Intersection Free Simplification

Stefan Gumhold

Wilhelm Schickard Institute for Computer Science
Graphical-Interactive Systems, Sand 14
72076 Tübingen, Germany
gumhold@gris.uni-tuebingen.de
http://www.gris.uni-tuebingen.de

Pavel Borodin

Computer Graphics Group, University of Bonn, Römerstraße 164
53117 Bonn, Germany
borodin@cs.uni-bonn.de
http://cg.cs.uni-bonn.de

Reinhard Klein

Computer Graphics Group, University of Bonn, Römerstraße 164
53117 Bonn, Germany
rk@cs.uni-bonn.de
http://cg.cs.uni-bonn.de

Triangle mesh decimation and multi-resolution techniques are widely used in visualization applications for huge scenes. A large collection of different simplification algorithms exists in order to build a multi-resolution model from a given triangle mesh. All of the existing approaches focus on the creation of a geometrically close approximation of the original model. In order to produce a simplified version of a model with close layers – such as dressed humans – self-intersections result in intolerable results. Even methods that allow the sewing of close surface parts lead to unpleasant self-intersections. Only the simplification envelops[4] allow to completely prevent them.

In this work we focus on the prevention *and* avoidance of self-intersection during simplication with vertex pair contractions. We examine the geomorph[9] of the parametrized vertex pair contraction and detect collisions of the affected simplices. If no collision arises the operation cannot cause any new self-intersection. Otherwise we can simply discard the operation to prevent self-intersections as is done in the approach of simplification envelops. Our approach goes even further and tries to avoid the self-intersection by testing different target locations. This leads to better approximations as exhibited by a lower RMS and Hausdorff-distance. Furthermore our approach allows for arbitrary changes in the topology and garantees that geomorphs during progressive reception cannot cause self-intersections.

Keywords: intersection-free simplification; pair contraction; collision detection.

1. Introduction

Because of their wide support by graphics accelerators, polygonal meshes have become the most commonly used surface representation in computer graphics. New

acquisition techniques allow for the generation of highly detailed objects with permanent increasing polygon count. The handling of huge scenes composed of these high-resolution models rapidly approaches the computational capabilities of any graphics accelerator. Level of detail techniques become inevitable. In order to build such a level of detail representation a large collection of simplification algorithms exist[16,15,8,2,4,14,10,9,3,13,6,12,11,7] that produce high quality approximations of complex models with a reasonable amount of polygons. However, none of the known techniques is trimmed on the avoidance of self-intersections during the simplification process.

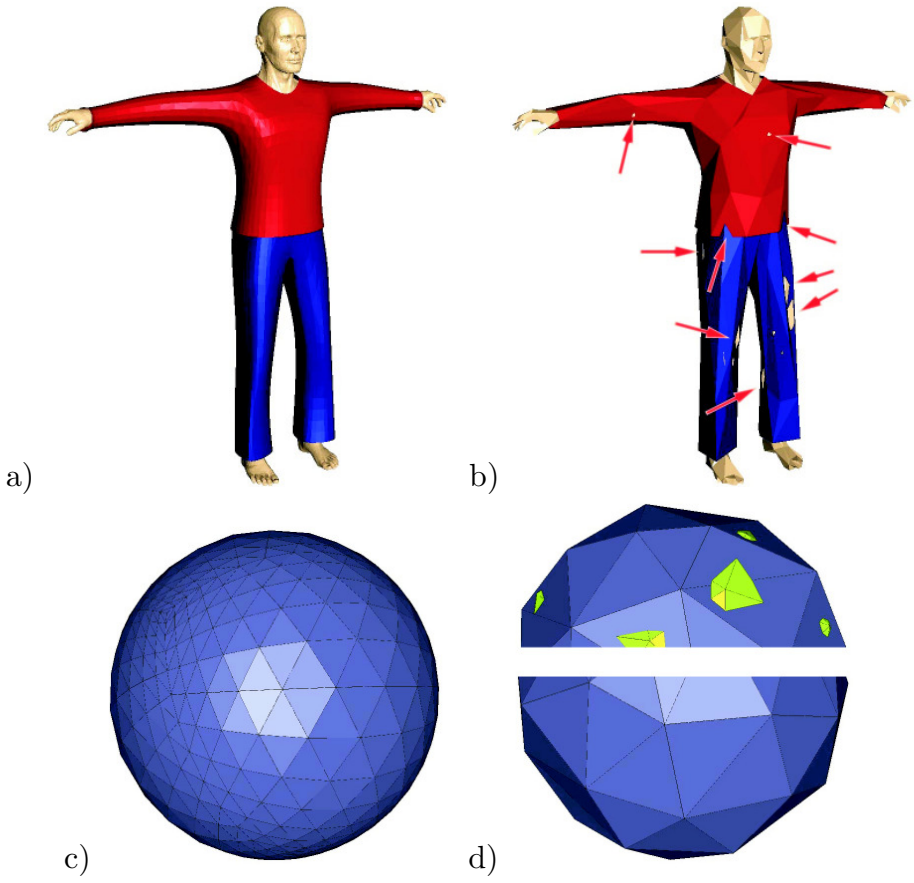

Fig. 1. a) man with shirt and trousers, b) reduced to 600 vertices, arrows show self-intersections c) green sphere in blue sphere d) reduced to 70 vertices, top without, bottom with with collision avoidance.

Figure 1 shows typical examples where self-intersections lead to severe visual artefacts. A human body is dressed with a shirt and trousers. A realistic adaption of the clothes to the body typically demands for a computationally expensive physical simulation with a not too coarse resolution of the model. For an efficient visualization the resolution of the model should be adaptable to the extent of the model in screen space, what can be achieved by the use of LODs. A standard simplification algorithm generates coarse approximations of the model with artifacts

as shown in Figure 1 b), where the combined model was reduced to 600 vertices. The body penetrates the cloth in several places. A simulation of the cloth in such a coarse resolution is not feasible and therefore is a simplification algorithm that avoids self-intersections mandatory.

The second purely synthetic model is composed of two nested spheres and was reduced to 70 vertices without (d top) and with (d bottom) avoidance of self-intersections. Self-intersection problems typically arise when two or more close surface layers are present, what is very common for models of dressed humans, for which our method was originally designed. Figure 5 a) illustrates the self-intersection problem in two dimensions for an edge collapse operation[14]. It is clear that similar cases arise for all types of simplification primitives such as vertex removal[16], triangle collapse[8], pair contractions[13,6] and also in vertex clustering approaches[15]. We simplified the man and the nested spheres once with edge collapse only and once with pair contractions, where the contracted vertices do not have to be adjacent. Both cases yielded results similar to Figure 1 b) and d).

In the typical greedy approach to simplification all possible atomic simplification operations are entered into a priority queue[10] sorted by the resulting error. We used the quadric error metrics[6] to measure the approximation error. In the main loop of the simplification algorithm the operation causing the smallest approximation error is extracted from the front of the queue and performed on the current approximation until a given approximation error or the target number of vertices is reached. The simplest approach to ensure that no self-intersections arise during such a simplification process is to check every time an atomic operation has been performed, whether a self-intersection appeared and if so discard the operation. We will refer to this approach by *prevention* of self-intersections. A more sophisticated approach for a simplification process based on pair contractions is the *avoidance* of self-intersections, i.e. when a self-intersection has been detected for a given vertex pair contraction, we try to modified the atomic operation by changing the target location such that this operation does not cause a self-intersection anymore. As it is not always possible to find a target location that does not cause a self-intersection, we can only try to avoid it. If the avoidance succeeded we cannot directly perform the modified operation as the caused approximation error has changed. Instead we recomputed the approximation error and re-insert the modified operation into the priority queue.

