Simplification of Reconstructed Meshes in Real Time

F. Pizarro
Universität Karlsruhe
fpizarro@ira.uka.de
http://i33www.ira.uka.de/

S. Preuß
Universität Karlsruhe
stpreuss@ira.uka.de
http://i31www.ira.uka.de/

A. Schmitt
Universität Karlsruhe
aschmitt@ira.uka.de
http://i31www.ira.uka.de/

Abstract

Immersing a scanned replication of a person into a virtual reality environment is a multi-faceted challenge. One key issue of this task is to improve the result and performance of the reconstruction algorithm that is used to create the 3D model out of pictures taken from the person. In this paper, we present a method to reduce the effort of a 3D model reconstruction process by polygonal simplification in real time (in less than 0.06 seconds). Besides being a relative fast algorithm, in every step the method holds a consistent mesh, so it can be aborted if calculation time runs short. Furthermore, we describe a data structure called *AI-Tree* (Adjustable Index Tree), that allows to speeding up the process of rebuilding a mesh data structure out of lists of vertices, edges and polygons that appear in file formats of 3D models and network communication. *AI-Trees* offer a fast way to find indices out of linked lists, and a scaleable trade-off between performance and memory consumption.

Keywords: Polygonal Simplification, Virtual Environments, Three-Dimensional Shape Recovery, Range Data Analysis, Data Structures, Data Transmission.

1 Introduction

The main goal of this work is to immerse real people into a virtual reality environment. Contrary to other works (e.g., Fautz [1]), the number and the flexibility of the camera positions are reduced. For the development and testing of our algorithms we use the setup illustrated in the top of Figure 1. Eight cameras are distributed around the workspace in front a the projection wall to provide pictures of the user, that are used to generate a 3D model of himself.

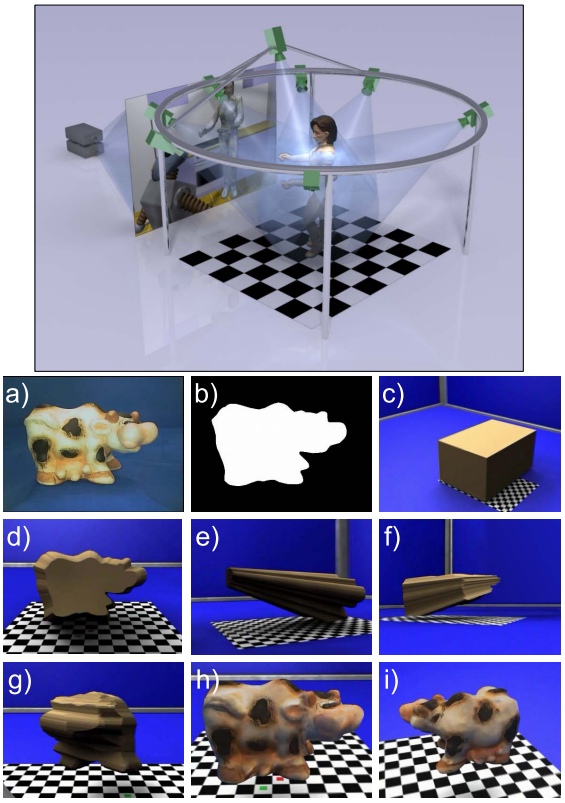

Figure 1: Illustration of our environment and reconstruction process.

The algorithm has to achieve visual realtime (currently at 15 fps, due to the synchronization mode of the used ieee1394-cameras). So the algorithm has to be optimized in account to speed,

further usability of the result and visual quality. In this paper we describe the optimization of a polygonal mesh resulting of the volume intersection algorithm. Polygonal simplification reduces the effort in the following cutting steps of the volume intersection algorithm and the effort of texturing the final visual hull (see Figure 1 c,i).

2 Related Work

[2] introduced the concept of a visual hull. Plainly speaking the visual hull of an object against a definite viewing space is the maximum body that casts the same silhouettes. Depending of the viewing space, it might be impossible to reconstruct certain concave features that lie inside the convex hull of the object. One statement of this paper is that, despite that some concave features might be skipped, the visual hull defines the best possible result for silhouette based reconstruction algorithms.

[3] propose a realtime approach to reconstruct objects (and people) which has similarities to our own. His precise connectivity method also uses the silhouettes acquired from the object's images and casts rays through the silhouette boundaries. Using epipolar geometry, the silhouettes taken from different camera positions provide information to separate these rays into intervals which belong to the object and intervals that do not. These intervals are assembled into the desired visual hull of the object.

