User-Controlled Simplification of Polygonal Models.

Conference Paper · January 2004

DOI: 10.1109/TDPVT.2004.1335413 · Source: DBLP

3 authors, including:

Muhammad Hussain
King Saud University
110 PUBLICATIONS **464** CITATIONS

User-Controlled Simplification of Polygonal Models

Muhammad Hussain[1,2], Yoshihiro Okada[1,2], and Koichi Niijima[1]
[1]Graduate School of Information Science and Electrical Engineering, Kyushu University,
6-1, Kasuga Koen, Kasuga, Fukuoka 816-8580, Japan.
{mhussain, okada, niijima}@i.kyushu-u.ac.jp
[2]Intelligent Cooperation and Control, PRESTO, JST

Abstract

Polygonal Models are ubiquitous in Computer Graphics but their real time manipulation and rendering especially in interactive application environments have become a threat because of their huge sizes and complexity. A whole family of automatic algorithms emerged during the last decade to help out this problem, but they reduce the complexity of the models uniformly without caring about semantic importance of localized parts of a mesh. Only a few algorithms deal with adaptive simplification of polygonal models. We propose a new model for user-driven simplification exploiting the simplification hierarchy and hypertriangulation model [1] that lends a user the most of the functionalities of existing adaptive simplification algorithms in one place and is quite simple to implement. The proposed new underlying data structures are compact and support real time navigation across continuous LODs of a model; any desirable LOD can be extracted efficiently and can further be fine-tuned. The proposed model for adaptive simplification supports two key operations for selective refinement and selective simplification; their effect has been shown on various polygonal models. Comparison with related work shows that the proposed model provides combined environment at reduced overhead of memory space usage and faster running times.

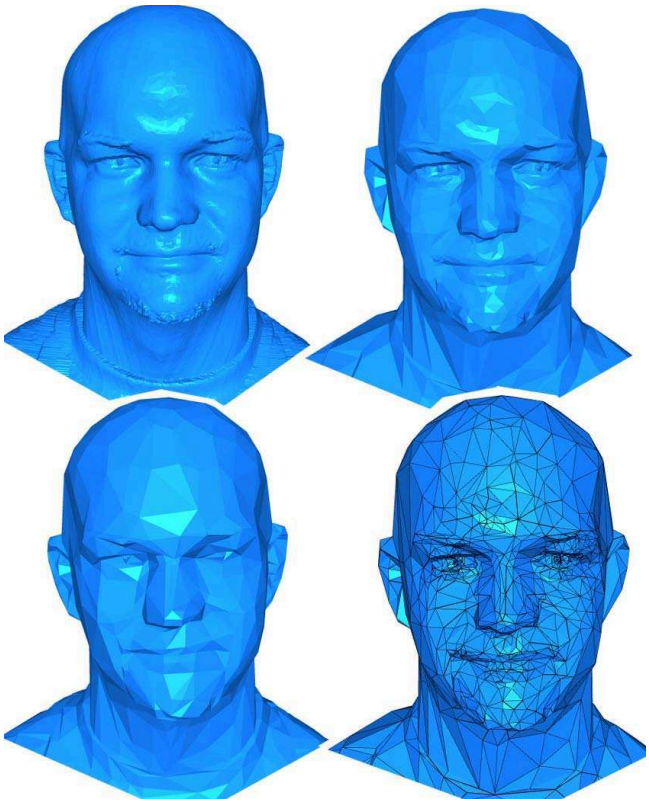

Figure 1: Male model, detailed version (top left), adaptively simplified version (top right, bottom right) #faces 2000, an LOD at uniform resolution (bottom left) #faces 2000.

1 Introduction

In various application areas of Computer Graphics, visualization and manipulation of real world data sets relies on polygonal models where a data set is typically specified with triangle-based surface mesh. The quest for realism on one hand and the enhanced ability to acquire polygonal models with arbitrary accuracy on the other hand have lead to highly complex and huge polygonal meshes, which are hard to store, manipulate and visualize in real time; real time rendering of such models especially in interactive applications is a challenging problem in CG. One approach to alleviate this problem was the proposal of LOD and Multiresolution paradigms where an object is stored at k different (discrete/continuous) levels of detail; a higher resolution version is employed when the object is close to the viewer and coarser version is substituted as the object recedes [2, 3]. This approach necessitated the need of polygonal

Proceedings of the 2nd International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT'04)
0-7695-2223-8/04 $ 20.00 IEEE

simplification algorithms and throughout the last decade, a whole family of automatic polygonal simplification algorithms came into existence [4–6, 8, 12, 15–17, 20, 21]. While they produce very appealing results in many cases, they perform poorly at extremely low levels of detail because they are blind to semantic or high level meanings of a model and simplify it uniformly without caring about whether its some localized area visually more important that others. To overcome the shortcomings of uniform automatic simplification, methods for adaptive simplification [10, 13, 22] were developed but here the focus is on the exploitation of view space constraints and so their scope is limited to applications where viewing space impact is important. In many applications it is very important that a model conveys overall, not only with respect to a certain view space, an optimized knowledge about an object even at extremely low levels of detail, and it can not be accomplished without user interaction. For example, in case of human body, face is visually important and face features must be preserved even at low levels of detail and other parts of the body such as torso or legs can be reduced drastically. In this situation some kind of user interaction is essential.

