View-dependent Simplification using Parallel Half Edge Collapses

Thomas Odaker
Ludwig Maximilians Universitaet
München, Germany
odaker@a1.net

Dieter Kranzlmueller
Ludwig Maximilians Universitaet
München, Germany
kranzlmueller@ifi.lmu.de

Jens Volkert
Johannes Kepler University
Linz, Austria
jv@gup.jku.at

ABSTRACT

Highly detailed models are a requirement for many applications in computer graphics. The necessary level of detail, however, may vary depending on the application. To provide a tradeoff, mesh simplification is used to generate approximations of a model which can be used to reduce processing time.
We present a parallel approach to triangle mesh simplification that is designed to allow fast, view-dependent simplification of manifold triangle meshes. Our approach performs a vertex analysis on every vertex of a given triangle mesh and selects a set of vertices for removal. Vertex removal is executed using the parallel half edge collapse. Based on the half edge collapse that replaces an edge with one of its endpoints, we have devised a set of boundaries that enable parallel application of half edge collapses even on neighbouring vertices.
Since the mesh topology may not allow removal of all vertices marked for removal in one step, we apply multiple iterations of the parallel half edge collapse, reevaluating remaining vertices marked for removal for further improvement of results.

Keywords
mesh simplification, level of detail, half edge collapse, computer graphics, view dependent simplification, real-time rendering

1 INTRODUCTION

Simplification of triangle meshes has been a well researched area for several decades [Cla76a]. Mesh simplification is the process of applying a simplification operator to a given triangle mesh with the effect of reducing the number of vertices or triangles that make up the mesh [Lue02a].
Mesh simplification is widely used as an approach to reduce the resources needed for processing a mesh by reducing the stored geometry data.
The disadvantage of simplification is usually a loss of detail. The triangles of a mesh are used to represent a surface shape. Removing triangles or vertices from a triangle mesh therefore means the result cannot represent the object as accurately as the original. Simplification algorithms can be aimed either to remove a large amount of vertices or triangles, create a simplification that represents the original object well or to provide fast processing times.

Over the years a wide range of algorithms for triangle mesh simplification has been developed with varying results in terms of performance and quality of the simplified mesh. The ones most important to this paper can roughly be classified in three categories:

Vertex clustering This algorithm presented by Rossignac and Borrell in [Ros92a] is designed to work on arbitrary polygonal models. First the bounding box of an object is determined. This bounding box is then divided into a number of cells. All vertices within a cell are clustered into a single vertex and the faces of the model are updated accordingly. While this approach can be very fast, it can also cause drastic alterations to the topology of a model and create low quality simplifications. Attempting to improve these shortcomings, several variants to cell generation have been devised [Sch03a], [Low97a].

Vertex removal Presented in Schroeder et. al. [Sch92a], the vertex removal approach removes a

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

vertex and all its adjacent edges and triangles from a mesh and then retriangulates the resulting hole.

Edge contraction Edge contraction provides mesh simplification by removing edges from a mesh and replacing it with a vertex [Hop93a]. This approach is usually performed iteratively, selecting one edge at a time and contracting it.

While many algorithms are designed to generate a generic simplification of a mesh, some have been devised for generating a view-dependent simplification. The latter take a current camera position into account and aim to minimize the visible differences between the simplification and the original.
One example can be found in Hu et al. [Hu09a] with the presentation of this approach being extended in [Hu10a]. It describes a bottom up approach where a simplified mesh is stored. During runtime the desired level of detail is restored by applying precalculated operations stored in a data structure.
Papageorgiou and Platis [Pap15a] present a GPU-accelerated approach that also relies on edge collapses with the goal to execute a number of edge collapses in parallel and therefore speed up the simplification. Their algorithm selects a number of independent areas. Each area can safely be simplified by an edge collapse without affecting other areas. Selection of independent areas and execution of edge collapses is repeated, until the desired simplification target is reached.
Another example is presented by DeCoro and Tatarchuk [DeC07a]. It describes a GPU accelerated implementation of vertex clustering to provide a fast, view-dependent simplification of an arbitrary mesh.
In comparison to these approaches, the parallel half edge collapse was designed not to rely on any precomputed simplification operations and provide a topology preserving, parallel approach to simplification that selects the simplification operations at runtime.