[1] described in his PhD thesis an automatic reconstruction process of real objects. Pictures of an object are taken by a camera on a robot arm, which provides a great flexibility in the choice of the viewing points. He developped a new variant of the principle of volume intersection, using boundary representations and thus avoiding disadvantages of volumetric representations. Additionally the visual impression of the resulting geometry is improved with refined textures out of the taken images of the object. The high performance of this algorithm at that time leads to this work, in which it is tried to establish it in a real time environment.

2.1 The Reconstruction Process

The reconstruction process as illustrated in the bottom half of Figure 1 starts with the extraction of the object from the background to obtain the silhouette and an usable texture (Figure 1 a,b). The description of the adaptive background model to extract the object from the background, while ignoring changing lighting conditions, shadows and the changing projection wall is described in [4].

In the first step to generate the visual hull, the viewing cones are build with the knowledge of the camera positions, defining the tip, and the silhouettes, which define the base (Figure 1 d,e). Starting with a cube representing the workspace (Figure 1 c), each viewing cone is successively cut from the previous result to further approximate the visual hull of the object (Figure 1 f,g). This reconstruction process is the main objective for the discussed polygonal simplification algorithm of this paper.

Finally the texture coordinates are calculated by projecting the images from the camera positions onto the reconstructed object. A set of blended images are used for view dependend texturing (Figure 1 h,i). This includes the detection of self shadowed parts, that are not to be seen from the cameras and should not be textured with their pictures (e.g. the inside of the legs from the view of a side camera).

3 Motivation

The meshes from the reconstruction process bear some uneccesary characteristics like stretched faces with low curvature to their neighbouring faces and highly faceted regions, that do not contribute to the overall visual appearance.

The time comsumption of subsequent reconstruction stages (e.g. smoothing, texturing with detection of self shadowing, network communcation) after the visual hull reconstruction, depends mostly on the number of facets of the reconstructed geometry, that are reduced by a predecessing polygonal simplification stage described in the Section 4. For a dynamical trade off of processing time between polygonal simplification and the following process stages, the polygonal simplification process must be interuptable, that means it has to always provide

a consistent geometry.

The geometry is represented in the usual cross referenced data structure of vertex, edge and face objects, that provide a fast access for navigating the geometry. Additionally the three kinds of objects are also stored in lists for sequential tasks, like rendering or file handling. Whereas filling the lists out of network or file data is an easy and fast task, the reconstruction of the geometric data structure is quite time consuming, as even a single indexed access on lists has a worst case cost of $O(n)$. The presented *AI-Tree* in Section 5 helps to quicken the indexed access on the lists for rebuilding the geometric data structure.

4 Polygonal Simplification

Our polygonal simplification has a main restriction according to the main process. If we consider at least 15 fps, then the simplification process must be executed or stopped in less than 0.06 seconds. As a second condition, our simplification should be based on the data structure used in the main reconstruction process (bidirectional lists with cross references between vertices, edges and polygons). Our goal is to reduce the mesh as much as possible in this stretch of time without visual degradation of the mesh.

4.1 Previous Works

Many good algorithms have been developed for the simplifications of meshes. Although it is not possible for us to cover all the existing approaches, we have try to cover a generic group with similar characteristics.

[5, 6] and more recently [7], use *vertex-removal* as the simplification method to gradually simplify meshes. This method removes vertices and its associated geometric information (polygons and edges), generating holes in the surface that are replaced with new triangulations. The holes around the removed vertices need to be replaced by concave triangulations to avoid self-intersection of the polygons. Also, it is necessary to find a plane that fits (as much as possible) all the adjacent vertices belonging to the created hole, to evaluate the volume cost of this operation using the vertex removed as reference.

[8, 9, 10] use *vertex-clustering* to simplify the mesh by incremental steps instead of small local simplifications (*vertex-removal* and *edge-collapse*). This approach is faster than [11], but with several changes in the features of the meshes that will produce undesirable effects for our purpose (the obtained meshes in our reconstruction process are already poor in detail quality). Furthermore, this process cannot be interrupted before its natural execution to obtain acceptable results. [10] has improved the visual performance of this technique, but increasing the execution time by a factor of 7.5.

[12, 13, 14, 15, 11] use *edge-collapse* as the local method. This method introduced by [12] has been used in many applications for automatic simplification of meshes. Undoubtedly this method has demonstrated to be very effective in mesh simplification by priority queue. Briefly each edge in the mesh have a collapse priority, defined by a cost funtion [14, 15]. Although [14] is a very fast algorithm of simplification with high quality approximation, only to compute the Q matrix of each vertex plus the derive matrix Q' of each edge (wich basically represent the collapse cost) and create the priority queue, would drain our simplification time very fast. [12, 13] use a strategy too slow to be considered in our real time problem.