In both approaches of prevention and avoidance atomic operations can be discarded as they are not allowed at the current state of the simplification process. Commonly used are two other validity checks that can cause operations to be discarded. The first one checks, if the topology is preserved by the operation and is only used in topology preserving simplification. The other one checks if the normals of the affected triangles do not change more than a user specified angle[14], in order to prevent triangle flips and local self-intersections. If a selected atomic simplification operation is invalid according to the topology, normal-flip or self-intersection test it

could become valid later on in the simplification process because of some change in the neighborhood. The once discarded operations will most probably produce the smallest approximation error, when they become valid again, and should therefore not be discarded forever. The re-insertion into the priority queue of the operations, whose approximation error changes because of the currently performed operation, also re-inserts discarded operations, that failed the topology or normal-flip tests. In the case of the self-intersection tests the re-insertion mechanism does not ensure that all discarded operations will be re-inserted as the self-intersections are normally caused by simplices that are topologically far away along the surface. Therefore, we remember the discarded operations in a FiFo. In order to reconsider discard operations from time to time, we keep a counter with the number of operations to be considered from the priority queue per operation considered from the FiFo. The counter is decremented after each operation considered from the queue. If it becomes zero we reconsider one operation from the FiFo and reset the counter. It is always set to twice the number of operations in the priority queue over the number of operations in the FiFo. We limit the possible initialization values for the counter to the interval $[5, 20]$ in order to avoid an excessive reconsidering of discarded operations and to avoid that discarded operations are reconsidered not frequently enough.

The proposed self-intersection test is based on the parameterization of the pair contraction operation over time into a so called *geomorph*[9]. For each moving simplex, i.e. vertex, edge or triangle, we check if it causes a collision with another stationary or moving simplex. In this way the problem of inside-outside switches as illustrated in Figure 5 b) is also avoided. Furthermore, if the simplification process is encoded in reverse order as a progressive representation, no geomorph between two successive operations can cause a self-intersection. To avoid unnecessary collision tests, we use a spatial search data structure as presented in section 3. The strategy based on collision tests only works, if the input model does not contain self-intersections in the first place. Therefore, we first eliminate all self-intersections as described in section 4. In section 5 we describe the necessary collision tests in more detail. The avoidance of self-intersections is discussed in section 6 before the results sections 7 and the conclusion. The main advantages of the proposed method are:

- prevention of self-intersection with more flexibility as previous approaches,
- first approach with avoidance of self-intersections,
- support of non-manifold meshes and changes in the topology
- self-intersection free progressive transmission with geomorphs

2. Related Work

As most of the models used with current simplification algorithms do not contain close layers of different color, few works considered the problem of self-intersections at all. Typically, only the normal-flip test as proposed by Ronfard and Rossignac[14]

is performed, which allows to prevent local self-intersections only. As motivated in the introduction this does not allow to produce acceptable simplifications of the very important class of meshes that describe dressed people.

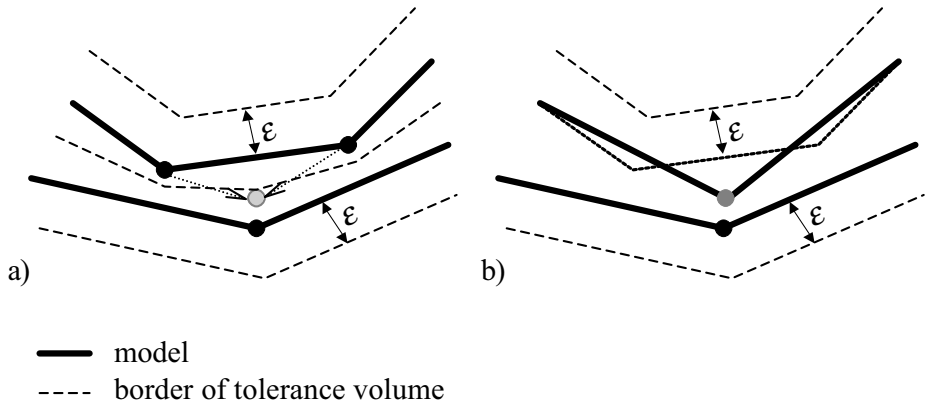

Fig. 2. Disadvantage of tolerance volume based approaches: a) the space between the close layers of the model has to be split into an upper and a lower part. The optimal target location of the edge collapse in the upper layer is not within the assigned part of the split space, b) although it causes no self-intersection.

A method that can at least prevent global self-intersections is simplification envelops[4]. In order to ensure a one-sided Hausdorff distance of ϵ from the simplified model to the original model, two offset surfaces of $\pm\epsilon$ around the original model are defined. To prevent self-intersections of the offset surfaces a different ϵ_i is specified for each vertex and is reduced from the user specified epsilon, until no self-intersections of the so called *simplification envelopes* remain. During simplification the prevention strategy as described in the introduction is performed to prevent on the one hand self-intersections and on the other hand an increase in the one-sided Hausdorff distance over ϵ by enforcing all triangles to stay within the tolerance volume enclosed by the envelopes. Simplification envelopes are not efficient in a setting where ϵ changes. Zelinka and Garland[19] discretize the tolerance volume around the original mesh on a regular grid to accelerate the validity tests for the atomic simplification operations. An increase in ϵ is achieved by growing the discretized tolerance volume by one cell in all directions. They do not check self-intersections of the tolerance volume and therefore do not prevent global self-intersections. Guéziec[7] dynamically keeps a tolerance volume around the simplified mesh such that the one-sided Hausdorff distance from the original model to the simplified mesh can be controlled. In this approach global self-intersections are neither avoided. Both the approaches of Guéziec and Zelinka/Garland could probably be generalized to avoid self-intersections. But all tolerance volume based approaches to the prevention of self-intersections have the disadvantage shown in Figure 2. The space between two close surface layers has to be split into two parts, one for each layer. This split is done in the initialization stage without knowing potential

self-intersection problems. The optimal target location of an edge collapse in the upper layer could then fall out of the corresponding tolerance volume (dashed circle in Figure 2 a), although no self-intersection is caused as can be seen in b). Thus the tolerance volume based approaches can discard the operation with the smallest approximation error even if it does not cause a self-intersection. Our approach on the other hand does not report any not existent self-intersections.

The simplification algorithm creating progressive tetrahedralizations as described by Staadt et. al[17] avoids self-intersections of the boundary surface of the tetrahedral mesh. They argue that in the case of volume boundary meshes these self-intersections can only arise at sharp boundary edges and restrict their intersection tests to these edges. Their assumption does actually not even hold for volume boundary meshes as a simple example shows: take a solid cube and cut out a very fine spherical layer. During the simplification the two sides of the layer can easily intersect without the vicinity of any sharp edges. In this case a method like ours is necessary.

3. Dynamic Spatial Search Data Structure

In three places of our intersection free simplification strategy we make use of a spatial search data structure.

(1) To remove initial self-intersections, all pairs of an edge and a triangle that intersect need to be found,
(2) to pair close vertices for vertex pairs not connected by an edge, the closest non adjacent vertex of another vertex has to be found and
(3) for collision detection during vertex contraction all simplices in the vicinity of the contracted vertices need to be known.

As simplices change their shape and location during simplification, the spatial search data structure needs to be dynamic. The interface of the data structure must support insertion of simplices – i.e. vertices, edges and triangles –, query for all triangles intersecting an edge, nearest neighbor queries from vertex to vertex and enumeration of all simplices in a given box.

We chose to use a regular grid for the spatial search because it performs well in practice in static and dynamic environments[18] and is easy to implement. In future work we will investigate, if an octree based spatial data structure performs even better. We store in each cell of the regular a list of the simplices partially or completely contained in the cell. In the beginning of the simplification process we used a grid with uniform edge length of twice the average edge length of the simplicial complex, such that each simplex was in average contained in one or two cells allowing for fast insertion and removal. During the simplification process we kept track of the increasing average edge length. When the latter exceeded the grid edge length, we re-created the grid with grid edge length twice as large and re-inserting all remaining simplices. For the used models we observed, that in average

each simplex had to be entered in a constant number of grid cells only.

3.1. *Simplex Insertion and Removal*

To insert a vertex we simply computed the enclosing cell and added it to its list of contained simplices. Edges and triangles can penetrate more than one cell into which they had to be entered. To find the penetrated cells of an edge or triangle we first tried a rasterization approach but it turned out that a conservative strategy where we entered an edge or triangle in all the cells, which were intersected by the bounding box of the simplex, performed even faster. To move simplices affected by the contraction operations, we just removed them from the grid and re-inserted them.

