$
- Compute the vertex position $\mathbf{v}_i'$ that minimizes $\mathbf{Q}'(\mathbf{v}_i')$. This also computes the new attributes needed for all wedges.

The value of $\mathbf{Q}'(\mathbf{v}_i')$ is the cost of the contraction. For edge contractions involving attribute discontinuities the computation of the unified quadric $\mathbf{Q}'$ is not trivial. One needs to choose which wedge/attribute pairs get unified to a single wedge during contraction and which wedges get removed from the target vertex [5].

3.2 Weights

It is obvious that the simplification process is controlled by the cost of the contractions (see section 3.1). Mesh areas with edges of low contraction cost will get reduced earlier and more heavily than areas with higher contraction cost, regardless of the semantic or functional importance of these areas.

To give the user more control over the mesh simplification, we let the user change the computed cost of the edge contractions. To achieve this, the cost of the contraction is weighted by a user-controlled value.

For our discussion of the weighted error metric it is sufficient to know that the cost of contracting an edge (i, j) is computed through a quadric $\mathbf{Q}'$. $\mathbf{Q}'$ is the sum of all wedge quadrics associated with the vertices $\mathbf{v}_i$ and $\mathbf{v}_j$. This combined quadric $\mathbf{Q}'$ is then used to choose the target vertex position $\mathbf{v}_i'$ of the edge contraction. The target vertex $\mathbf{v}_i'$ is assigned the position that minimizes $\mathbf{Q}'(\mathbf{v}_i')$. The cost of the contraction, which we will call the geometric cost, cost_g, is the value of the unified quadric $\mathbf{Q}'$ at the position $\mathbf{v}_i'$:

$$\text{cost}_g = \mathbf{Q}_i'(\mathbf{v}_i')$$

The cost of a contraction is always positive (≥ 0). Assuming that there is a weight function $\omega(i, j)$ that defines a weight for an edge (i, j), the weighted cost, cost_w of a contraction is computed as:

$$\text{cost}_w = \omega(i, j) \cdot \text{cost}_g = \omega(i, j) \cdot \mathbf{Q}'(\mathbf{v}_i')$$

Recall that the mesh simplification is driven by the cost of the edge contractions. By using cost_w as our cost function, we can influence the order of the simplification operations through the weight function $\omega(i, j)$.

An important property of the weighting scheme is that it does not change the geometric properties of a pair contraction. The target position of the unified vertex is left unchanged and still is only determined by the squared distance to the set of planes represented by the quadric. The weight only changes the order in which the contractions are applied, not the result of the contractions.

To define the weight function $\omega(i, j)$, we have associated each vertex $\mathbf{v}_i$ with a scalar weight value $w_i \geq 0$. By default all vertex weights are assigned the value 1. The two vertex weights w_i, w_j of a vertex pair are used to compute the pair's weight based on a function of the user's choice:

$$\omega(i, j) = \text{average}(w_i, w_j) \;\; \textit{OR} \;\; \min(w_i, w_j) \;\; \textit{OR} \;\; \max(w_i, w_j)$$

The weight of a vertex is directly connected to the semantic or functional importance of the vertex to the user. To keep the geometric detail of important regions, the user can assign large weights ($w_i \geq 1$). This will increase the cost of contraction and thus cause the algorithm to first reduce other regions of the mesh. Regions of low interest can be assigned small weights in the range [0…1], which will make the contractions appear as low cost to the algorithm. A weight of 1 does not change the cost of the contraction at all.
If all weights w_i are left at their default value of 1, i.e.,

$$\text{cost}_w = 1 \cdot \text{cost}_g = \text{cost}_g,$$

then geometry is reduced as if there was no weighting at all.

3.3 Combination of weights

After the weight for a vertex pair has been defined, we also need to define the weight w_i' for the unified vertex $\mathbf{v}_i'$. It is generally not sufficient to use the same value as $\omega(i, j)$ for the weight w_i' of the unified vertex. Through an edge contraction, the two original vertices of the edge get contracted to a single vertex. Repeated pair contraction operations produce a vertex hierarchy or vertex tree (cf. [7], [1], [8]). and often are used for selective refinement of meshes. In a vertex hierarchy, each vertex represents all the vertices of its sub tree. The weight of a vertex in the hierarchy should thus represent the overall user importance of the combined mesh region represented through the sub-tree of the vertex.