The algorithms proposed in [1, 9, 19, 23] address the problem from user-interaction point of view. The algorithms presented in [9, 19, 23] rely on *simplification hierarchy* whereas Zeta [1] employes *hypertriangulation model*; the algorithms proposed in [19, 23] provide high level control whereas Semisimp [9] and Zeta [1] provide low level control to a user. Zeta [1] provides relatively better control to a user but it is limited because it takes pre-computed multiresolution representation of a polygonal model as input and then builds a hypertriangulation model for resolution modeling. In this paper we propose a unified framework employing simplification hierarchy and hypertriangulation model that provides low level as well as high level control to a user. The proposed new underlying data structures are compact and support the construction of simplification hierarchy simultaneous with hypertriangulation model creation in one pass thereby extending the functionality of Zeta [1] to that of methods based on simplification hierarchy, with reduced overhead of memory space occupancy and faster running times; these data structures support real time navigation across continuous LODs of a polygonal model, extraction of any desirable fixed LOD and selectively refining and selectively coarsening it.

The overall organization of the paper is as follows. In the following section, we give a review of the most related algorithms. Section 3 describes in detail a new unified framework for adaptive simplification, and the construction algorithm has been detailed in Section 4. Techniques for navigation across continuous LODs, selective refinement and selective simplification have been presented in Section 5. Results have been discussed in Section 6 and Section 7 concludes the paper.

2 Related Work

A large number of algorithms for automatic polygonal simplification has been proposed, for through survey of these methods an interested reader is referred to [7, 11]. Here we restrict ourselves to an overview of algorithms addressing adaptive simplification of polygonal models.

Only a few algorithms exist in the literature, which allow user-controlled simplification of polygonal models. Zeta [1] is the first algorithm dealing with user-driven simplification and exploits a new type of multiresolution representation called *hypertriangulation model*, which allows to navigate selectively through continuous LODs of a model, and moving efficiently between adjacent faces on an LOD. A model at any fixed resolution can be extracted efficiently and can further be selectively refined /simplified by locally changing error thresholds. While it provides an elegant control to the user, its inability to take a given original model as input makes its usefulness limited; it requires a pre-computed multiresolution representation as input. Also, it can modify geometry locally but this change can not be propagated across other levels because it does not support any kind of simplification hierarchy.

Semisimp [9] is another tool for user-guided simplification; it takes an original detailed model as input and constructs a simplification hierarchy; any LOD is a cut across this hierarchy. It gives low level control to a user to redistribute detail on simplified models by changing the order of simplification hierarchy and to improve the position of vertices; the change in position of vertices is propagated across other levels of detail in a multiresolution editing fashion. Further, unlike Zeta, it allows a user to partition an input model to match the semantics of the model and simplify the model in a segmented fashion.

The common denominator for Zeta and Semisimp is that both provide low level control to the user. In case of Semisimp, while modifying the order of simplification hierarchy, the partial order of the hierarchy must be maintained, which restricts the editing operations, whereas Zeta does not rely on simplification hierarchy but on hypertriangulation model, and so it allows to reorder the simplification operations more freely.

The algorithms developed independently by Pojar et al. [19], and Kho et al. [23] are the most recent methods addressing the problem of user-guided simplification. They rely on simplification hierarchy just like Semisimp and use quadric error metric to create the hierarchy; to achieve controlled simplification, weights are associated with the quadric of each vertex and simplification is performed in an automatic fashion. So, both provide high level control for a user, where he/she controls the simplification hierar-

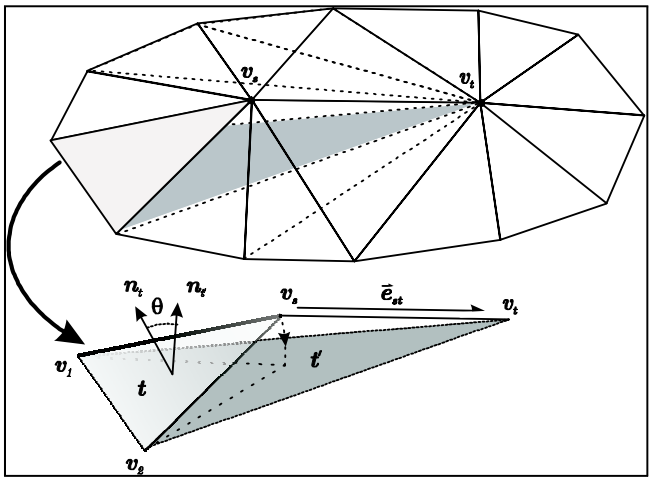

Figure 2. Half-edge collapse transformation $\vec{e}_{st}\ (v_s, v_t) \mapsto v_t$ **.**