3.2. *Spatial Queries*

To find all triangles intersecting an edge we determined all cells that partially contained the edge and enumerated all triangles partially contained in these cells. To sort out the non-intersecting triangles we actually performed the intersection tests between the edge and the selected triangles.

The closest non-incident vertex to a given vertex v can be found in the grid by a region growing strategy stating with the cell containing the vertex. While the so far closest vertex is further away from v as the closest not considered cell, we also consider the vertices in the closest cell.

Finally, to enumerate all simplices in a given box, we simply determine all cells intersecting the box and enumerate all partially contained simplices.

3.3. *Storage Space Consumption*

The spatial grid data structure is the only additional data structure necessary for our self-intersection free simplification approach. We used a simple hashing strategy to avoid that the storage space consumed by the grid had a worse asymptotic behavior as the rest of the consumed storage space. The number of buckets in the hash map was set to the number of grid cells touching the surface of the bounding box. As hash key we simply transformed the 3d integer grid position into a single index running from zero to the total number of grid cells. The key was clamped to the number of buckets by the modulo function. Non of the buckets ever grew over a size of 8 represented grid cells. With the simple hashing approach the grid consumed between 4 and 40 bytes per vertex depending on the used model. For the grid entries additional storage space was necessary. Each simplex was entered in average in $2-3$ grid cells demanding for $2-3$ entries with a size of 8 bytes each. As there are approximately six times as many simplices as vertices, this resulted in another $100-150$ bytes per vertex. On the other hand consumed the non-manifold representation of the mesh itself with the quadric error metrics as vertex attributes more than 130 bytes per vertex for connectivity information and 40

bytes for geometry. Adding face normals and the priority queue another 60 bytes per vertex were necessary. Thus our approach consumes only about twice as much storage space as a standard pair contraction simplifier.

4. Initial Clean-Up of Self-Intersections

In this section we describe how to eliminate self-intersections in the input mesh by generation of new vertices, edges and triangles, what generates non-manifold spots at the location of self-intersections. We did not try to undo the self-intersections by moving vertices as this can be arbitrarily complicated in general. In future work we want to investigate this difficult problem in more detail.

The crucial self-intersections are the ones between edges and triangles. The vertex-edge, vertex-triangle and edge-edge intersections can be eliminated in one step and all triangle-triangle intersections coincide with two edge-triangle intersections. Our first solution tried to remove the edge-triangle intersections iteratively but did not terminate in all cases.

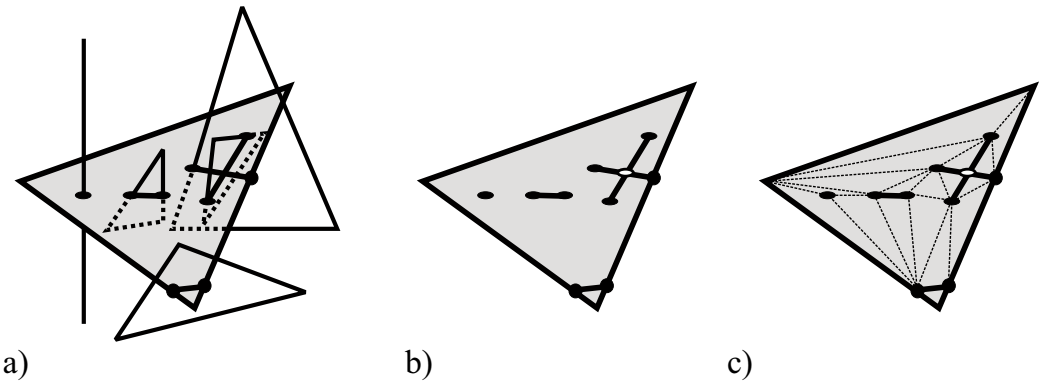

Fig. 3. a) intersections of triangle with other simplices b) creating of further vertices on intersecting segments c) triangulation of planar straight line graph.

Therefore, we came up with a method, that minimizes the number of newly generated simplices. The basic idea is to determine on each triangle the collection of points and segments resulting from intersections with other simplices as depicted in Figure 3 a). The resulting intersection graph is triangulated in a way to include the intersection points and segments. Suppose we want to split triangle t. The intersection with another triangle is a line segment and the intersection with an edge or a vertex is just a point. The resulting intersection graph can still have crossing edges, which are removed by insertion of additional nodes at the segment intersections as shown in Figure 3 b). Finally, the resulting planar straight line graph is triangulated with a standard sweep line algorithm as described for example in[5]. Nodes that are closer than a user defined ϵ are snapped together in order to avoid numerical problems with very short triangle edges. Care has to be taken in order to match identical newly introduced vertices on different triangles.

5. Detecting and Prevention of Self-Intersections

This section describes the detection of self-intersections caused during the simplification process. In a vertex pair contraction operation the two contraction vertices

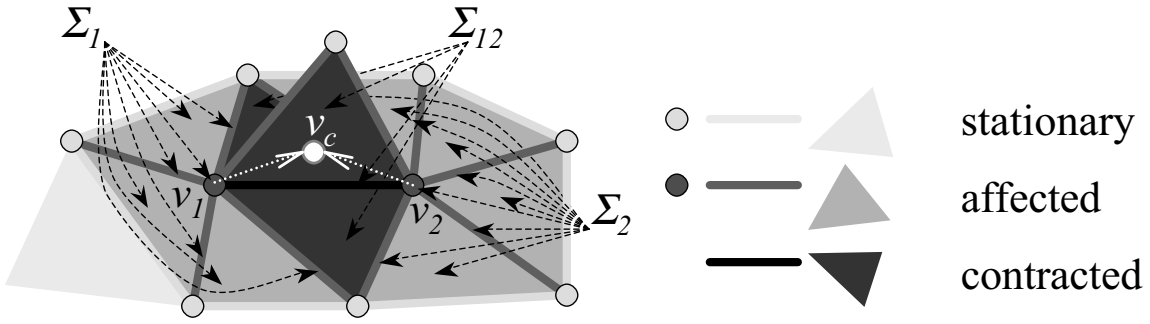

Fig. 4. Illustration of stationary, affected and contracted simplices in a non-manifold edge contraction.

v_1 and v_2 are contracted onto the target vertex v_c (see Figure 4). All simplices incident to one of the contraction vertices are called *affected* as they are sheared during the contraction. Simplices incident to both contraction vertices are called *contracted* as they are eliminated. The affected simplices can cause two problems as illustrated in Figure 5. Either an intersection can be caused as depicted in a) or a simplex can switch from the inside to the outside of a closed surface b). The inside-outside switch does not cause an intersection but can change the look of the model, when the switched simplex has a different color as the enclosing surface.

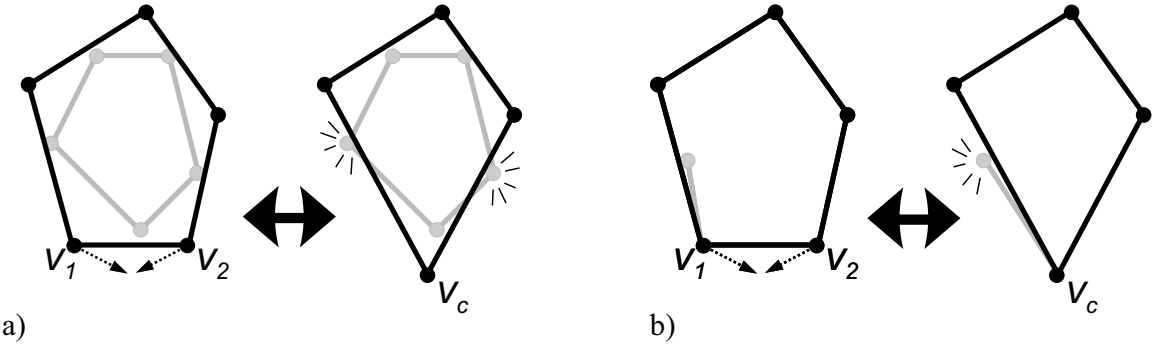

Fig. 5. a) intersection caused by vertex contraction b) inside-outside switch of differently colored edge.