To compute the combined weight of a vertex $\mathbf{v}_i'$ in a vertex hierarchy, we need to know the weights of the leaf vertices in the sub-tree of $\mathbf{v}_i'$. Let W be the set of these weights. Now we can compute the combined weight w_i' by selecting a function from:

$$w_i' = \text{average}(w \in W) \;\; \textit{OR} \;\; \min(w \in W) \;\; \textit{OR} \;\; \max(w \in W)$$

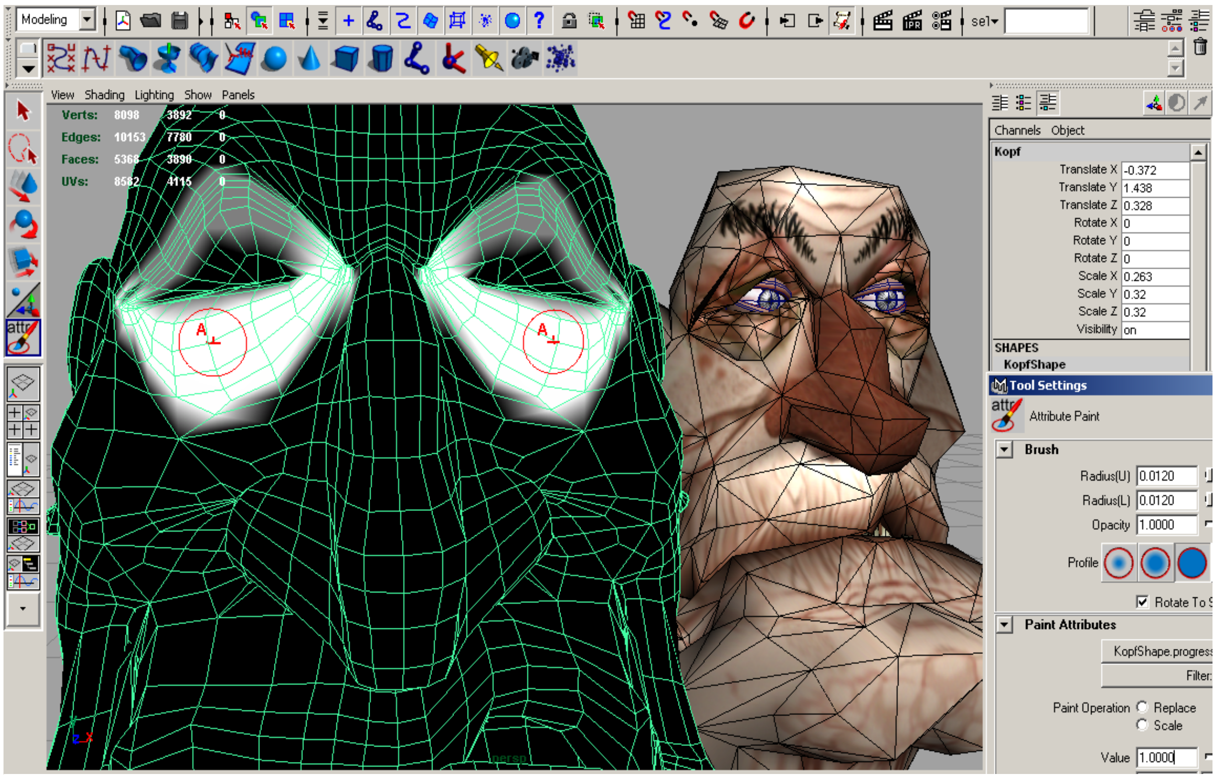
We found that using the average weight produces the most intuitive results. Note that the average has to be computed from all leaf weights in the hierarchy. Therefore we cannot generally use the $\omega(i, j)$ results to compute w_i'.

Moreover, we cannot simply sum up all the weights of the leaf vertices to compute the weight of a root vertex, because it leads to a counter-intuitive importance-weight feedback. Whenever two regions of high importance are merged, the result becomes even more important and thus unattractive for further reduction. As a consequence, other regions get reduced too heavily. This is also a reason for us to keep the weights and the quadrics separated at all times. Although component-wise a priori multiplication of $\mathbf{Q}$ by w leads to identical results with respect to the error metric, it also implies the undesired importance aggregation described above.

3.4 Enhanced deterministic metric

In his QSlim [13] implementation of the quadric error metric, Garland used a heap keyed on the cost of contraction to sort the edge contractions. We found that ordering to be not strict enough for an interactive multiresolution-editing environment.

When editing a multiresolution mesh it is common that the user switches the resolution back and forth very often. For example: if a user is interested in a simplified model representation with a face count of roughly 75% of the original model, the user will frequently switch between the representations in the range of 70% to 80% while editing. This causes edges to get removed and re-inserted into the heap frequently. If any two edges have the same cost of contraction, their relative order in the heap is undetermined. Their ordering would depend on the heap implementation and the insertion order. It is not an uncommon situation, that edges have the same cost of contraction. For example, on a tessellated plane, all edges have the same cost of contraction. In such a situation, a simplified model representation of a fixed resolution would not only depend on the user's editing operations but also on the *order* in which he applied them. Simply increasing the resolution and decreasing it again can produce a slightly different model representation.
This is clearly unintuitive and disturbing for a user, who will expect the simplification process to be deterministic. Therefore we used a stricter ordering for our implementation.