The parallel half edge collapse was designed to enable view-dependent real time simplification of manifold triangle meshes. The simplification operator of choice is the half edge collapse.
The edge collapse operator (Figure 1) is applied to a pair of vertices (V_1, V_2) connected by an edge e. It collapses V_1 and V_2 into a single vertex V', removing e and all triangles that contain it from the mesh [Hop96a]. The position of V' is chosen freely. This allows to optimize the replacement position for e and achieve a better simplification as well as improve the representation of the original mesh.
A more restrictive version of the edge collapse is the half edge collapse (Figure 2). The half edge collapse uses V_1 or V_2 as replacement for e. It therefore replaces an edge with one of its endpoints and does not change any vertex data like the stored normal.

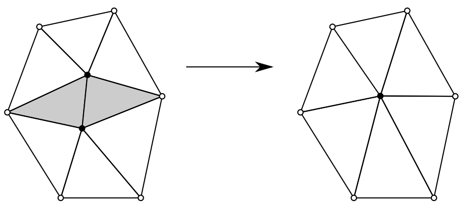
Figure 1: Edge collapse

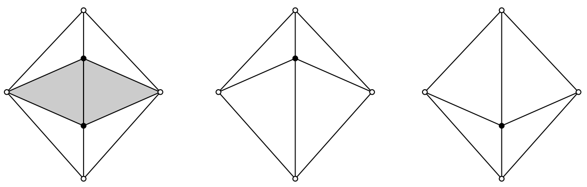
Figure 2: Half edge collapse

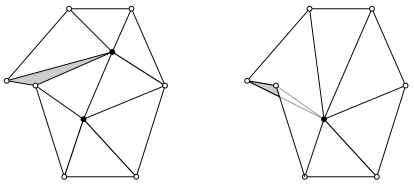
Figure 3: Edge collapse causing a mesh foldover

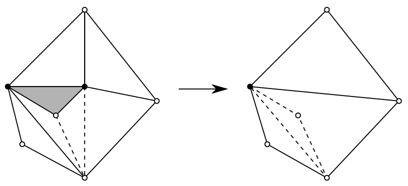
Figure 4: Edge collapse causing a topological inconsistency

Although the edge collapse and the half edge collapse are easy to implement, they can have a negative effect on the simplified mesh by causing a foldover (see Figure 3) or a topological inconsistency (Figure 4) which has to be avoided [Lue02a].

2 ALGORITHM OVERVIEW

Initially a vertex analysis is performed on all vertices of a given manifold mesh that selects a number of vertices for removal.
Given a vertex V that has been selected for removal, the algorithm determines all vertices $N = \{N_1, N_2, ..., N_q\}$ that share an edge with V. Each of these edges is considered a possible half edge collapse. V is removed by selecting a vertex pair V, N_i and performing a half edge collapse on it. The replacement position for the edge is N_i, since V is supposed to be removed from the mesh.
To speed up the simplification a parallel approach for the vertex removal is introduced. The algorithm proposed in this paper analyses all vertices $V = \{V_1, V_2, ..., V_n\}$ of a mesh with the intention of selecting a set of vertices $R = \{VR_1, VR_2, ..., VR_o\}$ that should be removed ($S = \{VS_1, VS_2, ..., VS_p\}, S = V \setminus R$,

step 1 in Figure 5). Since every vertex in R is to be removed from the mesh, not all the neighbours propose a valid half edge collapse. Only edges connecting VR_i to a vertex $N_j \in S$ are valid choices for a half edge collapse.
For each vertex $VR_i \in R$ the algorithm tries to:

- Find all neighbours $N \in S$ of VR_i that share an edge.
- Select one edge e_k connecting the pair N_j, VR_i.
- Perform the half edge collapse on e_k using N_j as the replacement position and removing VR_i from the mesh.