[11] has improved the performance of [14] about 2.5 times with a visual quality slightly inferior. This improvment is achieved avoiding the priority queue by Multiple-Choice-Techniques (MCT) and at same time using half-edge collapse instead of minimizing the geometric error of each collapse operation. Nevertheless they compute about 16 Q and 8 Q' matrix in the evaluation cost process of the 8 edges randomly selected during the MTC process. We assume that some pointer array data structure is used togheter with OpenMesh (as data structure for geometric representation) to efficiently access the edges randomly selected.

4.2 Fast Polygonal Simplification

Our approach uses *edge-collapse* as the local simplification method. Although the error metric proposed by [14] is improved in time by [11], we have developed a new error metric that con-

sumes less time and we obtain visually acceptable results. We also avoid the use of priority queue as [11], but we do not use a random selection of candidates to collapse using MTC, instead of that, we only process the vertices list verifying if each vertex can be collapse with a counterpart according to its curvature. Comparing our simplification process without the factor time our approach does not produce high quality simplification as [12, 13, 14, 15, 11], but we obtain visually acceptable results and without topological degradation of the meshes as [8, 9, 10].

We will use the notation v, e and t to represent the geometric elements; vertices, edges and triangles respectively. We have adopted the operators $\lceil\rceil$ and $\lfloor\rfloor$ defined by [15] to describe our approach.

We define our simplification method as the execution of successive *simplification phases*. The *simplification phase* is defined as the verification of each *valid vertex* v in the mesh to be collapsed with another *valid vertex*, where a *valid vertex* is any vertex that was not created by a successful edge-collapse in the current simplification phase (at the beginning of every phase each vertex is considered valid). We introduce this definition of our simplification method as formal definition, to explain our approach, however it is not our intention to present it as automatic polygonal simplification for meshes. Our goal is restricted only to be applied in our reconstructed process of meshes in real time.

4.2.1 Collapse Metric

Our approach runs in a real time environment and is based on mesh curvatures giving us the advantage to execute the simplification even during the process of reconstruction based on images, needing minimal geometric information around two vertices. Given a vertex v we choose its collapse counterpart v' by minimizing $\Omega(v, v')$. $\Omega(v, v')$ is defined as:

$$\Omega(v, v') = |\omega(v) - \omega(v')| \tag{1}$$

where:

$$\omega(v) = \sum_{\forall e \in \lceil v \rceil} \theta(e) \tag{2}$$

The operator $\theta(e)$ calculates the dihedral angle between the triangles in $\lceil e \rceil$ and $v' \in \{\lfloor\lceil v\rceil\rfloor - v\}$. Once v' is choosen, we replace the vertices v and v' by v_n in $\lceil\lfloor\lceil v\rceil \cap \lceil v'\rceil\rfloor\rceil$ and $\lceil\lceil\lfloor\lceil v\rceil \cap \lceil v'\rceil\rfloor\rceil\rceil$. v_n is defined as:

$$v_n = v + (v' - v)\left(\frac{1}{2} + \frac{\omega(v') - \omega(v)}{2 \cdot \max[\omega(v'), \omega(v)]}\right)$$

If we meet the condition $\omega(v) + \omega(v') = 0$ we choose the midpoint between v and v' for v_n.

4.2.2 Error Metric

The use of the previous described curvature metric to collapsed edges is not enough to guarantee the preservation of the original features. We must consider other restrictions to guarantee the feature preservation and also avoid the self intersection in the new triangulation generated by edge-collapse operations. These rectricions are defined as:

$$\omega(v) > \gamma \tag{3}$$

$$\Omega(v, v') > \mu \tag{4}$$

$$\omega(v_n) > \max[\omega(v), \omega(v')] \cdot \lambda \tag{5}$$

The edge-collapse operation is rejected if (3) or (4) is met and the operation is undone if (5) is met after the same one.

Restriction (3) garantees the preservation of strong change of curvature in the mesh. The values of γ can be defined reasonably for most of the meshes as $0 < \gamma \leq \pi$ (to get this range of values for γ, consider as example a corner vertex v of a perfect cube mesh in (3) and any other vertex in the same mesh).

Restriction (4) prevents to collapse vertices with angular differences bigger than μ degrees. As a result it allows to generate an approximate curvature of the original mesh. The value of μ is directly related with the final approximation of the mesh curvature that we want to obtain.