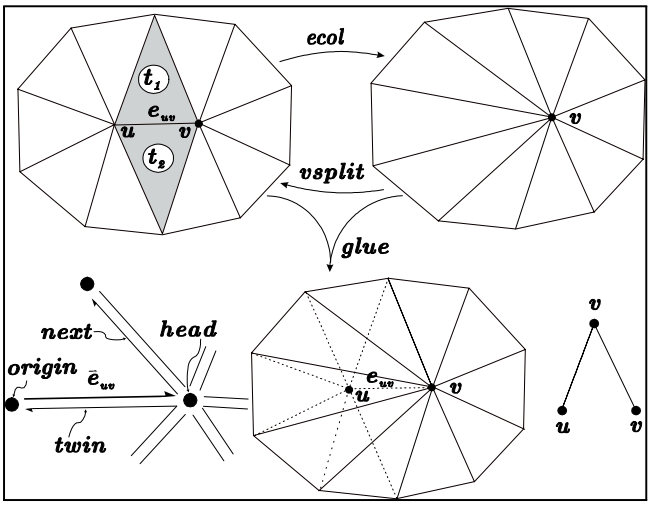

Figure 3. *ecol* **and** *vsplit* **transformations (top row), corresponding nodes of vertex hierarchy (right bottom row) and glue operation (middle bottom row). A half-edge and its adjacencies (left bottom row).**

chy by embedding additional information into the simplification metric. The main shortcoming of these algorithms is that a user needs to repeat the simplification process changing weights several times until he/she is satisfied with the approximation, to get rid of this weakness low level control is necessary.

Our algorithm exploits simplification hierarchy as well as hypertriangulation model, and it combines most of the advantages put forward by user-controlled simplification algorithms based on simplification hierarchy and that based on hypertriangulation model.

3 Adaptive Simplification Model

In this section, by the synthesis of simplification hierarchy and hypertriangulation model [1], we propose a unified framework for the construction of sophisticated multiresolution mesh representation of orientable , 2-manifold polygonal models in $\Re^3$; hereafter we call it *adaptive simplification model (ADSIMP).*

3.1 Simplification Hierarchy

Automatic simplification algorithms based on edge-collapse have become attractive because of their elegant features. Edge collapse (*ecol*) and vertex split (*vsplit*) transformations build naturally a vertex hierarchy; Figure 2 shows *ecol* and *vsplit* transformations and the associated nodes of the vertex hierarchy. *ADSIMP* exploits half-edge collapse transformation for simplification because it behaves like both edge-collapse and vertex decimation but does not create new vertices like half-edge collapse and needs not any optimization algorithm like vertex decimation to fill the hole, and the resulting vertex hierarchy can be encoded with as many nodes as there are vertices in the input mesh M.

The sequence of half-edge collapse transformations is driven by a memory-efficient and feature preserving error metric proposed in [16]; according to this the cost of the half-edge collapse transformation $\vec{e}_{st}\ (v_s, v_t) \mapsto v_t$ (see Figure 1) is

$$Cost(\vec{e}_{st}) = \sum Q_t.$$

where $Q_t = l_t \cdot \theta_t$ with $l_t = \frac{1}{2}(A_1 + A_2)$, $A_1 =$ *area of triangle* $t\ (v_1, v_2, v_s)$, $A_2 =$*area of triangle* $t'(v_1, v_2, v_t)$, and θ_t is the angle described by the normal of the triangle t when edge e_{st} is collapsed, and summation is taken over all triangles incident on v_s. For detailed account, consult [16].

3.2 Hypertriangulation Model

Hypertriangulation model proposed by Cignoni et al. [1] is of particular relevance for us because our system also exploits this model. The concept of hypertriangulation is based on the idea of *gluing*. At *i*-th iteration of an automatic simplification algorithm, a patch of triangles T_u is removed from M_{i-1}, an intermediate mesh obtained after $i-1$ iterations, and new triangulation $T_u^{'}$ replaces it in M_i. The idea of gluing is to past the new set of triangles $T_u^{'}$ over the old triangles T_u , see Figure 2.

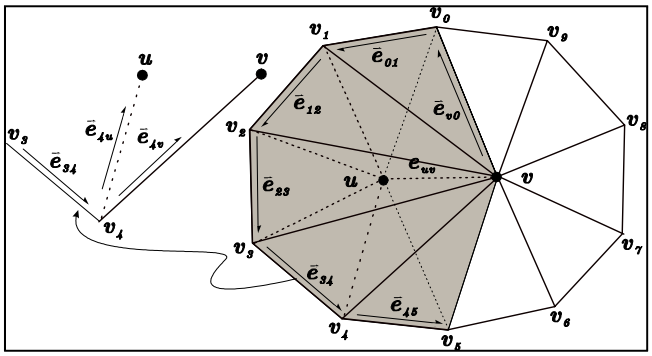

Figure 4. Half-edge collapse transformation $\vec{e}_{st}$ $(v_s, v_t) \mapsto v_t$ removes the patch T_u of triangles incident on u shown in shaded region, and adds new patch $T_u^{'}$ of triangles also shown in shaded region with solid line segments.