Both problems can be detected by parameterizing the contraction operation over time as Hoppe did to generate geomorphs[9] and by looking for a collision between the simplices. An intersection is a collision whereas an inside-outside switch always cause a collision. Prevention of collisions will also ensure that no intersection arises during a geomorph.

5.1. *Classification of Collisions*

Let Σ_1 be the set of affected simplices incident only to v_1 and Σ_2 the set incident to v_2. Σ_{12} is the set of contracted simplices incident to both v_1 and v_2. The remaining simplices are called *stationary* and collected in the set Σ_0. Only the stationary simplices in a bounding box containing affected and contracted simplices and the target vertex can cause a collision. These are collected by a range query with the spatial grid data structure.

We parametrize the vertex pair contraction operation over the time interval $t \in [0, 1]$ by specifying the movement of the contraction vertices

$$v_j(t) = v_j + t \cdot (v_c - v_j)\,, j \in \{1, 2\}, \tag{1}$$

where we distinguish between the starting locations v_j of the contraction vertices and their time evolution $v_j(t)$ only by the additional time parameter. As we cleaned up all intersections in the beginning, we can start off with the precondition that there is no collision at $t = 0$. We distinguish five types of collisions:

- *hit collisions of the first kind* between a simplex from $\Sigma_1 \cup \Sigma_2$ and a simplex from Σ_0,
- *hit collisions of the second kind* between a simplex from Σ_{12} and a simplex from Σ_0,
- *fan collisions* between two simplices from Σ_1 or two simplices from Σ_2,
- *contraction collisions of the first kind* between a simplex from Σ_1 and a simplex from Σ_2 and
- *contraction collisions of the second kind* between a simplex from $\Sigma_1 \cup \Sigma_2$ and a simplex from Σ_{12} or between two simplices from Σ_{12}.

5.2. *Efficient Collision Computations*

To reduce the number of to be checked collisions and the number of to be discussed and implemented collision types, we state the following lemma, which directly follows from the continuous parameterization of the pair contraction operation.

Lemma 5.1. *If a pair contraction is applied to a simplicial complex without self-intersections, any collision between an edge and a triangle or between two triangles is preceded by or coincides with a collision between a vertex and a simplex or between two edges.*

The lemma implies that it is sufficient to test for collisions between a vertex and another simplex or between two edges, because if any other collision arises there will also be a vertex-simplex or edge-edge collision happening even earlier in time.

If we split the pair contraction into two phases by firstly dragging vertex v_1 with fixed v_2 onto v_c and by secondly dragging v_2 onto v_c, only hit collisions of the first kind can arise. All to be checked vertex-simplex and edge-edge hit collisions of the first kind can be detected by an intersection test between the time sweep of

the affected vertex or edge with the stationary simplex as illustrated in Figure 6, where without loss of generality only the dragging of v_1 onto v_c is shown.

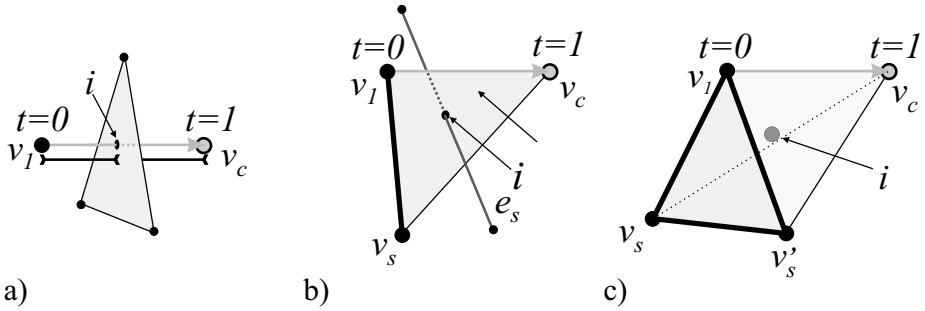

Fig. 6. Detection of hit collisions of the first kind with the time sweep of the affected a) vertex, b) edge, c) triangle and calculation of collision time.

The time sweep of the affected vertex is a line segment, which needs to be tested against all stationary vertices, edges and triangles. Figure 6 a) illustrates the case of a stationary triangle. Segment-vertex, segment-segment and segment-triangle intersection tests can be implemented efficiently. In Figure 6 b) the time sweep of an affected edge is shown to be a triangle, which needs to be tested for intersection with all stationary vertices and edges. Finally, the time sweep of an affected triangle is a tetrahedron as depicted in c). Here only intersection tests with stationary vertices are necessary, which reduce to a simple point inclusion test. In all hit collision tests of the first kind, the collision time is given by the barycentric coordinate of the intersection point i within the time sweep simplex of the affected simplex.

If we do not follow the two phase strategy one can additionally show, that fan collisions and contraction collisions of the second kind always coincide with another type of collision and therefore do not have to be tested. The intersection tests for hit collisions of the second kind and contraction collisions of the first kind are slightly more complicated.

5.3. *Detecting Hit Collisions of the Second Kind*

The collision detections between a contracted edge or triangle and stationary simplices are very similar to the hit collisions of the first kind. Figure 7 illustrates the two remaining cases after applying lemma 5.1. In a) an edge is contracted and the time sweep is again a triangle. For any time $t \in [0, 1]$ the contracting edge $v_1(t)v_2(t)$ is parallel to the edge v_1v_2. This follows directly from the theorem on intersecting lines. If a vertex or stationary edge is hit, the collision time is proportional to the distance of i from v_1v_2, zero if i is on v_1v_2 and one if $i = v_c$. The barycentric coordinate $\sigma_c(i)$ corresponding to the target vertex behaves exactly the same, such that $t_i = \sigma_c(i)$.

The case of a contracted triangle hitting a point as depicted in b) is very similar.

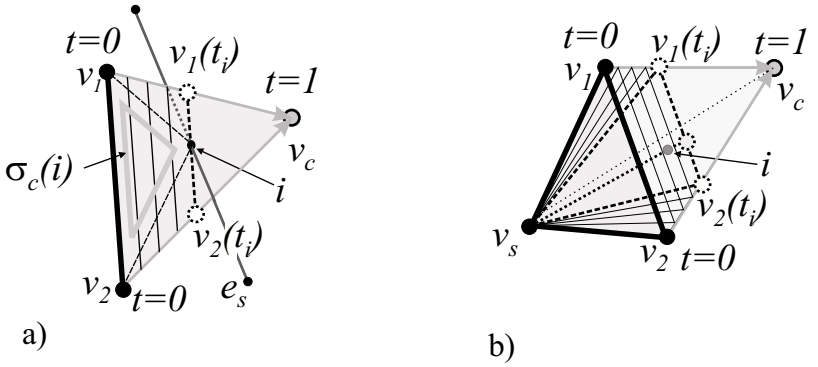

Fig. 7. a) contracted edge collides with a stationary edge, b) contracted triangle collides with stationary vertex.

At collision time the intersection point i is inside the triangle $v_1(t_i)v_2(t_i)v_s$. t_i is computed by moving i in barycentric coordinates inside the contracting triangle onto the triangle $v_1v_2v_c$ such that σ_s becomes zero. This results in the just discussed case yielding

$$t_i = \frac{\sigma_c(i)}{\sigma_1(i) + \sigma_2(i) + \sigma_c(i)}. \tag{2}$$

5.4. *Detecting Contraction Collisions of the First Kind*

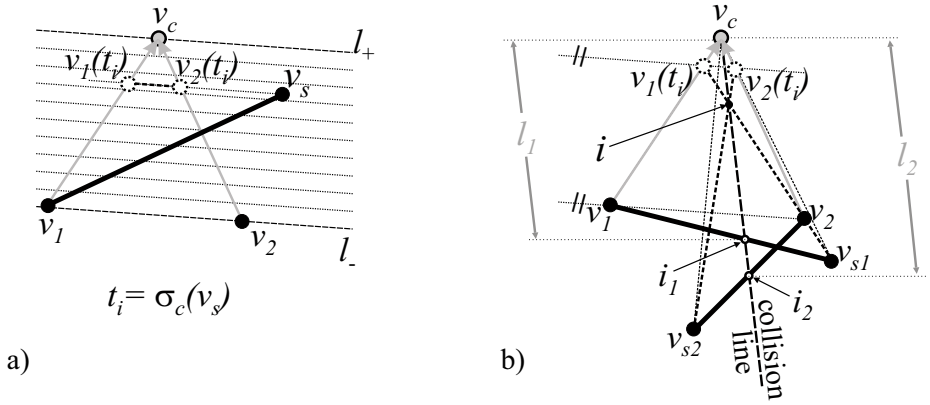

Fig. 8. a) affected edge $v_1(t)v_s$ collides with the other contraction vertex $v_2(t)$, when $v_1(t_i), v_2(t_i)$ and v_s are collinear, b) two affected edges incident to other contraction vertices collide at time t_i in the intersection point i.