Verts:	8098	3892	0
Edges:	10153	7780	0
Faces:	5368	3890	0
UVs:	8582	4115	0

Kopf

Translate X	-0.372
Translate Y	1.438
Translate Z	0.328
Rotate X	0
Rotate Y	0
Rotate Z	0
Scale X	0.263
Scale Y	0.32
Scale Z	0.32
Visibility	on

SHAPES
KopfShape

Radius(U)	0.0120
Radius(L)	0.0120
Opacity	1.0000
Paint Operation	Replace / Scale
Value	1.0000

Figure 3: The Artisan interface is used to paint the importance of mesh regions on the "Hagen" model. The regions around the two eyes on the source mesh in front are marked simultaneously as important using the reflection tool. The destination mesh in the back is reduced to 15%. Note the preserved detail around the eyes.

Instead of only using the cost of contraction as our ordering criteria, we also include the vertex indices of the edge in the ordering. If two edges have the same contraction cost, the edge with the smaller indices will be placed in front of the other edge. This stricter ordering fulfills our need for deterministic simplification.

4 User Interface

To demonstrate the usefulness of our approach in a real world setting, a Maya plug-in is presented that lets the user create multiresolution meshes with importance weighting interactively and intuitively. To start an editing session, the user needs to create a multiresolution mesh from an existing polygonal mesh. The selected source mesh, with all texture coordinates, normals and vertex colors, is cloned into a multiresolution mesh, which is inserted into the scene. For the editing session, the original source mesh is also left in the scene as a reference.

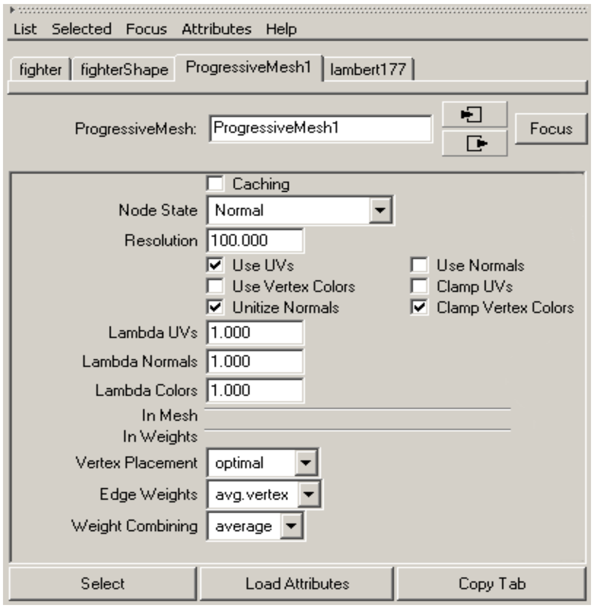
Once a multiresolution mesh has been created, the user can start editing it. There are several global parameters that control the overall reduction process (See Figure 4).

- Resolution: The resolution attribute controls how many faces are used in the approximation of the source model. The resolution is specified in percent.
- Vertex Placement: The vertex placement policy controls the position of the resulting vertex from a pair contraction. There are three vertex placement policies to choose from: optimal, end-or-mid and endpoint. Optimal placement chooses a position that minimizes the error of the contraction. While this produces the best results, it introduces new vertex positions, which can sometimes be problematic for interactive applications. For applications which only want to use the original vertices, endpoint placement is best suited. In this case a vertex pair always gets contracted into one of the original vertices.
- Attribute Controls: Here the user can control the influence of mesh attributes on the reduction process. There are three attribute groups: texture coordinates, normals and vertex colors. Each group can be individually switched on and off. The importance of each group can be specified through a scalar weight factor (lambda terms from [5]).
- Weight Controls: The weight controls let the user choose how the combined, hierarchical vertex weights and the edge weights are calculated. The user can choose between min, max and average.

All attribute changes are immediately applied to the multiresolution mesh. Because these attributes are global parameters of the reduction, changing them means that the simplified model has to be completely recalculated. Usually this takes only little time, because the quadric error metric evaluates very quickly and models for interactive applications like games or simulations usually do not exceed a few thousand polygons.

All attributes are fully accessible by Maya's dependency graph. This means that they can be set, read, connected and animated like any other attribute in Maya. For example one could connect a distance-to-camera evaluator to the resolution attribute to achieve dynamic LOD.

The attributes presented so far were global parameters for the mesh reduction. To specify the importance of certain mesh regions, the user simply paints the weights reflecting the functional or semantic importance directly onto the source mesh. See Figure 3 for an image of the mesh painting in action. The painted weights are displayed color coded directly on the source mesh. The mesh painting is done through Maya's Artisan Interface. Through the use of Artisan, the user has access to a well-known, intuitive and feature-rich user environment (see Figure 6).