The algorithm does not apply a series of iterative edge collapses. It rather processes all vertices in R in parallel. Although this approach has great potential to speed up the simplification process, it causes several difficulties:

- All neighbouring vertices of a vertex VR_i being included in R can prevent VR_i from being removed instantly.
- Parallel execution of half edge collapses on neighbouring vertices may cause hard to detect mesh foldovers and topological inconsistencies.

Not all vertices $VR \in R$ may have a neighbour in S. Since the algorithm only considers an edge connecting a vertex VR_i to a neighbour $N_j \in S$ a valid half edge collapse, it may be impossible for the vertex to be removed.
The removal of the vertices in R has to be divided into several iterative steps. In each step a list of removal candidates $C = \{C_1, C_2, ..., C_n\}$ is composed, containing all $VR \in R$ that have at least one neighbour $N_j \in S$ (step 2 in Figure 5). The algorithm tries to remove all vertices in C in parallel (step 3 in Figure 5). The steps of determining the candidate list and removing those vertices is repeated iteratively, until all $VR \in R$ have been processed and no vertices remain in R.

Another problem arising due to the parallelism of the algorithm is the selection of the individual replacement positions for vertices in R. As described by Xia et al. [Xia97a] a single edge or half edge collapse performed on a mesh may cause a mesh foldover or a topological inconsistency which has to be avoided. While this is relatively easy to detect by checking for rotations of normal vectors of an affected triangle before and after a collapse, the parallel approach chosen here renders this test invalid. It is possible that a single half edge collapse is valid - it does not have a negative effect on the mesh - the parallel execution

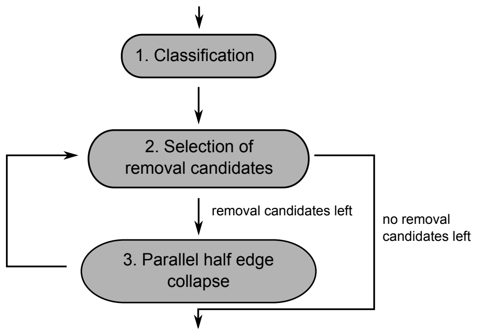
1. Classification
2. Selection of removal candidates
removal candidates left
no removal candidates left
3. Parallel half edge collapse

Figure 5: Algorithm overview

of two valid half edge collapses however, can cause a foldover or topological inconsistency.
To make sure that a possible replacement position for a vertex does not cause the aforementioned problems, a set of boundaries is defined. For each vertex $VR_i \in R$ an individual set of boundaries $B(VR_i)$ is calculated. It can be used to determine, which neighbouring vertices $N \in S$ can be safely used as a vertex pair for a half edge collapse. Any neighbour $N_j \in S$ that lies within the boundaries $B(VR_i)$ can be used as a vertex pair VR_i, N_j for a half edge collapse.

Given two vertices $VR_i, VR_j \in R$ are connected by an edge and have neighbours in S. VR_i and VR_j are to be processed in parallel and have one or more possible half edge collapses. In order to avoid foldovers and topological inconsistencies, the boundaries $B(VR_i)$ have to take all possible replacement positions for VR_j into account. The same condition is in effect for $B(VR_j)$ and the replacement positions for VR_i. The parallel half edge collapse was designed to avoid communication between neighbouring vertex removals to allow for a fast implementation. The boundaries are created to block all half edge collapses that may cause issues whenever a neighbouring vertex is removed in parallel. This may result in combinations of half edge collapses for neighbouring vertices in R that would not cause foldovers or topological inconsistencies being considered invalid.
The algorithm has to take into account that for some vertices in C no replacement position, that would not cause a foldover or a topological inconsistency, can be found. A distinction has to be made whether this is caused by the topology or by the restrictive boundary due to the execution of parallel half edge collapses and an isolated vertex removal could be safely executed.
Furthermore these restrictive boundaries may cause an issue where neighbouring vertices in C may mutually block a removal, effectively causing a deadlock and preventing a region of the mesh from being simplified. This situation has to be identified and resolved to avoid vertices remaining in R indefinitely and keep the

simplification from being completed.