Contraction collisions of the first kind are collisions between a simplex from Σ_1 and a simplex from Σ_2. There are three to be considered cases: a collision between a contraction vertex, let us say v_2, and an affected edge or triangle incident to v_1 and a collision between two affected edges incident to v_1 and v_2 respectively. The vertex-edge collision is illustrated in Figure 8 a). A collision of $v_1(t)v_s$ with v_2 can only happen, if v_1, v_2, v_s and v_c are coplanar. It happens exactly, when $v_1(t), v_2(t)$ and v_s are co-linear and when $v_1(t)$ is further apart from v_s than $v_2(t)$. This can

only happen, when v_s is within the line l_- through v_1 and v_2 and the parallel line l_+ through v_c. The collision time t_i is proportional to the distance of v_s from l_- with zero on l_- and one on l_+. Thus t_i is equal to the barycentric coordinate $\sigma_c(v_s)$ of the triangle $(v_1 v_2 v_c)$.

The case when a triangle collides with a vertex works exactly the same. Suppose v'_s of the affected triangle $(v_1 v_s v'_s)$ lays in the plane through $v_1 v_s$ orthogonal to the drawing plane and $v_2 v_c$ intersects the affected triangle in the intersection point i. Suppose further more that the line through $v_1 i$ intersects the edge $v_s v'_s$ in i_s. Then the collision time is computed from $\sigma_c(i_s)$.

The most complicated collision at all is the collision between two affected edges incident to the two different contraction vertices as depicted in figure 8 b). A necessary condition for the collision of the edges $v_1(t)v_{s1}$ and $v_2(t)v_{s2}$ is that the triangles $(v_c v_1 v_{s1})$ and $(v_c v_2 v_{s2})$ intersect in the *collision* line, that intersects $v_1 v_{s1}$ as well as $v_2 v_{s2}$. The edge intersections with the collision line are denoted i_1 and i_2. The two affected edges can only collide on the collision line. In the example of figure 8 b) the edges collide in the intersection point i, for which the contraction vertices $v_1(t_i)$ and $v_2(t_i)$ must be on a line parallel to $v_1 v_2$. To find the collision point i in barycentric coordinates with regard to triangle $\tau_1 = (v_c v_1 v_{s1})$ we search the intersection between the ray from $v_1(t_i) = (t_i, 1 - t_i, 0)^T$ in direction of v_{s1} and the ray from $i_1 = (0, 1 - \sigma_{s1}(i_1), \sigma_{s1}(i_1))^T$ to v_c. This yields the following equations:

$$(1-\lambda)\cdot\begin{bmatrix} t_i \\ 1-t_i \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \lambda \end{bmatrix} = (1-\gamma)\cdot\begin{bmatrix} 0 \\ 1-\sigma_{s1}(i_1) \\ \sigma_{s1}(i_1) \end{bmatrix} + \begin{bmatrix} \gamma \\ 0 \\ 0 \end{bmatrix}. \tag{3}$$

Now we can eliminate λ and γ and compute the intersection point i with respect to the barycentric coordinates of triangle τ_1. The same can be done to compute the barycentric coordinates with respect to triangle $\tau_2 = (v_c v_2 v_{s2})$ where $v_2(t_i)$ has exactly the same barycentric representation as $v_1(t_i)$ with respect to τ_1 and i_2 is represented by $(0, 1 - \sigma_{s2}(i_2), \sigma_{s2}(i_2))^T$. At the intersection the σ_c components of the two barycentric representations need to be compared. Suppose $\sigma_{c,\tau_1}(i)$ is the first barycentric coordinate of i with respect to τ_1 and σ_{c,τ_2} with respected to τ_2. If further more l_1 is the distance of i_1 from v_c and l_2 the distance from i_2 to v_c, then at the intersection holds

$$l_1 \cdot (1 - \sigma_{c,\tau_1}) = l_2 \cdot (1 - \sigma_{c,\tau_2}).$$

Solving equations 3 for σ_c yields

$$\sigma_{c,\tau_j} = \frac{t_i \cdot (1 - \sigma_{sj}(i_j))}{1 - t_i \cdot \sigma_{sj}(i_j)}, j \in \{1, 2\}.$$

Plugging this in the previous equation and solving the resulting quadratic equation for t_i yields either $t_i = 1$ or more interesting

$$t_i = \frac{l_1 - l_2}{\sigma_{s2}(i_2) \cdot l_1 - \sigma_{s1}(i_1) \cdot l_2}. \tag{4}$$

We gather all and specify an algorithm to test the intersection between two affected edges $v_1 v_{s1}$ and $v_2 v_{s2}$.

- check if planes through $(v_c v_1 v_{s1})$ and $(v_c v_2 v_{s2})$ intersect
- compute the barycentric coordinate $\sigma_1(i_1)$ and $\sigma_2(i_2)$ and check if they are in the interval $[0, 1]$
- compute l_1, l_2 and then with equation (4) the collision time t_i and check if t_i is in the interval $]0, 1[$.
- if all checks succeeded, return collision time t_i, otherwise no collision.

5.5. *Numerical Issues*

The first numerical issue arises when one of the time-sweeps degenerates, i.e. when the time-sweep of an edge degenerates from a triangle to an edge or the time-sweep tetrahedron degenerates to a triangle or even to an edge. In all these cases the test with the degenerated element is not necessary as a test with a lower dimensional simplex incident to the degenerated simplex will already find the collisions.

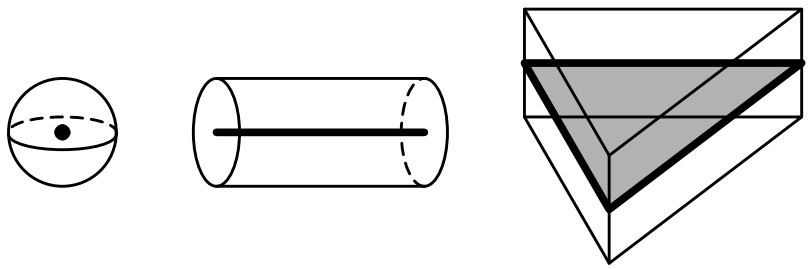

Fig. 9. Illustration of blow-up by the collision epsilon to avoid numerical instabilities.

The second numerical issue arises when one surface part comes such close to another, that the geometric tests become numerical unstable and lead to different results depending on the order of computation. To avoid these problems we virtually blew up all simplices by a *collision epsilon* ϵ_C as illustrated in figure 9. The collision epsilon is not to be mixed up with the approximation error achieved by the simpification algorithm. It is much smaller and of the size of the numerical precision used for the calculations.

The geometric tests can be easily modified to support the collision epsilon. The time-sweep elements are as well blown up in the same manner, except for the time-sweep tetrahedra. There is no need to blow the latter up as the incident time-sweep triangles and edges describing its surface are blown up. Finally, we blow up the bounding box computed for the collection of the stationary simplices also by the collision epsilon.

The blow up of the simplices forces us to compute several square root operations more; for example the normal vectors of the planes defined by the triangle vertices need to be normalized. This slows down the collision tests by about ten percent, but the overall running time was not significantly affected. On the other hand one

can also adjust the collision epsilon to a number, significantly larger than zero, for example one percent of the average edge length. This will avoid problems arising when the model is rendered with a z-buffer of finite precision.