List Selected Focus Attributes Help

fighter | fighterShape | ProgressiveMesh1 | lambert177

ProgressiveMesh: ProgressiveMesh1 — Focus

	☐ Caching	
Node State	Normal	
Resolution	100.000	
	☑ Use UVs	☐ Use Normals
	☐ Use Vertex Colors	☐ Clamp UVs
	☑ Unitize Normals	☑ Clamp Vertex Colors
Lambda UVs	1.000	
Lambda Normals	1.000	
Lambda Colors	1.000	
In Mesh		
In Weights		
Vertex Placement	optimal	
Edge Weights	avg.vertex	
Weight Combining	average	

Select | Load Attributes | Copy Tab

Figure 4: Global Parameters of a multiresolution mesh

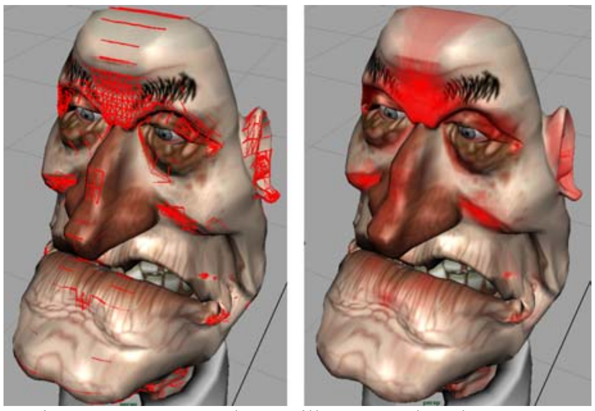
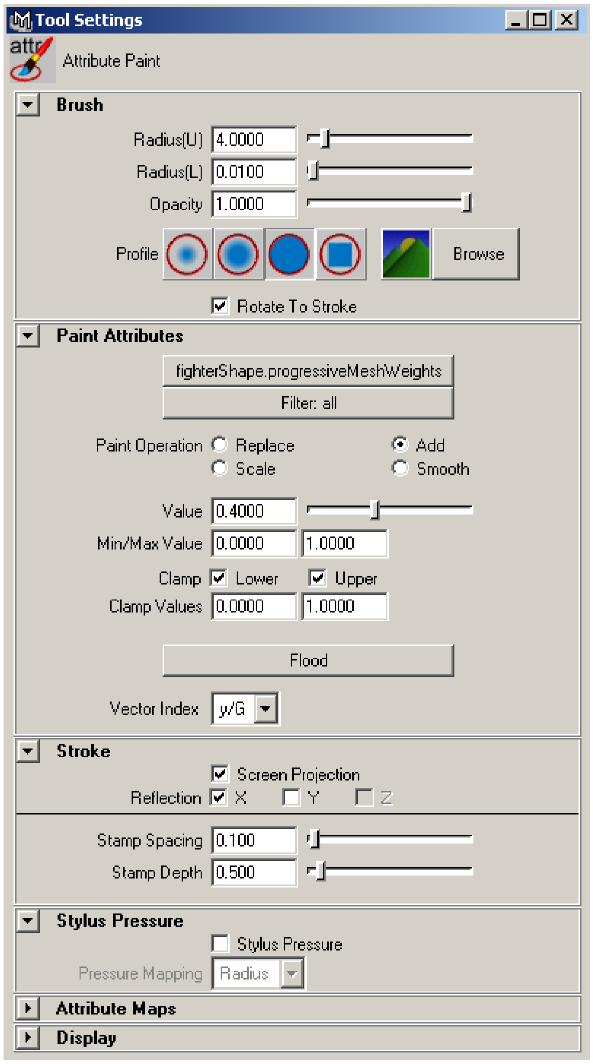


Figure 5: Areas that will get reduced soon are highlighted. The user can choose between two different visualization methods: Lines (left) or Faces (right). Either the line width or the face transparency is used to visualize reduction order.

Artisan features different paint operations (add, replace, scale, smooth), brush styles and sizes, global flood fill operation, mirrored painting on symmetric meshes, support for pressure sensitive input devices etc. An embedded graphical user interface allows the user to easily and freely configure the value range to paint the weights and has numerous display control options.

All weights painted onto the source mesh are immediately reflected in the reduced version of the mesh. Consider the editing situation depicted in Figure 3: the "Hagen" model has been reduced to 15% but the user wants more geometric detail around the eyes. By painting the importance of the eyes onto the source mesh, the geometric detail around the eyes increases as the user paints. However, the overall amount of geometry is left unchanged. The model stays reduced to 15%, these 15% are just redistributed in a different way.

Again, this means that the mesh reduction has to be recalculated, to take the new weights into account. Currently this happens after each brush-stroke. We are aware that for some models this can be a bit slow. Alternatively, a *delayed update* feature allows to freely paint the weights, while the mesh is not recalculated until the corresponding command (hot key release) is given.