3 VERTEX ANALYSIS

Given a manifold triangle mesh every vertex V_i is analysed and categorized either for being removed from the mesh or remaining. Two sets of vertices are created. $R = \{VR_1, VR_2, ..., VR_n\}$ contains a list of vertices that are to be removed while $S = \{VS_1, VS_2, ..., VS_m\}$ contains the remaining ones.
This initial classification of vertices does however not determine a final list of vertices that are removed from the mesh. Since several iterations of parallel half edge collapses may be performed on the mesh and the results of each iteration cannot be predicted, an additional step is introduced. After a half edge collapse has been executed on a vertex pair VR_i, VS_j, all vertices VR_k that were neighbours of VR_i are analysed again and may be removed from R and added to S. This step allows for an adaptation to the chosen half edge collapses and may improve the overall result of the simplification.

For classification a vertex analysis is performed. This step assigns an error value to each vertex of the mesh. Since this vertex error has to be updated during the reclassification step after each half edge collapse, the error metric was chosen with data dependency and computing time in mind.

The initial analysis is done on a per-vertex basis and only takes the position of neighbouring vertices into account. The error metric chosen here relies on the distance of the neighbouring vertices from the tangent plane of a vertex that is defined by the vertex normal stored for the vertex. Although this only takes the local effect of a vertex removal into account and ignores a possible removal of neighbouring vertices, it was chosen for two reasons. Firstly this error metric requires little computing time and since it only relies on neighbouring vertices, it does not require data collection before a calculation, which allows for fast updates after half edge collapses.
Secondly during reclassification after a half edge collapse some neighbours of vertices in C may be members of R and therefore marked for removal. These do not provide relevant data since they may be removed in subsequent iterations. The metric has to be easily adaptable to take into account data from neighbours only, that are to remain in the mesh.

Given a vertex V the tangential plane is constructed as $ax + by + cz + d = 0$ with $t = [a, b, c, d]$. For each neighbouring vertex N_i with the position $n = [n_x, n_y, n_z, 1]$ the quadratic distance from the tangential plane $d(V, N_i) = (t \bullet n)^2$ is calculated (Figure 6).
The final error value for a vertex $e(V)$ is defined as

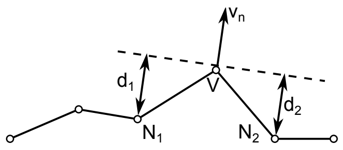

Figure 6: Vertex error calculation

the average distance between the tangent plane and the neighbouring vertices:

$$e(V) = \frac{\sum_{i=1}^{m}(d(V, N_i))}{m} \tag{1}$$

For a view-dependent simplification $e(V)$ is scaled using the angle between the view vector and the vertex normal as well as the distance between vertex and camera. This step has to be performed at runtime and computes the per-vertex classification. While the error $e(V)$ is computed from the static mesh data, the influence of the camera data allows for a view-dependent simplification as the camera data is updated before the simplification is computed.

Vertex classification into R and S is performed by comparing the calculated vertex error $e(V)$ with a user defined error threshold u. Any vertex with an error value smaller than or equal to u is marked for removal and stored in R. The remaining ones form S.
The problem that arises when using this approach is a potential selection of a majority of vertices (if not all) of a mesh for removal. Since the algorithm performs a half edge collapse on an edge that connects a vertex in R with one in S, the number of parallel operations could be limited severely. In case of all vertices being marked for removal, the execution of the parallel half edge collapse would be impossible.
To guarantee the algorithm is always functional and improve parallelism, an artificially modified vertex error is introduced for some vertices.
This approach selects a number of vertices from the mesh and assigns them a very high vertex error e_m. The artificially assigned error is defined as $e_m > u$. This guarantees these chosen vertices to always remain in the mesh and the algorithm remains functional.
For further refinement of this approach several "layers" of error manipulation are introduced. The first layer L_0 contains the aforementioned vertices that are assigned $e_m(V) > u$. Each additionally created layer L_i (i > 0) selects additional vertices and assigns them a predefined error value $e_m(L_i)$. This value is user defined. Modifying the user threshold u can hence be used not only to control the level of detail of the simplified mesh, but also to select a set of vertices that is to remain in the mesh with the purpose of accelerating the execution of the simplification.
For this approach a number of vertices has to be selected for each layer:

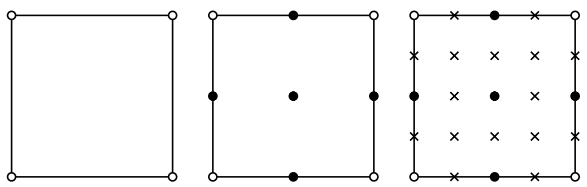
Figure 7: Example point generation

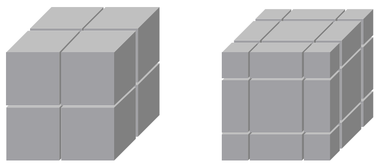
Figure 8: Example volume generation

- The bounding box of the mesh is determined and a 3-dimensional grid of points $G_0 = \{P_1^0, P_2^0, ..., P_n^0\}$ for layer L_0 is created within and with an equal axial distance from each other.
- Addition of layers L_i with sets of points $G_i = \{P_1^i, P_2^i, ..., P_o^i\}$. The new points are generated at halfway points between points in G_{i-1}. The distance between two points in G_i is $d(G_i)$. This distance is separate for each axis (2-dimensional example in Figure 7).
- For each point: generation of a volume $B(P_j^i)$ centered around P_j^i with a side length of $d(G_i)$ (trimmed to the bounding box). Figure 8 shows the volumes for the first two examples in Figure 7.
- For each volume: determination of all vertices $V(P_j^i)$ within the volume
- For each volume: selection of one vertex from $V(P_j^i)$ and manipulation of the vertex error

This approach creates a number of points and the corresponding volumes. Then a number of vertices has to be selected. For each point $P_i \in G_j$ the algorithm selects a vertex that lies within the assigned volume - if any can be found. The magnitude of the error manipulation depends on the layer L_j the point was created for and is selected by the user for each L_j.
The vertex selection takes the distance between a point P_i^j and the vertex as well as the vertex error into account. This is done to favour vertices close to the generated points and simultaneously consider the vertex error to choose vertices that are more likely to remain in the mesh.
Given a point P_i^j in set G_j all vertices V_k within the volume around P_i^j are determined. Then the distance between V_k and P_i^j is calculated $d(V_k, P_i^j) = |p_i^j - v_k|$. The value $m(V_k, P_i^j)$ is calculated using the maximum side length l of the volume around P_i^j and the error $e(V_k)$.

$$m(V_k, P_i^j) = (l - d(V_k, P_i^j))^2 * e(V_k) \qquad (2)$$

The result of this calculation is a weighted vertex error. For a point P_i^j the vertex with the largest $m(V_K, P_i^j)$ within the corresponding volume is selected.
For each layer a minimum error is defined by the user. If a vertex is selected by a point P_i^j and the stored error $e(V)$ is smaller than the user selected value $e(L_j)$, the error is replaced by the user value.
Since the vertices are classified by comparison of the vertex error with a threshold, the user can select which set of vertices determined by this approach is to be used for the simplification.

4 PARALLEL HALF EDGE COLLAPSE

The result of the vertex classification is all the mesh's vertices being divided into two sets: R containing all vertices to be removed from the mesh and S with all vertices to remain.
The algorithm only considers half edge collapses on edges connecting a vertex in R to one in S. So the first step of the actual simplification is to determine a list of vertices C that contains all vertices $C_i \in R$ that have at least one neighbour $N \in S$.
For each C_i all possible half edge collapses are determined and one of them has to be selected and executed. The problem with half edge collapses lies in topological inconsistencies and mesh foldovers that may occur.
In order to allow for a fast implementation, the execution of half edge collapses on neighbouring vertices in parallel has to be carried out without any exchange of data. A simple approach to prevent negative effects on the mesh would be to compare triangle normals before and after the execution of the half edge collapse. A maximum angle between the triangle normal of a triangle before and after the collapse can be defined. If the angle is greater than a defined threshold, the collapse is considered invalid.
While this difference in angles works for isolated (half) edge collapses, it cannot be applied to the parallel half edge collapse, since two valid half edge collapses executed on neighbouring vertices may cause foldovers or topological inconsistencies. Figure 9 shows an example for this effect. Executing just the half edge collapse V_4, V_2 or V_3, V_1 creates a valid mesh. When both are applied in parallel however, the result is a folded triangle that has to be avoided. Here the original triangle V_3, V_4, V_5 and the modified triangle V_1, V_2, V_5 are overlapping.