Finally, restriction (5) prevents a triangle intersection being produced by collapse operations. If triangle intersection happens, then $\omega(v_n)$ produces a considerable imbalance of (5) due to the abrupt change of curvature between the intersected triangles. We have defined $\lambda = 1.25$ empirically after many tests, to assure triangulations free of intersections.

Of course if we consider meshes with hundred of thousands or millions of polygons, to

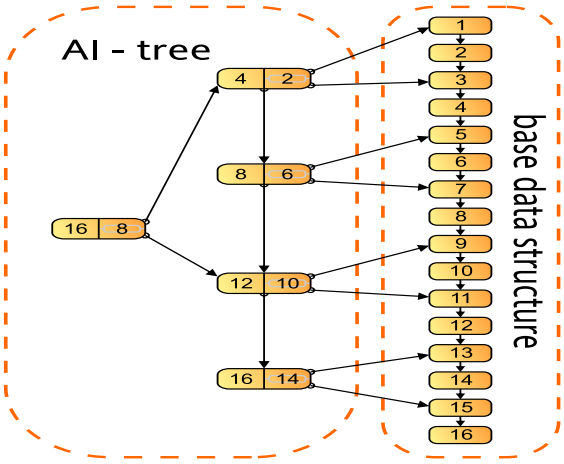

Figure 2: Structure of the *AI-Tree*.

be simplified about 0.06 seconds, then these restrictions are unnecessary, due to the high density of vertices. However with meshes between 6,000 and 8,000 polygons obtained using our reconstruction process, the limitless application of edge-collapse operations without any error metric produce severe visual changes only in 0.06 seconds.

4.2.3 Additional Considerations

The reconstruction process in real time does not create meshes with holes, only closed manifold meshes, but we developed our polygonal simplification by many meshes of public domain. Of course, to reduce our tests only to closed meshes would deprive us of interesting meshes to test. The problem resides in determining (2) for vertices that belong to the boundary of a mesh, because the edges in that area have only one associated triangle and make it impossible to calculate $\theta(e)$. To solve this problem we define $\omega'(v)$, $\Omega'(v, v')$ and $\theta'(e_1, e_2)$, These functions are defined as:

$$\Omega'(v, v') = |\omega'(v) - \omega'(v')| \qquad (6)$$

where:

$$\omega'(v) = \theta'(e_1, e_2), \forall e_1, e_2 \in \lceil v \rceil \qquad (7)$$

The operator θ' calculates the angle between the directional edges e_1 and e_2. Although we do not need this metric in our real time simplification, it allows us to examine the behavior of our simplification approach in many meshes. We consider

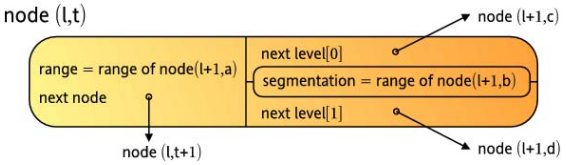

Figure 3: AI-Tree Node

also a similar restriction such as (3) for (7) by $\omega'(v) > \gamma'$ and (4) for (6) by $\Omega'(v, v') > \mu'$ (notice that a similar restriction such as (5) is not necessary with (7) because the edge-collapse operation with boundary vertices does not generate self intersections).

5 AI-Tree

The reconstruction of the data structure configuration by meshes stored in files or received by network is an inefficient process because lists (used in our reconstruction process) has a worst case of $O(n)$ in the search operation. As a consequence to create the cross references between vertices, polygons and edges becomes a process quite inefficient for meshes such as the Stanford Bunny.

In order to solve this problem we designed a data structure called *AI-Tree* (Adjustable Index Tree) that allows us to rebuild the original mesh structure quickly and efficiently (less than 1 second for meshes with 100,000 polygons). The *AI-Tree* is attached to another base data structure to improve the execution costs to $O(\log_2(n))$ for search operations (see Figure 2). Additional minimal memory is used to improve the search and it is possible to obtain a trade-off between performance and memory consumption reducing the number of levels in the *AI-Tree*.

5.1 Implementation

We will consider the base data structure attached to the *AI-Tree* as the last level.