Half-edge collapse transformation employed by *ADSIMP* naturally and elegantly builds hypertriangulation model simultaneous with vertex hierarchy creation; this transformation removes triangles from a small localized region and introduces new triangles to fill the hole following a specific automatic pattern, so gluing operation can trivially be accomplished by updating just a few adjacencies.

Consider Figure 3, half-edge collapse transformation $\vec{e}_{uv}$ $(u, v) \longmapsto v$ replaces current patch T_u of triangles incident on vertex u with a new patch $T_u^{'}$; patch $T_u^{'}$ is glued over T_u by modifying the next adjacent relations of the half-edges $\vec{e}_{v0}, \vec{e}_{01}, \vec{e}_{12}, \vec{e}_{23}, \vec{e}_{34}, \vec{e}_{45}$ along the boundary of each patch e.g. after gluing, next adjacent half-edge $\vec{e}_{4u}$ of the boundary half-edge $\vec{e}_{34}$ will become inactive and $\vec{e}_{4v}$ will become active. Observe that when gluing is accomplished, there are two half-edges next to each of the half-edges along the common boundary of patches T_u and $T_u^{'}$; a list of variable size is associated with each half-edge to store reference to these next adjacent half-edges. Pasting is carried out by storing a new next adjacency in this list, setting it active and setting current next adjacency inactive.

3.3 Multiresolution Data Structures

Vertex hierarchy and hypertriangulation model described in the previous subsections is encoded by two entities of data structures: *Vertex* and *PackedEdge*; *Vertex* encodes geometrical and topological information about a vertex, and *PackedEdge* encodes a half-edge and its adjacencies in *ADSIMP*; hereafter, we will use *PackedEdge* and half-edge interchangeably. Their representation is C++ format is as follows.

```
struct Vertex {
        float           position[3];
        PackedEdge*     pe;
        int*            children;
        float           cost;
        int             heapspot;
        unsigned short  ch_indx;
        int             parent ;
        bool            mark;
};
```

Here *position* field holds three coordinates representing the geometric position of the corresponding vertex, and *pe* is a pointer to the associated *PackedEdge* that makes possible the traversal of all vertices in 1-ring neighborhood of this vertex and its adjacencies, it is representative of the half-edge whose collapse will remove this vertex, we term this as *optimal half-edge* because its collapse causes minimum geometric deviation; *children* and *parent* fields encode simplification hierarchy, *children* is a variable length array that holds the pointers to those vertices that will collapse to this vertex in the order of their simplification and *parent* holds the pointer to the vertex to which this vertex will be collapsed; *mark* field is used to hold the information whether this vertex is active or not in an LOD, if vertex is chosen for display it is active otherwise inactive. The remaining two fields *cost* and *heapspot* hold the cost of collapse of the half-edge *pe* and the position of the vertex in the priority queue respectively; each half-edge collapse transformation removes one vertex, so the cost of a half-edge can also be regarded as cost of the vertex , which will be simplified, that is why we store the cost in vertex data structure.

```
struct PackedEdge {
        int             origin;
        PackedEdge*     twin;
        PackedEdge**    next;
        unsigned short  idx_next;
        bool            mark;
};
```

Here *origin* stores the index of the starting vertex (origin) of the corresponding half-edge, *twin* is its mate that shares the common edge with it (see Figure 2), *next* is a variable length array that holds in order the pointers to those half-edges which are next to this half-edge in *ADSIMP* across different resolution levels (see Figure 3), its size depends on the number of half-edge collapse transformations which have this half-edge on the boundary of the simplification region, one of these half-edges would be active at a time and *idx_next* holds the index of this active half-edge; *mark* field is used to mark and unmark the PackedEdge.

Note that there is no need of data structure to explicitly specify facets, they are implicitly defined by the adjacencies which are encoded in PackedEdge data structure.