6. Avoidance of Self-Intersections

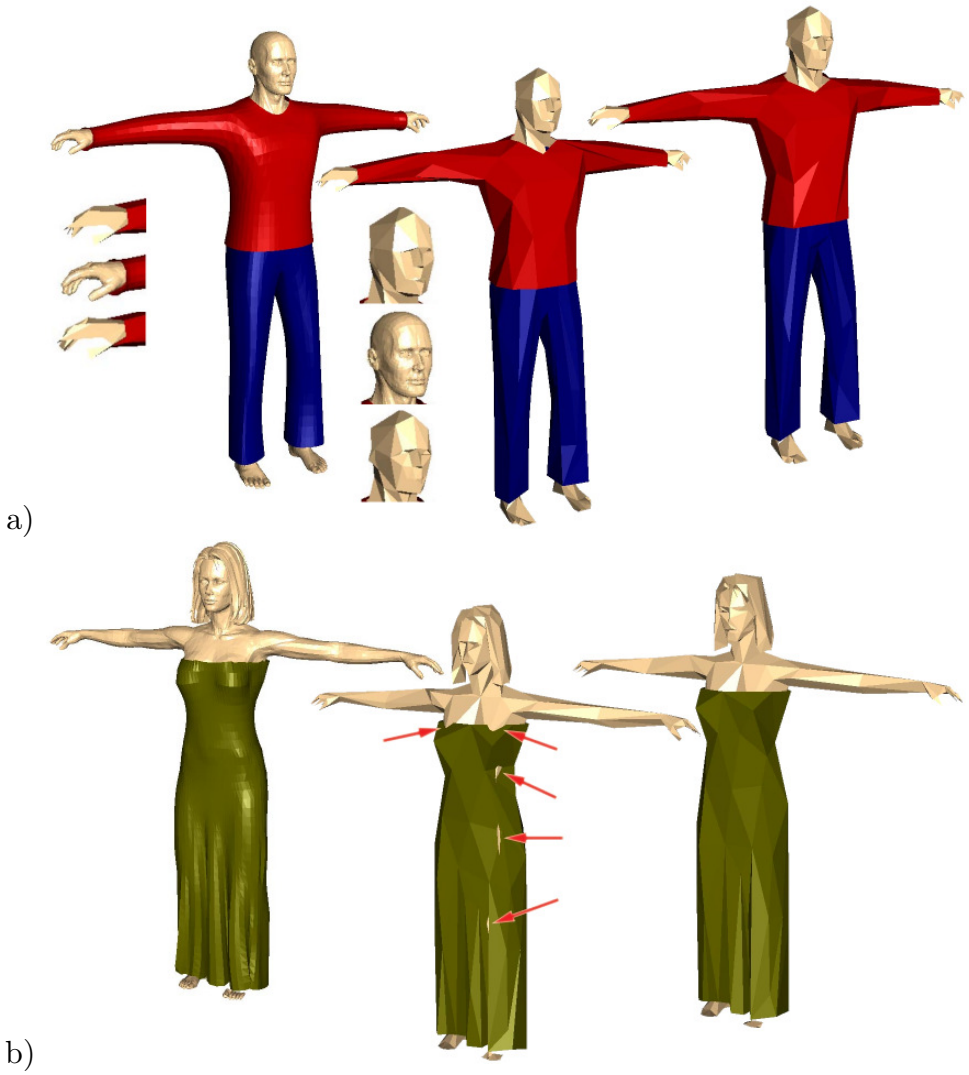

Fig. 10. a) dressed man: original, reduced to 600 vertices with prevention of self-intersections and with avoidance; close-ups: top – prevention, middle – original, bottom – avoidance b) woman model: original, reduced to 370 vertices without intersection test and with avoidance.

The simple prevention strategy of just discarding operations that cause a collision does not allow the generation of very coarse approximations with low error. As shown in the results section does the approximation error grow much faster if a lot of low error operations have to be discarded and furthermore is it possible that the simplification process reaches a point when all possible operation would cause a self-intersection. In the model of the dressed man most operations of vertices close

to the cloths won't be valid in a late stage of the simplification process. Thus only the head, hands and feet can be reduced further yielding a bad approximation of these parts as can be seen in Figure 10 a), where we zoomed onto head and left hand. The top hand and head are from the model in the middle, reduced with prevention only. The bottom parts are produced with the strategy proposed in this section resulting in the simplified model on the right.

To actually avoid the large number of invalid operations, it is either possible to look for a different target position, that does not cause a self-intersection, or to move the part of the mesh, into which the contracted part bumped, or do even both. We decided to only change the target position, as in this case the progressive representation of the model can be stored in exactly the same way as if no self-intersection avoidance had been performed.

Thus we simply try to find a target location that does not cause a self-intersection. In some situations this is not possible at all. But in most situations there is a so called *valid region* of valid target positions onto which the two contraction vertices can be contracted without causing a self-intersection. The target position, which is optimal with respect to the quadric error metric, causes a self-intersection and is therefore not in the valid region. What we are looking for is the location in the valid region that minimizes the quadric error metric. As the explicit computation of the valid region is very complicated, we came up with three approximate solutions:

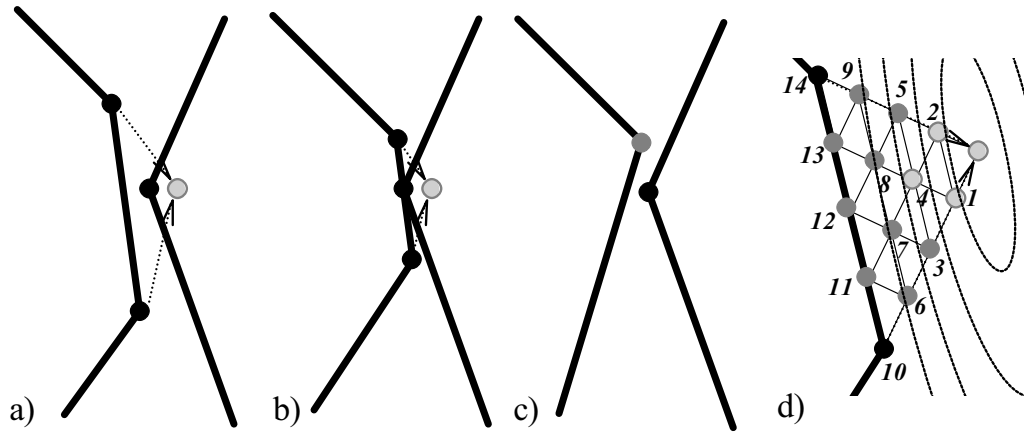

Fig. 11. First hit (a-c) and barycentric sampling (d) self-intersection avoidance strategy a) edge contraction with optimal target location as dashed circle, b) situation at first hit, c) location of upper contraction vertex incident to colliding edge is selected as new target location, d) again is the target location invalid. The triangle formed by contraction vertices and target location is sampled in barycentric space to find one of the valid target locations (filled circles). The dotted elliptical lines are iso-lines of the quadric error metric.

first hit: For all collision tests introduced in the previous section one automatically computes the parameter value $t = t_c$, when the collision arises. By minimizing the time parameter over all colliding affected simplices one can determine the time

parameter $t_{\min}$ when the first collision arises. In the first hit strategy we guess the new collision free target location from the location of the contraction vertex incident to the simplex causing the collision at time $t_{\min}$. In the case this target location is again causing a collision, we use the same strategy again up to five times before we give up. Figure 11 a) to c) illustrates the first hit strategy on a 2d example. a) shows the edge contraction with the target position of lowest quadric error metric. In b) the partially contracted edge is drawn at $t_{\min}$, when the first collision arises. As both contraction vertices are incident to the colliding simplex, one of them – the upper one – is selected as new to be checked target location. In this case the location is valid (c) and no further attempts are necessary.

barycentric sampling: Here we sample among all possible target locations only the ones on the triangle spanned by the two contraction vertices and the QEM-optimal target location. We sample the triangle on fourteen further locations, that result from two one to four subdivisions as shown in Figure 11 d). We sorted the sample locations by increasing quadric error metric, whose iso-lines are illustrated by the dotted lines in Figure 11 d). The numbers show the sort order, in which the collision checks are performed until the first yields a valid target location or until all of them failed and the operation has to be discarded.

extensive sampling: As the results section shows, it is not obvious at all how to do an extensive sampling of the possible target location in order to keep the approximation error low. In our extensive sampling strategy we sample the space around the optimal target location in 26 directions given by the vertices, edge centers and face centers of an axis aligned octahedron, i.e. all possible directions with one, two or three ± 1 as coordinates. We sampled an iso-surface of the quadric error metric, where it is hit by the 26 directions. Let $\epsilon^2_{1/2}$ be the quadric error values at the initial locations of the contraction vertices and ϵ^2_c the one at the optimal target location. Let

$$\epsilon^2_\alpha = \alpha \min\{\epsilon^2_1, \epsilon^2_2\} + (1-\alpha)\epsilon^2_c.$$

We determined the minimum value for α by an interval subdivision of the interval $[0, 1]$, where we checked for each α, if at least one valid target location exists.