Tool Settings

Attribute Paint

Brush

Radius(U)	4.0000
Radius(L)	0.0100
Opacity	1.0000
Profile	Browse
	☑ Rotate To Stroke

Paint Attributes

fighterShape.progressiveMeshWeights

Filter: all

Paint Operation	○ Replace	● Add
	○ Scale	○ Smooth
Value	0.4000	
Min/Max Value	0.0000	1.0000
Clamp	☑ Lower	☑ Upper
Clamp Values	0.0000	1.0000

Flood

Vector Index: y/G

Stroke

	☑ Screen Projection
Reflection	☑ X ☐ Y ☐ Z
Stamp Spacing	0.100
Stamp Depth	0.500

Stylus Pressure

	☐ Stylus Pressure
Pressure Mapping	Radius

Attribute Maps

Display

Figure 6: Artisan user interface for importance painting

When editing a mesh by painting weights onto it, it is often useful to know in advance which regions of the mesh will get reduced next. Usually, the user wants to increase the weights for important regions as long as they are about to get simplified immediately. To provide the user with this information we have implemented a visualization node that highlights mesh areas about to get simplified (Figure 5). The visualization simply highlights the top n (adjustable by the user) edge contractions from the simplification heap.

The user can choose between two visualization modes: lines and faces. Like all other features, the highlighting is also always updated in real-time while the user paints. In effect, this gives the user the possibility to paint weights to remove the reduction highlights from important areas. The user can continue painting until all the important regions are not highlighted any more and less important regions are highlighted for reduction instead.

5 Results

In this section we present some examples of user-controlled creation of multiresolution meshes. All examples were produced using our Maya plug-in. Figure 7 shows the "Fighter" model reduced to 15% (1044 triangles). The model was simplified completely automatic. Texture coordinates where taken into account in the simplified model. Although the result is not bad, there are some problematic areas:

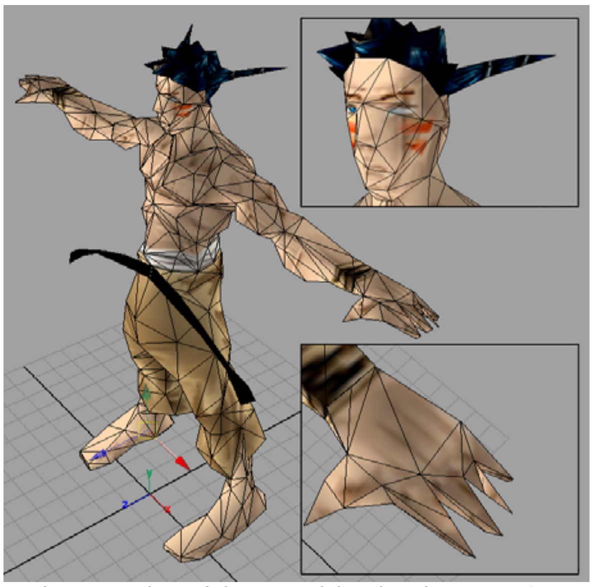


Figure 7: The "Fighter" model reduced to 15% (1044 triangles) with no weighting.

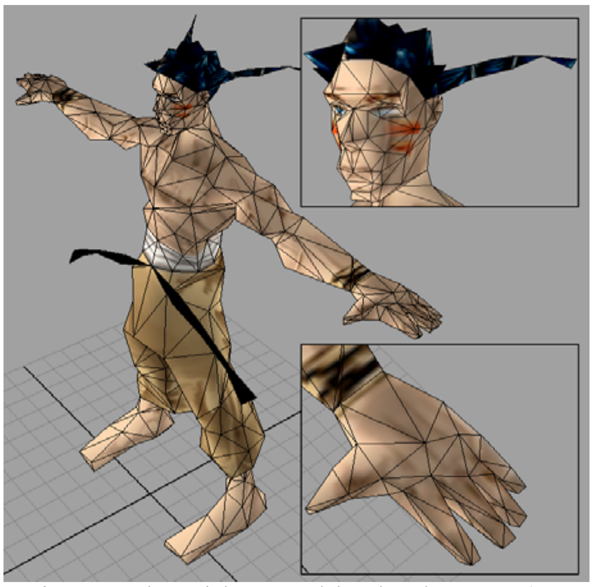


Figure 8: The "Fighter" model reduced to 15% (1044 triangles), using importance weighting

The fingers start to degenerate to single triangles, a crack in the trousers around the ankles has appeared, the ears are gone, the braids are shortened and the face has lost a lot of its original detail. Figure 8 illustrates the kind of improvement achievable through the use of importance weighting. Figure 9 shows the original model and the applied weights: the face and the ears are marked as most important. The hands are also marked as important regions, but the applied weights are smaller than for the face. The smallest weights have been applied to some parts of the trousers and the braids. Through the weighting we could improve the quality of the face, the fingers and the braids and also removed the crack in the trousers. Notice that the models detail has not been increased; the available polygon budget has only been redistributed differently.