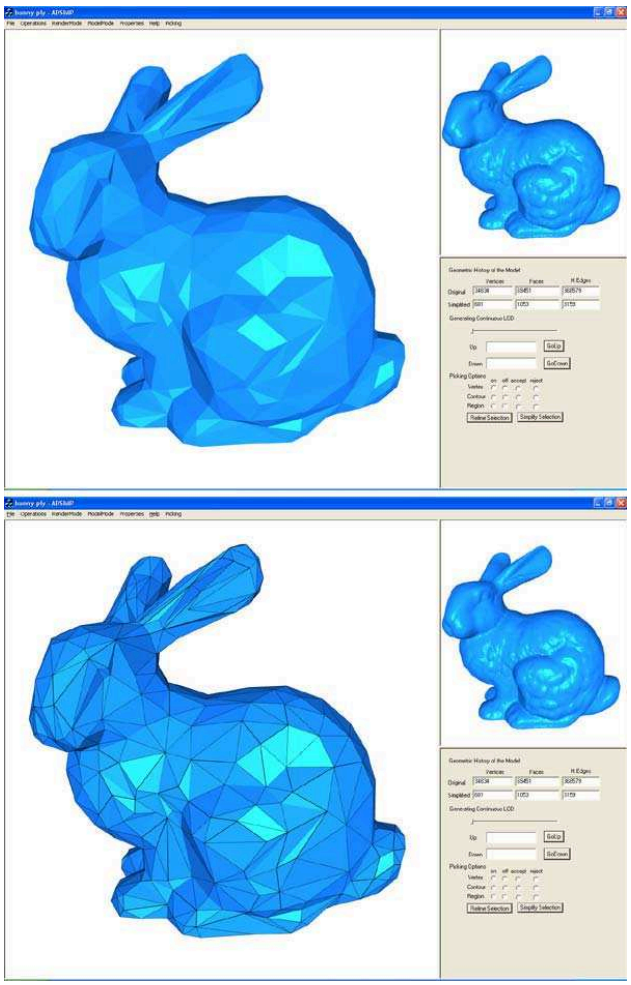

Figure 5. An LOD of bunny model at uniform resolution, #faces 1009. Small window shows original model.

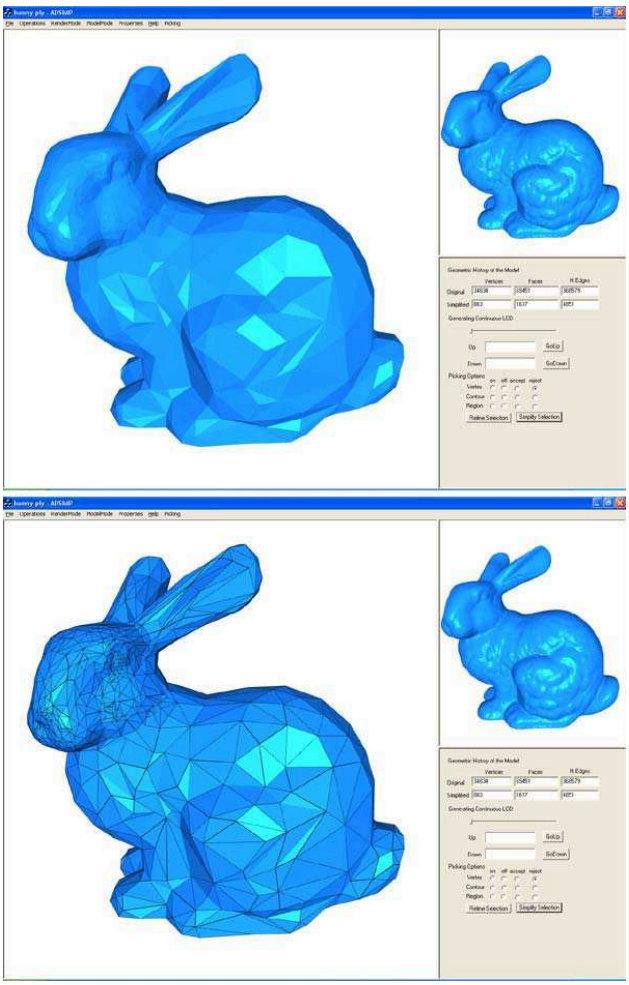

Figure 6. Adaptively simplified version of the bunny model #faces 1504.

4 Construction Algorithm

The construction algorithm for *ADSIMP* described in Section 3 takes original fully detailed polygonal mesh M as input and performs the following steps:

1. Initialize *ADSIMP* by creating *PackedEdge* record for each half-edge $\vec{e}$ and *Vertex* record for each vertex u of the original input mesh M.

2. For each vertex $u \in M$, determine optimal half-edge $\vec{e}_{uv}$, where u is origin and v is head of $\vec{e}_{uv}$ (see Figure 2); this is the one among out-going half-edges of u whose collapse removes u and causes minimum geometric deviation (Section 3.1). Cost of the corresponding optimal half-edge is the cost of the vertex.

3. Put each vertex in the priority queue taking cost of the vertex as priority value.

4. Remove minimum cost vertex u from the queue, collapse $\vec{e}_{uv}$ by putting index of u in *children* field of v and putting index of v in *parent* field of u, and weld the new patch $T_u^{'}$ onto the current patch T_u (Section 3.2)

5. Re-evaluate the cost of each vertex in 1-ring neighborhood of u, and update the priority queue.

6. Repeat steps 4 and 5 until no vertex can be removed.

Note that not all edges are considered legal for collapse; for a detailed discussion on the legality of edge collapses, refer to [14].