In case a valid target location is found by one of the strategies, we evaluated the quadric error metric at the new approximation of the target position and re-inserted the operation into the priority queue. Figure 10 a) shows the man simplified self-intersection free with intersection avoidance by the first hit strategy on the right side.

7. Results

We tested our collision free simplification algorithm on four test data sets, a dressed man, a dressed woman, the nested spheres and the bunny as an example of a model that does not cause self-intersections. Figure 1 d) shows in the lower part the reduced nested sphere model. In Figure 10 the man a) and woman models a) are

models	man	woman	spheres	bunny
orig nr. verts	17060	19498	1028	34838
redu. nr. verts	600	370	100	500
time orig.	1.87	2.02	1.23	1.29
time prevention	7.52	8.45	4.30	3.11
time first hit	10.18	10.71	4.43	3.18
time barycentric	18.52	17.67	7.058	3.06
time extensive	63.77	72.89	14.02	3.62
HD orig.	0.1050	0.1597	0.4841	0.1069
HD prevention	0.1293	0.2027	0.4719	0.1070
HD first hit	0.1246	0.1850	0.4894	0.1074
HD barycentric	0.1183	0.1648	0.4816	0.1072
HD extensive	0.1195	0.1708	0.4857	0.1071

Table 1. Experimental results. Reduction times are in seconds per 1000 pair contractions and where measured on an Athlon processor with 1200MHz.

depicted. The bunny did not produce any self-intersections and is not shown. Table 1 gathers the experimental results. The first two rows show the number of vertices of the original and simplified models. The next five rows show the simplification time without and with prevention and with the three different avoidance strategies. The last five rows show the Hausdorff distance measured with the M.E.S.H. tool[1] between the simplified models and the original models.

For the bunny model only two collisions arose. The increase of running time is due only to the insertion and removal of the simplices to/from the spatial data structure. For the other models there are a lot of collision tests because of the large portion of double layered parts. The measurements of the Hausdorff distances show how the problem of the invalidation of operations by the simple collision detection strategy can be nearly completely avoided by the barycentric sampling.

In Figure 12 we plotted the Hausdorff distance (a) and the RMS (b) over the number of remaining vertices for the five different strategies. One first notices, that the no prevention strategy achieves the best approximation. Secondly, for the simple prevention strategy the error increases very fast if the model is coarse. It is even not possible to reduce the man model below 150 vertices as all operations cause a collision. In the RMS sense at first the extensive sampling is superior, then the first hit and in the end the barycentric sampling. In the Hausdorff distance also the barycentric sampling is the winner for very coarse approximations although the first hit strategy is much better before. This is probably the case as the first hit moves the target location right before a collision arises, whereas in the barycentric setting only a fixed resolution is tested. Interesting is also that the extensive sampling strategy does not really perform well in the terms of approximation quality. It seems to be much better to restrict the search for a valid target location within the triangle

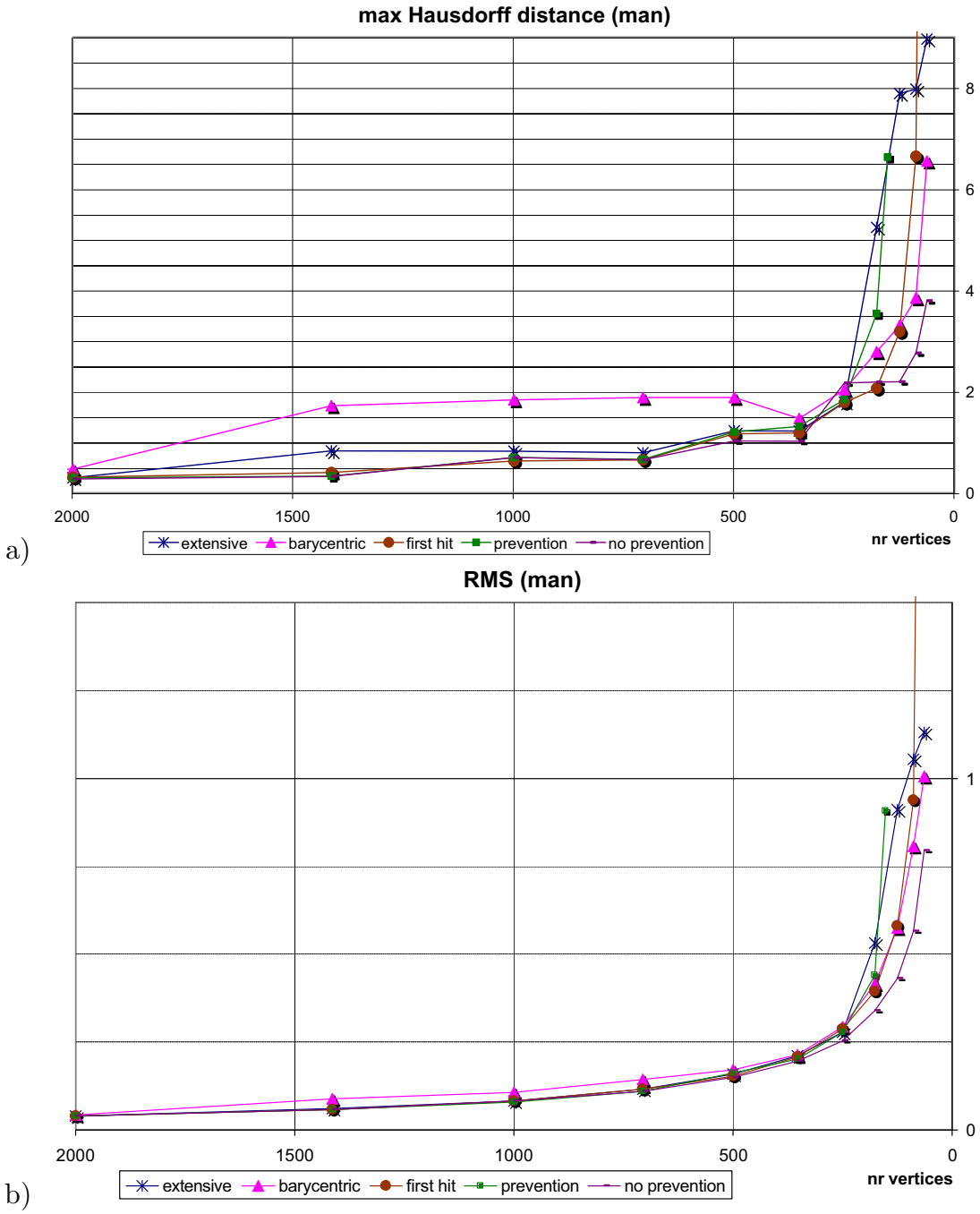

Fig. 12. Approximation quality for the man model over the number of vertices left in the reduced model on a logarithmic scale in percent of the bounding box diagonal: a) two-sided Hausdorff-distance, b) RMS

spanned by the contracted vertices and the target location.