1. The *AI-Tree* has a total of *L* levels for *n* elements. *L* is defined as:

$$2 \leq L \leq \lceil log_4(n) \rceil \qquad (8)$$

2. Each level L_i will have T_i elements. T_i is

Number of operations in the Search Operation AI-Tree (worst case)					
levels v/s n	5,000	35,000	100,000	250,000	500,000
2	72	188	317	500	708
3	27	51	71	96	120
4	19	28	36	47	56
5	15	24	25	33	35
6	17	18	24	24	30
7	13	20	21	21	27
8		16	23	24	24
9			17	18	26
10					19
Number of operations in the search operation AVL-Tree (worst case)					
AVL-Tree	13	16	17	18	19

Table 1: Execution cost for the *AI-Tree* and AVL-Tree.

defined as:

$$\underset{0\leq i\leq L}{T_i} = \begin{cases} \frac{\lceil T_{i+1}\rceil}{4} & \text{if and} \quad L=\lceil log_4(n)\rceil \\ & \quad i\neq L \\ n^{\frac{i}{L}} & \text{else} \end{cases} \tag{9}$$

3. The execution cost in the worst case for the search operation is defined by *C* as:

$$C = L + \sum_{i=1}^{L} c_i \tag{10}$$

where:

$$c_i = \begin{cases} 0 & \text{if} \quad T_i - 2\cdot T_{i-1} \leq 0 \\ \left\lceil \frac{T_i}{2\cdot T_{i-1}} - 1 \right\rceil & \text{else} \end{cases} \tag{11}$$

We will use the expression *node*(l,t) to refer to a specific node in the *AI-Tree*, where l is the level number $(0 \leq l \leq L)$ and t the position in the level $(1 \leq t \leq T_i)$. If $l = L$, the expression *node*(l,t) will be an element in our base data structure, but we consider only as part of the *AI-Tree* for this analysis. Figure 3 shows the visual appearance of the node with its variables. Such a node consists of three pointers and two integer variables. So a node occupies 20 byte of memory on a 32-bit architecture. The variables for the *AI-Tree* node represent respectively:

Range: represents the data found in the last level (the base data structure) with *id* smaller than or equal than our *id* of search. If we meet the previous condition we use the references *NextNode[0]* and *NextNode[1]* according to the variable *segmentation*. If we do not meet the previous condition then we navigate to the next node in the same level (*NextNode*).

segmentation: represents the data found in the last level through the references to the next level. If the current *segmentation* corresponds to an *id* smaller than or equal to our *id* of search, then we navigate to the next level through the first reference (*NextLevel[0]*). If we do not meet the previous condition we navigate to the next level using the second reference (*NextLevel[1]*).

NextLevel[0], *NextLevel[1]*: represent the navigation references from one node to another node in the next level of the *AI-Tree*.

NextNode: represents the navigation reference from one node to the next node in the same level of the *AI-Tree*. This navigation in the same level that is not used in normal trees, allows us to modify the total nummber of levels in the *AI-Tree* according to (8) and to produce a optimal trade-off between memory consumption and performance for long amoung of data.

We will define formally every variable in each *node*(l,t) according to Figure 3 as:

$$\begin{array}{lllllll} a & = & 2tD_l & & c & = & (2t-2)D_l+1 \\ b & = & (2t-1)D_l & & d & = & (2t-1)D_l+1 \end{array}$$

where:

$$D_l = \frac{T_{l+1}}{2T_l}$$

Note that we always meet the condition $c \leq b \leq d \leq a$. Through these variables it is easy to create the *AI-Tree* in a descending cascade beginning from the last level. Let X be a single node *node*(l,t) with $l < L$. X needs to complete all its references the creation of *node*$(l+1,a)$, where a is the variable of the current X node. Of course we can reverse the problem so that the next level $(l+1)$ would send a signal to the level of X at the moment of the creation of *node(l+1,a)*. Once the reference between the

Trade-off with memory cost in MB.					
levels v/s n	5,000	35,000	100,000	250,000	500,000
2	0.001	0.004	0.006	0.010	0.014
3	0.006	0.022	0.044	0.081	0.128
4	0.014	0.055	0.199	0.234	0.391
5	0.022	0.099	0.222	0.454	0.781
6	0.032	0.148	0.344	0.721	1.264
7	0.033	0.202	0.479	1.020	1.812
8		0.233	0.622	1.341	2.406
9			0.667	1.667	3.033
10					3.333

Table 2: Temporal memory cost *AI-Tree*.

Data Structure Reconstruction			
n	B.List	AVL-Tree	AI-Tree
5,000	300,030,016	816,495	770,595
35,000	14,700,210,176	6,992,770	6,573,261
100,000	120,000,602,112	21,948,272	20,598,234
250,000	750,001,520,640	59,166,936	55,461,368
500,000	300,000,293,6832	124,833,856	116,922,744

Table 3: Operation cost for generating the cross references between vertices, edges and polygons.