5 Selective Refinement and Selective Simplification

ADSIMP supports two key methods to traverse the hierarchy of continuous LODs of a mesh and these are *refine_vertex*() and *simplify_vertex*(); these methods perform *ecol* and *vsplit* operations just by modifying a few next adjacent half-edge references. These methods allow navigation across continuous LODs of a mesh and extraction of any desirable fixed LOD. Further these methods can implement selective refinement and selective simplification by adding/removing detail from a localized region.

We maintain an order list that preserves the simplification order of the vertices and navigation across continuous LODs is accomplished by moving up or down the list. Each cell of this list holds only the index of a vertex and the counter of this cell specifies the order of simplification of the vertex. Moving up or down the list is realized by *refine_vertex*() and *simplify_vertex*() respectively and a pointer that points to a particular cell; the vertices whose indices are hold by all the cells above this cell will be active and all other vertices would be inactive. In this way each cell is representative of a cut in the simplification hierarchy and represents an LOD. A user chooses a desirable LOD by moving the pointer up or down the order list. If some localized parts of the selected LOD are not pleasing, these are further selectively refined or selectively simplified.

Selective simplification involves reduction of some vertices and is achieved by *simplify_vertex*(). The user selects a vertex or a small region interactively and the method *simplify_vertex*() is called to simplify the selected vertex or region.

Selective refinement is realized by *refine_vertex*() and is a little bit involved. A vertex and its child encode *vsplit* and selective refinement exploits this fact of the simplification hierarchy. A localized region of the chosen LOD is selected interactively and the child of each vertex in the selected region is split by *refine_vertex*() method. However, there is a limitation when a vertex is split; if all vertices in the 1-ring neighborhood of a vertex are not active, then it can not be split, so first we have to activate those neighbor vertices which are not active by splitting them.

6 Discussion

We implemented *ADSIMP* using C++, MFC classes, and OpenGL on a system with PentiumIV 2.9GHz and main memory 512MB. Figures 5 and 6 show snapshots of our system with original bunny model and an LOD at uniform resolution, and its adaptively refined version. To comprehend the capability of the system, see also Figures 1, 7, and 8. We can efficiently navigate through different levels of detail at run time using the slider, can extract any fixed LOD and can further fine tune it. In the following, we discuss space and time complexity of our model.

Vertices: 34,834		Faces: 69451		PackedEdges:371,04	
l	1	2	3	4	5
f	138,902	142,523	65,879	19,031	3,436
l	6	7	8	9	10
f	1094	126	36	10	2

Table 1. *PackedEdge* **history for bunny model.** l **stands for the length of next field and** f **is its frequency. Average length is 1.9.**

6.1 Space Complexity

Let *n* be the total number of vertices and *m* the total number of half-edges (*PackedEdges*) in *ADSIMP*. In each *Vertex* record, *children* is a variable length field which holds the number of vertices that will be collapsed to the corresponding vertex; since each vertex is removed once, so the total size of all *children* fields will not exceed *n* $i.e.$ average length of each field does not exceed 1. As such, list of *Vertex* records will occupy $35n$ bytes of memory. Each half-edge collapse transformation will add 2(r-3) half-edges, where r is the valence of the vertex to be collapsed and the total number of legal edge collapse transformations can not exceed n. So, assuming the average valance to be 6, the number of half-edges introduced by half-edge collapse transformations will not exceed $6n$. Also, since in a manifold mesh, number of half-edges is about $6n$, therefore, the total number of half-edges does not exceed $12n$ i.e. $m \leq 12n$. We found empirically that for bunny model $m \approx 10n$, see Table 1. In *PackedEdge* record, *next* is a variable length field and its average length is found empirically to be 1.9 for bunny model. Thus, the memory size occupied by *PackedEdges* is about $228n$ bytes, and the overall memory size occupied by *ADSIMP* is at the maximum $263n$ bytes. But in case of Zeta, this size is $347n$. It means that our proposed model consumes 24% less memory. Although, the authors of the methods proposed in [9, 19, 23] have not reported memory space occupancy, the method of Youngihn et al. associates two quadric error metrics with each vertex and this requires $80n$ bytes only for error metrics in addition to the memory required for geometry and connectivity of the mesh.

6.2 Time Complexity

Construction of *ADSIMP* takes constant time to compute cost for each optimal half-edge and the time complexity of the priority queue is O($n$log$n$). Thus theoretically the time