In terms of avoidance on the other hand does the extensive sampling perform best as Figure 13 b) shows, where we plotted the number of avoided collisions. Here the extensive sampling outperforms the other strategies, followed by the first hit strategy. Finally, we plotted the elapsed time over the number of left vertices in

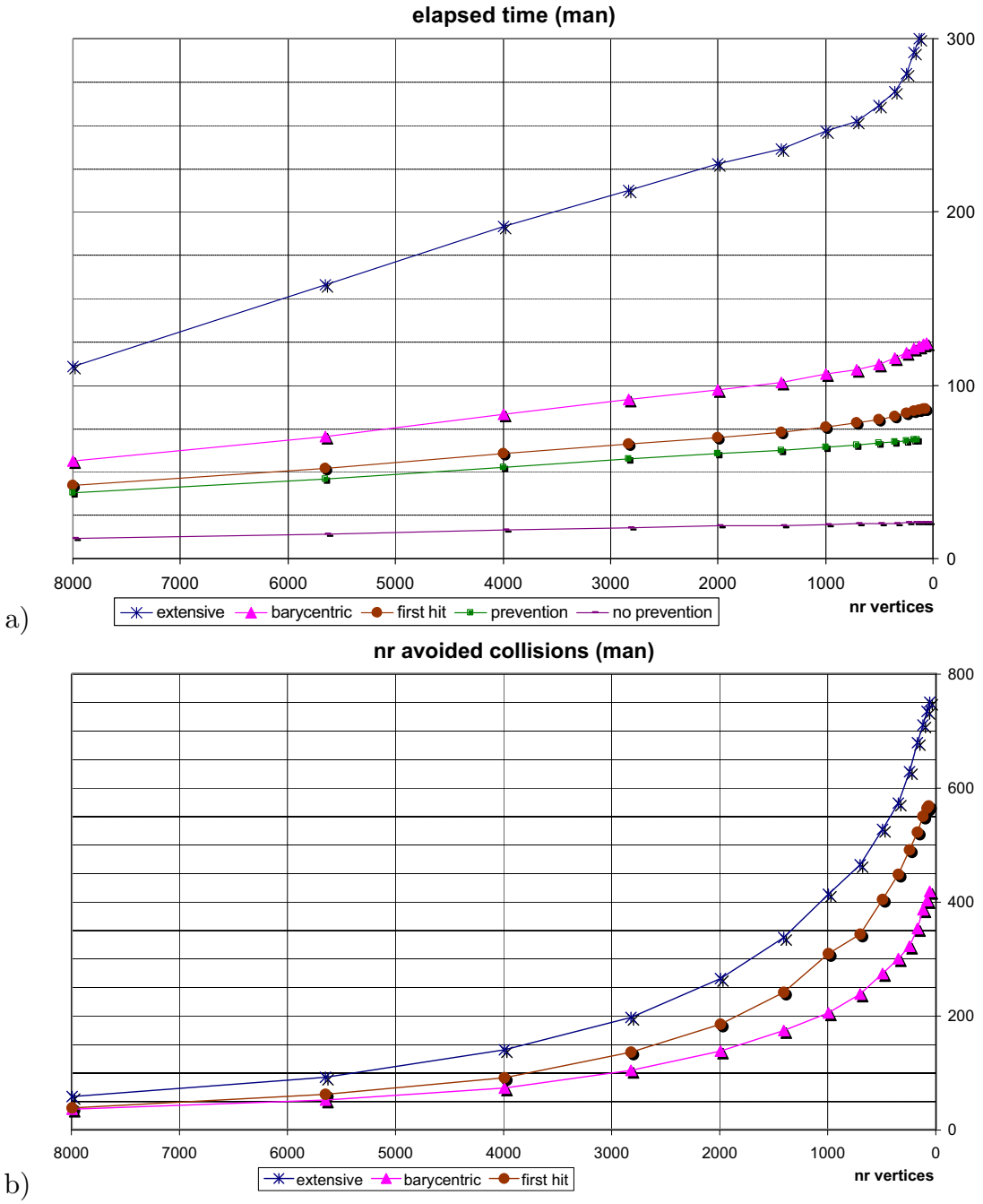

Fig. 13. Measurements for the man model over the number of vertices left in the reduced model: a) consumed time, b) number of avoided operations

Figure 13 b). It shows that the three simplest strategies have close to constant speed, whereas the barycentric sampling and the extensive sampling slow down when the model becomes coarse and more searches for collision free target locations become necessary. We can conclude from the results that the first hit strategy is sufficient and is only ten percent slower than prevention alone. The best results for very coarse approximations can be achieved with the barycentric sampling strategy.

8. Conclusion

We presented for the first time an approach to globally prevent *and* avoid self-intersections during mesh simplification. We based our implementation on the general vertex pair contraction approach and support non-manifold meshes and modification of the topology. Other simplification approaches could be supported as well. Progressive transmission and several levels of detail can be supported self-intersection free without any modifications, as we did not introduce any new operations but only changed the target locations of the pair contractions. From the introduced strategies to find valid target locations in the case of self-intersections the first-hit strategy proved to be very efficient in terms of running time and achieved approximation quality. The proposed method is the first that can create high quality simplified versions of dressed people.

Acknowledgements

We would like to thank the Virtual Try On group for providing the models of the dressed humans.

References

1. N. Aspert, D. Santa-Cruz, and T. Ebrahimi. Mesh: Measuring error between surfaces using the hausdorff distance. In *IEEE International Conference on Multimedia and Expo 2002 Proceedings*, pages 705–708, 2002.
2. C. Bajaj and D. Schikore. Error-bounded reduction of triangle meshes with multivariate data. *SPIE*, 2656:34–45, 1996.
3. A. Ciampalini, P. Cignoni, C. Montani, and R. Scopigno. Multiresolution decimation based on global error. *The Visual Computer*, 13(5):228–246, 1997.
4. J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. P. Brooks, Jr., and W. Wright. Simplification envelopes. In *SIGGRAPH 96 Conference Proceedings*, pages 119–128, 1996.
5. M. de Berg, M. von Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Chapter 3*, pages 49–59. Springer, 1997.
6. M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH 97 Conference Proceedings*, pages 209–216, 1997.
7. A. Guéziec. Locally toleranced surface simplification. *IEEE Transactions on Visualization and Computer Graphics*, 5(2):168–189, 1999.
8. B. Hamann. A data reduction scheme for triangulated surface. *Computer Aided Geometric Design*, 11:197–214, 1994.
9. H. Hoppe. Progressive meshes. In *SIGGRAPH 96 Conference Proceedings*, pages 99–108, 1996.
10. R. Klein, G. Liebich, and W. Straßer. Mesh reduction with error control. In *IEEE Visualization 96 Conference Proceedings*, pages 311–318, 1996.
11. L. Kobbelt, S. Campagna, and H.-P. Seidel. A general framework for mesh decimation. In *Proceedings of the 24th Conference on Graphics Interface*, pages 43–50, 1998.
12. P. Lindstrom and G. Turk. Fast and memory efficient polygonal simplification. In *IEEE Visualization 98 Conference Proceedings*, pages 279–286, 1998.
13. J. Popović and H. Hoppe. Progressive simplicial complexes. In *SIGGRAPH 97 Conference Proceedings*, pages 217–224, 1997.
14. R. Ronfard and J. Rossignac. Full-range approximation of triangulated polyhedra. *Computer Graphics Forum*, 15(3):67–76, C462, 1996.
15. J. Rossignac and P. Borrel. Multi-resolution 3d approximations for rendering complex scenes. *Modeling in Computer Graphics: Methods and Applications*, pages 455–465, 1993.
16. W. J. Schroeder, J. A. Zarge, and W. E. Lorensen. Decimation of triangle meshes. In *SIGGRAPH 92 Conference Proceedings*, pages 65–70, 1992.
17. O. G. Staadt and M. H. Gross. Progressive tetrahedralizations. In *IEEE Visualization 98 Conference Proceedings*, pages 397–404, 1998.
18. G. Zachmann. Optimizing the collision detection pipeline. In *The First International Game Technology Conference GTEC*, January 2001.
19. S. Zelinka and M. Garland. Permission grids: practical, error-bounded simplification. *ACM Transactions on Graphics*, 21(2):207–229, 2002.