For animated meshes, the deformable regions usually are of special functional importance. Deformed regions tend to get stretched and squashed during animation and therefore often need to keep more geometric detail than the surrounding rigid regions. See Figure 10 for an illustration of the problem. Here the "Fighter" model was first simplified and then deformed using a skeleton. The resulting deformation could be improved by weighting the vertices in the elbow region.

Figure 11 shows a different scenario: Here the skeleton was not bound to the simplified model but to the source model instead. While the model is animated a simplified mesh (reduced to 50%) is created for each animation frame. Because the simplification algorithm uses the already deformed source model as input, the deformed regions get reduced already taking the deformation into account. In effect, each simplified model for each frame is slightly different. In such a case we do not need to weight areas to account for functional importance. However, one still might want to weight other regions of high semantic importance.

Finally, Figure 12 is another example of functional importance. Facial animation is applied to the "Old Man" model. Without proper weighting, the simplified version folds over in several places and the mouth region is distorted heavily. Even though the weights where applied very quickly and uniformly the resulting simplified model is much more faithful to the original: the mouth is deformed properly and the folded mesh areas are gone. (Figure 12, right).

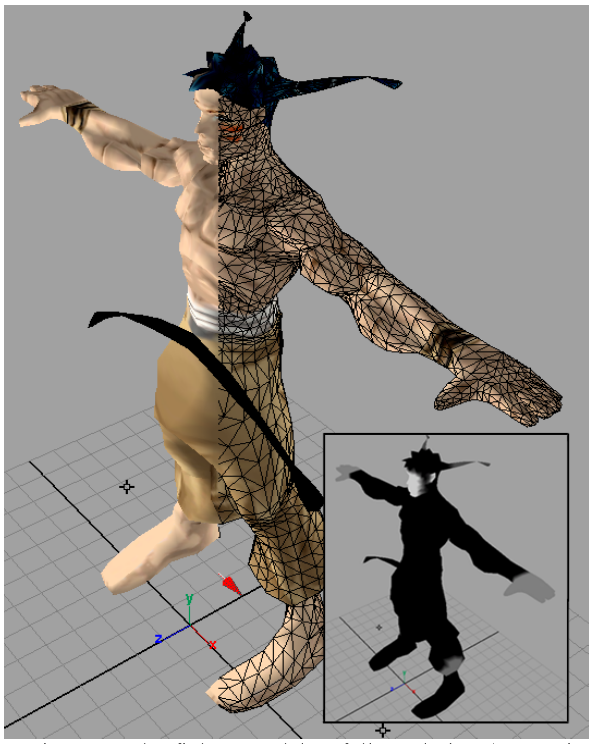


Figure 9: The fighter model at full resolution (6952 triangles). The inset shows the visualization of the importance weights. The face and the ears are marked as being more important as the hands.

6 Summary and future work

We have described a weighted quadric error metric for the user-controlled creation of multiresolution meshes. The user is enabled to control the simplification of a mesh by interactively painting the importance of regions as weights onto the mesh. This enables the user to improve the quality of multiresolution meshes by taking semantic and functional importance into account. While we used weights with a quadric cost metric, user-controlled weighting should be easily applicable to all mesh simplification algorithms that use iterative simplification driven by a cost metric. For algorithms that do not produce vertex hierarchies, the combined weights need to be calculated differently, but a modified algorithm for that is straight forward.

To demonstrate the usefulness of our approach in a real world setting, a Maya plug-in for the creation and manipulation of multiresolution meshes was presented. The plug-in can operate on arbitrary (non manifold) meshes with arbitrary mesh attributes. To release our work as a Maya plug-in is both an attempt to bridge the gap between academic research and real-world applications and also an attempt to provide a highly useful tool. By embedding a method for user-controlled polygonal simplification into an existing modeling package, both the polygonal reduction tool and the modeling package are enhanced, and the user gains most.

In future work, it would be desirable to further enhance the user interface and to add the possibility of the following editing features: (weighted) positional constraints for vertices, enabling the user to lock the position of certain vertices and a possibility to exclude certain vertices and edges from the reduction; this could be desirable for borders connecting to other meshes. Also, we would like to include an interface for selecting arbitrary vertex pairs, which also should be contracted during simplification.

We would also like to investigate methods for automatic or assisted creation of importance weights: For animated meshes, it should be possible to deduce higher weights for deformable regions by either sampling the animated mesh or by analyzing the weights for skeletal animation. Another possibility for automatic weight generation may be to analyze the local texture density of a mesh. Models created by hand tend to have higher texel density in areas of high importance.