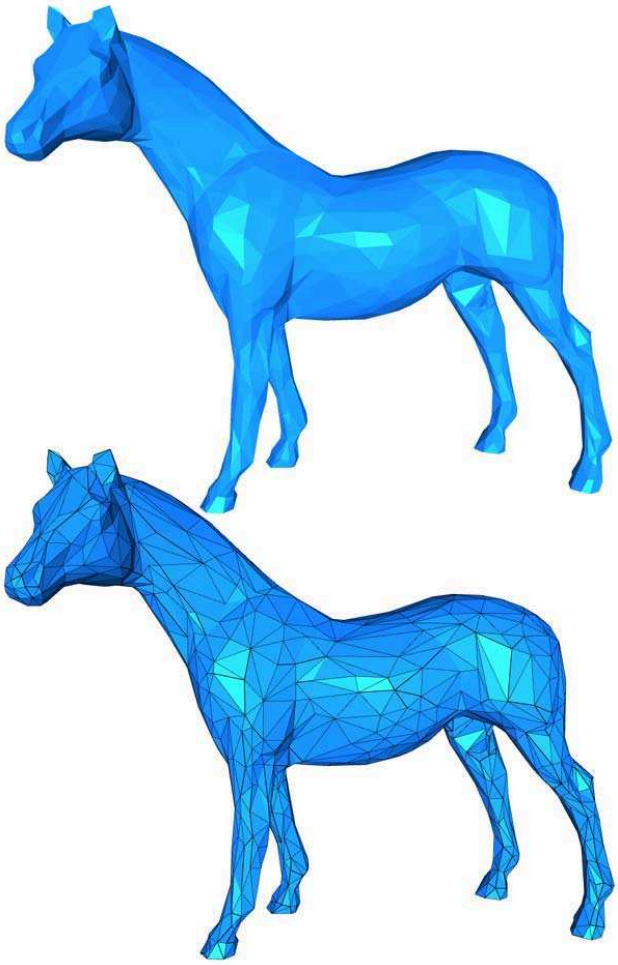

Figure 7. A uniform LOD of horse model, uniform resolution #faces 1996.

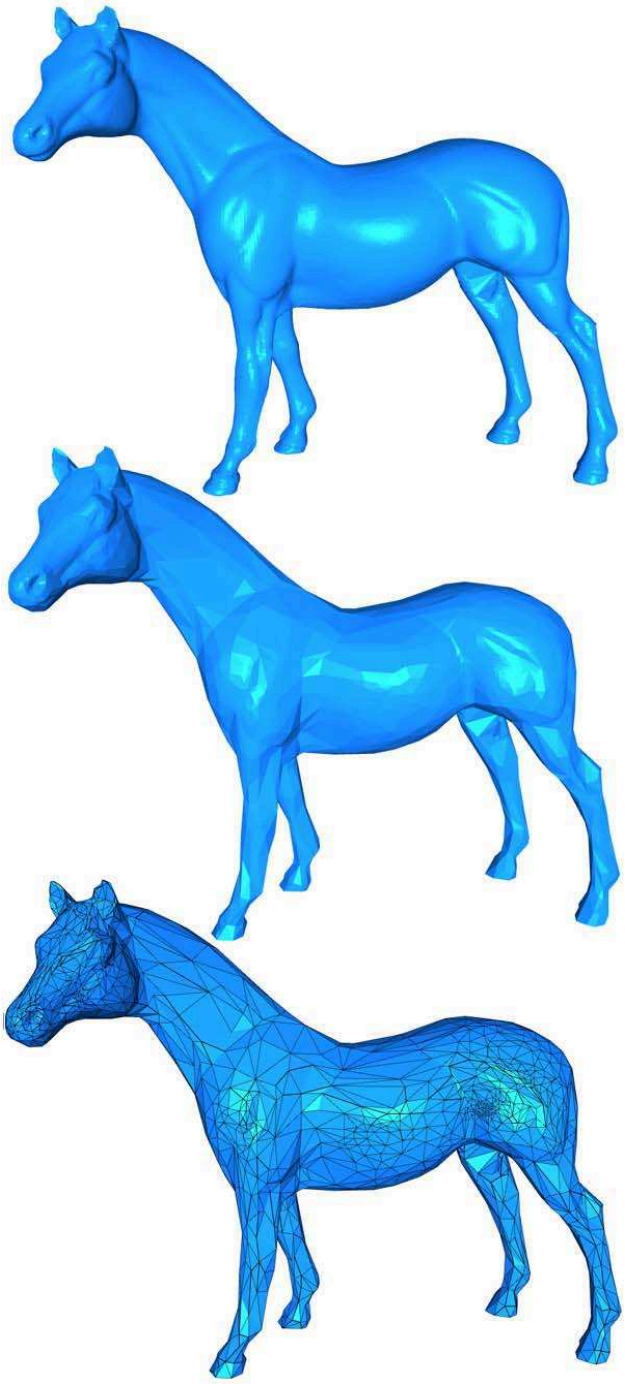

Figure 8: Original horse model (top) #faces 96966, adaptively simplified version #faces 3 070 .

complexity of the preprocessing phase is O($n$log$n$). During interactive session, the only time consuming methods are *refine_vertex*() and *simplify_vertex*() but each involves traversal of the half-edges around a vertex u and modification of next adjacent half-edge references of the half-edges along the boundary of T_u, so each takes constant time. Therefore, the complexity of refining/simplifying s vertices exploiting simplification hierarchy is O(s); but in case of Zeta, it is O($s$log$s$) because they maintain a priority queue.