Polygonal Simplification			
n	B.List	AVL-Tree	AI-Tree
5,000	15,003	42,126	15,003
35,000	105,003	343,933	105,003
100,000	300,003	1,058,365	300,003
250,000	750,003	2,811,128	750,003
500,000	1,500,003	5,872,239	1,500,003

Table 4: The *AI-Tree* keep intact the performance of the real time reconstruction environment.

node X and $node(l+1, a)$ is created, new signals from the level $l+1$ will be sent to the succeeding node of X in the same level. This parallel signaling system between levels according to the variables a, b, c and d facilitates the creation of the *AI-Tree* simultaneously with the creation of the bidirectional list.

5.2 Performance

The *AI-Tree* uses half of the levels of a binary tree but maintains its performance in the search operation. We can prove this hypothesis easily if we use a continuous function of (10). This continuous function can be expressed as:

$$C' = L\frac{n^{\frac{1}{L}}}{2} \tag{12}$$

Notice that (10) is a discreet function and (12) is equivalent to (10) as a continuous function. The second expression in (12) represents the worst navigation case in the same level before navigate at the following level in the search operation.

According to our hypothesis we can affirm the following statement:

> *Let A be an AVL-Tree and B a AI-Tree. A and B have n elements each one, then B uses half the levels of A to have the same performance than A does in the search operation.*

Using (12) and our previous statement, we can easily prove our hypothesis[1]:

$$\log_2(n) \cdot \frac{n^{\frac{1}{\log_2(n)}}}{2} = \log_4(n) \cdot \frac{n^{\frac{1}{\log_4(n)}}}{2} \tag{13}$$

Table 1 shows also the performance of the *AI-Tree* using different levels and we observe that our hypothesis is correct according to (13). Table 2 shows the trade-off between memory and levels.

5.3 Creation Cost

As was mentioned, the *AI-Tree* is created simultaneously with the creation of the bidirectional list. Although the insertion of elements into the list has cost *O(1)*, we must consider the creation cost of the *AI-Tree*. The worst case creation for the *AI-Tree* happens when it has the maximum possible levels. Using (8) and (9) we can create the next recurrent equation:

$$T(n) = \begin{cases} 1 & \text{if} \quad n \leq 4 \\ \frac{n}{4} + T\left(\frac{n}{4}\right) & \text{else} \end{cases}$$

Solving this equation we obtain $T(n) = \frac{n-1}{3}$.

5.4 Collective Performance

We can make an analysis considering the global simplification process in two parts; *data structure reconstruction* and *polygonal simplification*. These two processes can be evaluated in function of the operations; *insert (I)*, *search (S)* and *delete (D)* respectively creating general cost functions of the processes.

The data structure reconstruction implies a continuous insertion in the bidirectional list for n vertices, $2n$ triangles and $3n$ edges[2]. Also this reconstruction needs the creation of the cross references through search operations for $12n$ vertices by triangles and edges[3]. It is defined

[1] The left side of (13) uses $L = \log_2(n)$ according to the normal number of levels in a AVL-Tree for n data.

[2] For the relationship between number of vertices, edges and triangles, see the Euler Formula for meshes.

[3] Note that the $12n$ search operation correspond to 3 references to vertices by $2n$ triangles and 2 references by $3n$ edges.

the execution cost of these processes as:

$$\sum_{i=1}^{n} I(i) + \sum_{i=1}^{2n} I(i) + \sum_{i=1}^{3n} I(i) + \sum_{i=1}^{6n} S(n) + \sum_{i=1}^{6n} S(n) + \underbrace{2 \cdot \left(\frac{n-1}{3}\right)}_{\text{only AI-Tree}} \quad (14)$$

The polygonal simplification implies a deletion of vertices, triangles and edges. We can assume for this analysis a polygonal simplification at less than 50%. The execution cost of this process is defined as:

$$\sum_{i=\frac{n}{2}}^{n} D(i) + \sum_{i=n}^{2n} D(i) + \sum_{i=\frac{3n}{2}}^{3n} D(i) \quad (15)$$

Through these functions we can compare the performance of several alternatives. Our natural intention is to use the original bidirectional list of our simplification methods but the search operation between vertices, edges and polygons its excessively costly. Considering that our simplification process deletes information dynamically, we could consider the AVL-Tree as an alternative data structure. This solution will improve the time problem in the cross references reconstruction process, nevertheless, on the other hand this change will modify the scenario of our real time problem and also will increase the number of operations for the simplification process. The *AI-Tree* fits in our real time scenario perfectly as Tables 3 and 4 show. We included the creation and the elimination of the *AI-Tree* in our analyses. Of course this expression in (14) is not considered in the calculations with the bidirectional list and the AVL-Tree.