7 Acknowledgments

We would like to thank Tobias Mayr for providing the "Fighter" model and Christian Haas for the "Hagen" and "Old Man" models. The "Cow" model is freely available from ViewPoint DataLabs. (http://avalon.viewpoint.com). The second author is supported by the Austrian Science Fund *FWF* under grant Y193.

8 Software

The presented Maya plug-in and its source code is available for download from http://www.pojar.net/ProgressiveMesh/

9 References

[1] David P. Luebke, Carl Erikson. View-Dependent Simplification of Arbitrary Polygonal Environments. Computer Graphics (SIGGRAPH '97 Proceedings), pages 199-208.

[2] David P. Luebke. A Developer's Survey of Polygonal Simplification Algorithms. IEEE Computer Graphics & Applications, 2001

[3] Gong Li, Benjamin Watson. Semiautomatic Simplification. Symposium on Interactive 3D Graphics, pages 43-48, 2001.

[4] Hugues Hoppe, Steve Marschner. Efficient Minimization of New Quadric Metric for Simplifying Meshes with Appearance Attributes.

[5] Hugues Hoppe. New Quadric Metric for Simplifying Meshes with Appearance Attributes. In David Ebert, Markus Gross, and Bernd Hamann, editors, IEEE Visualization '99, pages 59-66. IEEE, October 1999.

[6] Hugues Hoppe. Progressive Meshes. Computer Graphics (SIGGRAPH '96 Proceedings), pages 99-108, 1996.

[7] Hugues Hoppe. View-dependent refinement of progressive meshes. Computer Graphics (SIGGRAPH '97 Proceedings), pages 189-198, 1997.

[8] Julie C. Xia, Amitabh Varshney. Dynamic view dependent simplification for polygonal models. Computer Graphics (SIGGRAPH '96 Proceedings), volume 30(4), 1996.

[9] Leif Kobbelt, Swen Campagna, Hans-Peter Seidel. A General Framework for Mesh Decimation. In Proceedings of Graphics Interface, pages 43-50, 1998.

[10] Michael Garland, Paul S. Heckbert. Simplifying Surfaces with Color and Texture using Quadric Error Metrics. IEEE Visualization '98 Proceedings, pages 263-269, 1998.

[11] Michael Garland, Paul S. Heckbert. Surface Simplification Using Quadric Error Metrics. Computer Graphics (SIGGRAPH '97 Proceedings), pages 209-216, 1997.

[12] Michael Garland. Multiresolution Modeling: Survey & Future Opportunities. Eurographics '99 - State of the Art Reports, pages 111-131, 1999

[13] Michael Garland. QSlim Simplification Software. http://graphics.cs.uiuc.edu/~garland/software/qslim.html

[14] Paolo Cignoni, Claudio Montani, Claudio Rocchini, Roberto Scopigno. Zeta: a Resolution Modeling System, Graphical models and image processing: GMIP, volume 60, pages 305-329, 1998.

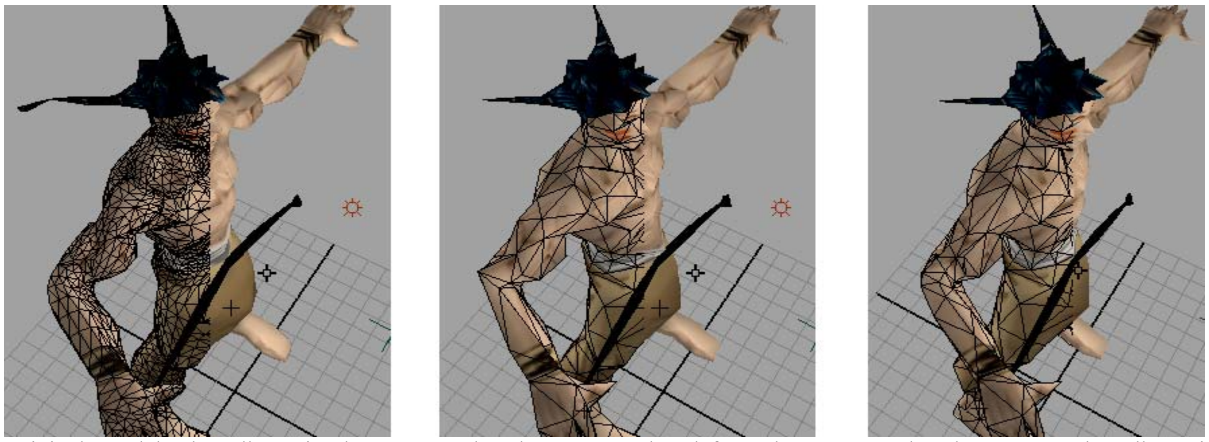
[15] Paul Steed. The art of low-polygon modeling. Game Developer, pages 62–69, June 1998, http://www.gdmag.com/backissue1998.htm#jun98.