7 Conclusion and Future Work

We proposed a new multiresolution model *ADSIMP* for user-controlled simplification of polygonal meshes that exploits simplification hierarchy as well as hypertriangulation model and consequently provides the functionality of both low-level and high level user-driven methods for adaptive simplification of polygonal meshes in a unified environment with reduced overhead of memory consumption and faster execution time and is quite simple to implement. The method of construction of the proposed model and, the techniques for navigation across continuous LODs, selective refinement and selective simplification have been presented.

Employing this model one can efficiently navigate through continuos levels of detail of a mesh, and can extract mesh at a constant desirable resolution and can further locally simplify or refine the selected LOD as he/she pleases.

Consistency check in selective refinement sometimes causes to refine those vertices i.e. activates the vertices which are not desired. It is one of the future work to investigate how can we overcome this issue. It seems that this model can be exploited for efficient collision detection and partitioning of polygonal models; we intend to investigate this model for these purposes.

References

[1] Cignoni, P., Montani, C., Rocchini, C., and Scopigno, R. Zeta: A resolution modeling system. GMIP: Graphical Models and Image Processing 60(5):305-329.

[2] Clark, J. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM* 19, 10, pp:547-554.

[3] Funkhouser, T., A., and Sequin, C., H. Adaptive display algorithm for interactive frame rates during visualization of complex environment. In *Proc. SIGGRAPH'93,* pp:247-254, 1993.

[4] Ciampalini, A., Cignoni, P., Montani, C., and Scopigno, R. Multiresolution decimation based on global error. *The Visual Computer*, 13:228-246, 1997.

[5] Cohen, J., Varshney, A., Manocha, D., Turk, G., Weber, H., Agarwal, P., Brooks, F., and Wright, W. Simplification envelopes. In *Proc. SIGGRAPH'96*, pages 119-128, 1996.

[6] Garland, M., and Heckbert, P., S. Surface simplification using quadric error metric. In *Proc. SIGGRAPH'97*, pages 209-216, August 1997

[7] Garland, M. Multiresolution modeling: survey & future opportunities. *Eurographics'99*- State of the Art Report, 111-131.

[8] Guéziec, A. Locally toleranced surface simplification. *IEEE Transactions on Visualization and Computer Graphics*, 5(2): 168-189, April-June 1999.

[9] Li, G., and Watson, B. Semiautomatic simplification. In *ACM symposium on Interactive 3D Graphics* 2001, 43-48.

[10] Luebke, D., and Erikson, C. View-Dependent Simplification of Arbitrary Polygonal Environments. In *Proc. SIGGRAPH '*97, pp: 199–208, 1997.

[11] Kuebke, D. A Developer's survey of Polygonal simplification Algorithms. *IEEE Computer Graphics & Applications*, 24-35.

[12] Hoppe, H. Progressive meshes. In *Proc. SIGGRAPH'96,* pages 99-108, August 1996. http:// research.microsoft.com/~hoppe/

[13] Hoppe, H. View-dependent refinement of progressive meshes. In *Proc. SIGGRAPH '*97, pp: 189–198, 1997.

[14] Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., and Stuetzle, W. Mesh optimization. In *Proc. SIGGRAPH '*93, pp: 19–26, 1993.

[15] Hussain, M., Okada, Y. and Niijima, K. Fast, simple, feature-preserving and memory efficient simplification of triangle meshes. International Journal of Image and Graphics, 3(4):1-18, 2003.

[16] Hussain, M., Okada, Y. and Niijima, K. Efficient and Feature-Preserving Triangular Mesh Decimation. Journal of WSCG, 12(1):167-174.

[17] Lindstrom, P., and Turk, G. Fast and memory efficient polygonal simplification. In *Proc. IEEE Visualization'98*, pages 279-286, Oct. 1998.

[18] Lindstrom, P., and Turk, G. Evaluation of memoryless simplification. *IEEE Transactions on Visualization and Computer Graphics*, 5(2):98-115, April-June 1999.

[19] Pojar, E., and Schmalsteig, D. User-controlled creation of multiresolution meshes. In *ACM symposium on Interactive 3D Graphics 2003, pp: 127-130, 243.*

[20] Algori, M., and Schmitt, F. Mesh simplification. *Computer Graphics Forum*, 15(3), August 1996., *Proc. Eurographics'96, 1996.*

[21] Ronfard, R. and Rossignac, J. Full range approximation of triangular polyhedra. *Computer Graphics Forum*, 15(3), 1996. *Proc. Eurographics'96, 1996.*

[22] Xia, J., C., and Varshney, A. Dynamic view-dependent simplification for polygonal models. In *Proc. Visualizaion'96*, pages 327-334, Oct. 1996.

[23] Kho, Y., Garland, M. User-guided simplification. In *ACM symposium on Interactive 3D Graphics 2003, pp: 123-126, 242.*