The *AI-Tree* not only improves the number of operations in the reconstruction process in comparison to the AVL-Tree, it also does not produce alterations to prove our simplification according to our real reconstruction process. The *AI-Tree* has better performance than any other dynamic structure for the reconstruction process and to execute the simplification process to our knowledge (in the simplification process the bidirectional list offers the best performance to eliminate dynamically data, given its cost $O(1)$ by cross references between geometric information). We have not considered other aspects in our previous analysis that affect the time process as recursive implementations of the AVL-Tree, as example, the reorganization to remain as AVL-Tree and the navigation for rendering. On the other hand the *AI-Tree* and bidirectional lists do not need recursive implementations and stack memory for keeping its performance and navigation operation making it faster.

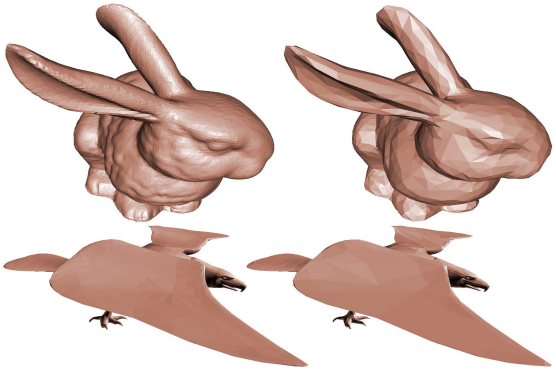

Figure 4: Examples of our simplification approach. The Stanford Bunny and Eagle mesh (on the left) were simplified from 69,451 to 3,756 polyons and from 33,072 to 2,815 polygons respectively.

6 Results

We have tested our simplification approach under a wide range of meshes. In all our tests we keep the features of the original meshes after thousands of applied collapse operations, guaranteeing the silhouette conservation in the first 0.06 seconds of simplification. In all our tests we have used the same parameters to assure a robust algorithm ($\gamma = \frac{3\pi}{4}$ and $\mu = \frac{\pi}{9}$ respectively).

Figure 4 shows examples of our proposed simplification without considering the factor time, after 7 optimization phases in less than 2.6 seconds. It is possible to see a loss of detail after high simplification of the meshes, nevertheless all the features have been kept intact. Notice that drastic changes of curvature are kept intact and these are a principal characteristic of the meshes created under the process of reconstruction in real time (see Figure 6).

Figure 5 shows examples of our proposed er-

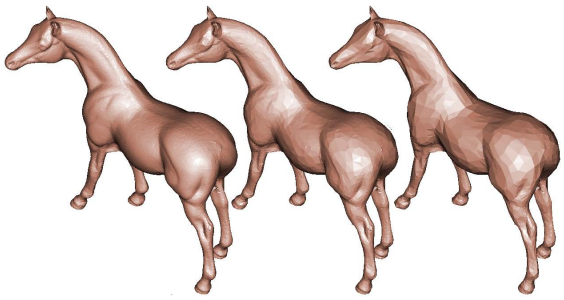

Figure 5: Left: original horse model (39,698 polygons). Middle and Right: examples of our proposed error metric. The resultant meshes show different approximations of the original mesh curvature using $\mu = \frac{\pi}{18}$ and $\mu = \frac{5\pi}{36}$ respectively.

ror metric after 7 simplification phases. The error metric allows to adjust the simplification according to the output quality of our reconstruction process. At the moment we do not use high restrictions for our error metric due to the low detail of the meshes generated in the reconstruction process, but according to its improvement we have the posibility to adjust our simplification process.

Figure 6 show meshes created with our reconstruction process and simplified in less than 0.06 seconds[4]. The simplifications of the same ones reduces the time spend for the texturing process and network communication in the same range ($\approx 33\%$).

7 Future Work

The polygonal simplification exposed here is our first approach to simplify meshes in real time, developed with meshes of public domain, however we will concentrate on optimizing the polygonal simplification based on meshes with specific features generated under our real time process (such as Figure 6). These meshes bear common features of small elongenated triangles, that are not present in otherwise typical meshes.

Although our error metric is less time consuming than the error metric proposed by [14], the restriction (5) is responsible for undoing a

[4]These results were obtain on a machine AMD Athlon 1.8GHz with 384MB memory.