[16] Peter Lindstrom, Greg Turk. Image-Driven Simplification. ACM Transactions on Graphics, 19(3), pages 204-241, July 2000.

[17] Pierre Alliez, Mark Meyer and Mathieu Desbrun: Interactive Geometry Remeshing. Proc. SIGGRAPH 2002

[18] Rémi Ronfard, Jarek Rossignac: Full-range Approximation of Triangulated Polyhedra. Computer Graphics Forum 15(3): 67-76

[19] Stephen White. Postmortem: Naughty Dog's Jak & Daxter: The Precursor Legacy by Stephen White. Game Developer magazine, April 2002, http://www.gamasutra.com/features/20020710/white_01.htm



Original model, the elbow is deformed through skeletal animation. | Reduced to 15%, the deformed elbow is stretched and squashed. | Reduced to 15%, the elbow is improved through weights.

Figure 10: An example of functional importance. Weighted deformable regions can improve the quality of animated meshes.

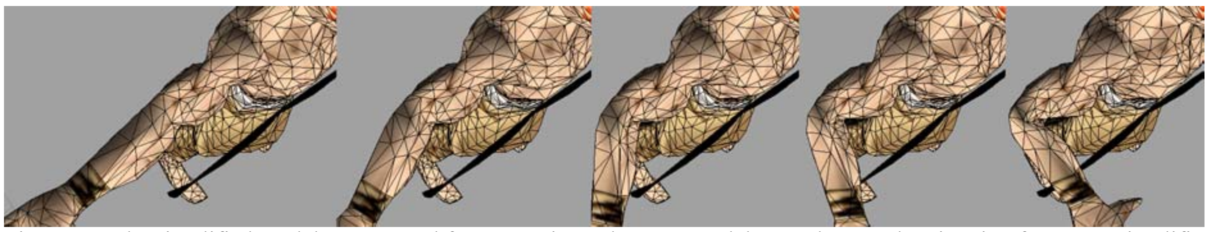
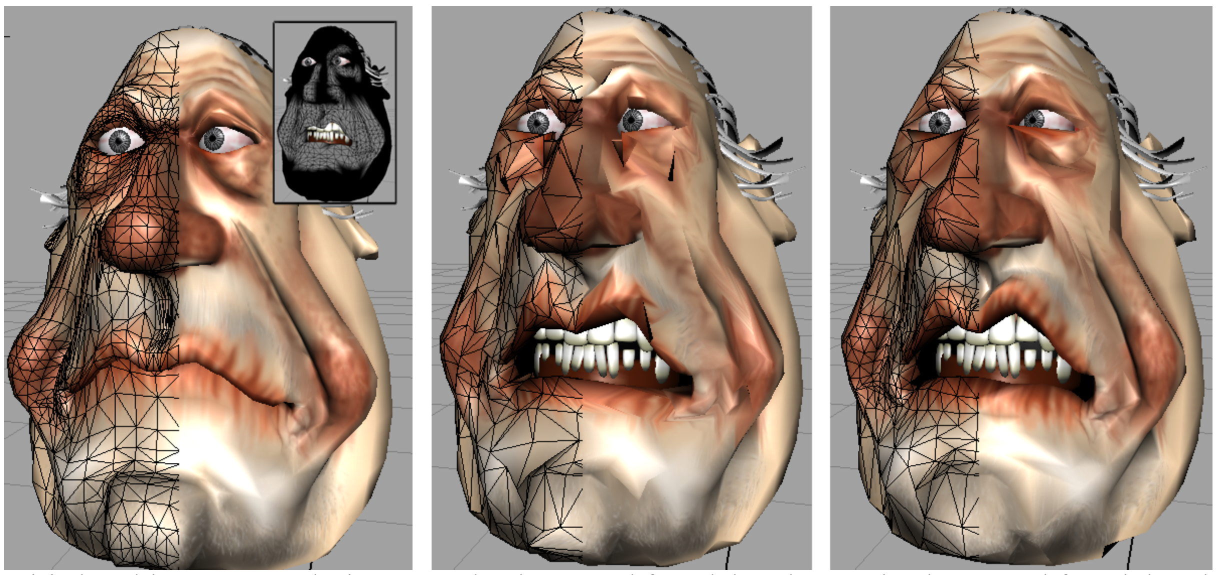


Figure 11: The simplified model was created from an animated source model. Note how each animation frame got simplified slightly different, adapting to the deformation of the elbow. No weights where used.



Original model at rest pose, the inset shows the importance weights | Reduced to 50%, deformed through facial animation. No weights applied | Reduced to 50%, deformed through facial animation. Weights applied

Figure 12: The „Old Man“ model reduced to 50%, deformed through facial animation. The middle image shows the results without importance weighting, the right image shows the improved results after importance weights have been applied.