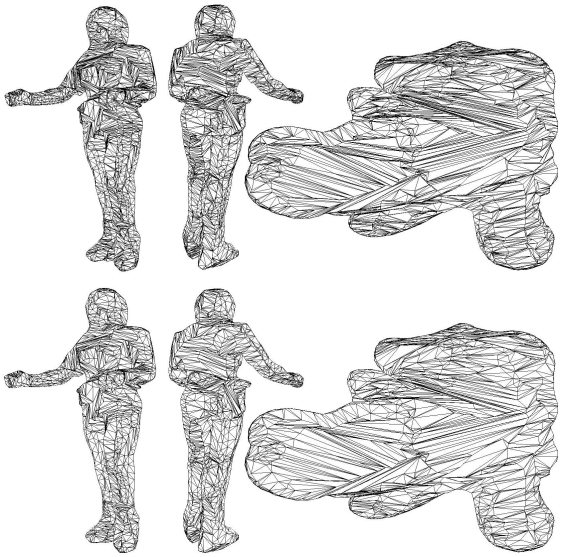

Figure 6: Examples of our simplification in real time. Top: the result Aki and Cow meshes obtained under the reconstruction process. Bottom: Aki mesh simplified from 8,085 to 5,281 polygons and Cow mesh from 8,020 to 5,436 polygons.

considerable amount of collapse operations. For example, in the simplification process executed in Figure 6 were more than 1,000 collapse operations undone in each mesh, and it is not only a non negligible lost quantity of simplification, but also lost time. This problem is produced by our edge collapse approach, due to the generation of polygonal intersection. Avoid this restriction by trying other collapse operations, that produce less possibilities of polygonal intersection.

The reduction of data to proccess due to our simplification give us the oportunity to work on our reconstruction process. At the moment the simplification is used after the reconstruction process but we are working on integrating it in the reconstruction process. The algorithms use only local geometric information and the simplification of intermediate meshes of the reconstruction process will speed up the cutting of the subsequent viewing cones.

Another use of the simplification algorithm will be to generate different level-of-detail sets of a scanned moving person, that could be used in real time graphics.

Further parallelization of our VR-environ-

ment and addition of further modules into it, will lead to a stronger use and refinement of the AI data structure for the exchange of meshes.

Acknowledgements

The first author was supported partially by its university of pre-grade *Universidad Católica del Norte* (UCN), Chile and by the *Deutscher Akademischer Auslands Dienst* (DAAD). Also we would like to thank everyone who helped to review drafts of this paper; Claudio Meneses, Raphael Straub and Dieter Finkenzeller.

References

[1] Michael Fautz. *Objekt- und Texturrekonstruktion mit einer robotergeführten Kamera*. PhD thesis, Universität Karlsruhe (TH), 2002.

[2] Aldo Laurentini. The visual hull concept for silhouette-based image understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1994.

[3] Jean Sébastien Franco, Clément Ménier, Edmond Boyer, and Bruno Raffin. A distributed approach for real time 3d modeling. *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2004.

[4] Sven Thüring, Jörn Herwig, and Alfred Schmitt. Silhouette-based motion capture for interactive VR-systems including a rear projection screen. In *CASA 2005 Proceedings*, 2005.

[5] Michel van Klink and Michael S. Lew. Decimation of visible surfaces. *Pattern Recognition, Fourteenth International Conference*, 1998.

[6] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of triangle meshes. *SIGGRAPH 92*, 1992.

[7] Pierre Alliez and Mathieu Desbrun. Progressive compression for lossless transmission of triangle meshes. *SIGGRAPH 01*, 2001.

[8] Chan K.F., Wong Y.T., and Kok C.W. Multiresolution mesh representation using vertex cluster contraction. *IEEE International Symposium*, 2001.

[9] Remi Ronfard and Jarek Rossignac. Full-range approximation of triangulated polyhedra. *Computer Graphics Forum*, 1996.

[10] Dmitry Brodsky, BenjaminWatson, I. Scott MacKenzie (editor), and James Stewart (editor). *Model simplification through refinement*. Morgan Kaufmann Publishers, Montreal, Canada, 2000.

[11] Jianhua Wu and Leift Kobbelt. Fast mesh decimation by multiple-choice techniques. *Vision, Modeling, Visualization 2002 Proceedings*, 2002.

[12] Hugues Hoppe, Tony DeRose, Tom Duchamp, John Mc-Donald, and Werner Stuetzle. Mesh optimization. *SIGGRAPH 93*, 1993.

[13] Hugues Hoppe. Progressive meshes. *SIGGRAPH 96*, 1996.

[14] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. *SIGGRAPH 97*, 1997.

[15] P. Lindstrom and G. Turk. Fast and memory efficient polygonal simplification. *IEEE Visualization'98*, 1998.