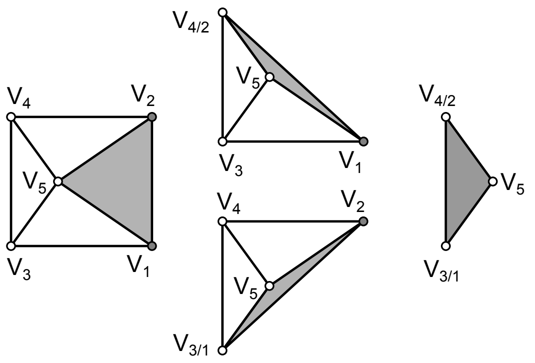

Figure 9: Two valid half edge collapses cause an inconsistency

The approach used in the algorithm presented here defines a set of boundaries for each vertex in C. This takes any neighbouring vertices to be removed in parallel into account and prevents foldovers and topological inconsistencies. It is also designed for view-dependent refinement and defines the boundaries using the view vector of the camera.
Given a vertex VR that is a candidate for removal, all triangles that contain VR have to be determined. For each triangle a number of planes is calculated that form the per-triangle boundaries and that are added to the set of boundaries $B(VR)$. Any potential half edge collapse has to be checked against the entire boundary $B(VR)$ to make sure that none of the triangles containing VR are affected by undesired effects. Since the result of any half edge collapse executed in parallel on neighbouring vertices is not known, the boundaries are defined in a way that prevents the aforementioned effects, no matter which half edge collapse a neighbouring vertex chooses (if any at all). This has the negative side effect of restricting the boundaries and possibly blocking combinations of half edge collapses that would not have negative effects on the mesh. In order to allow a higher degree of freedom when choosing collapses, the boundaries are created with regard to how many vertices of a triangle are to be removed in parallel. For a triangle, one of three sets of boundaries can be created, dependent if one, two or all three vertices of the triangle are current candidates for removal.

4.1 Test 1

The first case covers a triangle with only one vertex subjected to a half edge collapse. One edge of the triangle remains unchanged. While the aforementioned simple test that checks for large variations in triangle normals would suffice for this situation, the boundaries are created here as well to provide a uniform test and allow for a fast implementation.
Given the view-dependent approach of this algorithm a foldover or a topological inconsistency is created when the normal of a triangle before and after the manipulation changes orientation in relation to the view vector (the dot product between the view vector and the triangle normal changes sign). Therefore all boundaries are created taking the camera position E into account.

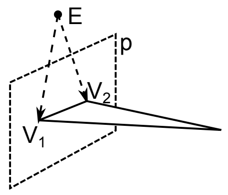

Figure 10: Boundary 1

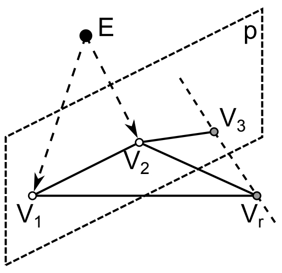

Figure 11: Boundary test 1

For a single vertex V_r to be removed in a triangle composed of V_r, V_1, V_2, a plane p is defined. Since V_1 and V_2 as well as the edge between them remain unchanged, removing V_r rotates the triangle normal around this edge. The plane p is defined using both V_1 and V_2, as well as the vectors $\overrightarrow{V_1 - E}$ and $\overrightarrow{V_2 - E}$.
Figure 10 shows an example for such a boundary plane with the camera looking down at the triangle from above. In order to test whether a half edge collapse is valid, the edge that is to be removed is intersected with the plane. If an intersection can be found, the endpoints of the edge lie on opposite sides of the plane and the half edge collapse is considered invalid.

Figure 11 shows an example for such a test. The edge V_r, V_3 is to be collapsed into V_3. While the vector $\overrightarrow{V_3 - V_r}$ has an intersection with the plane, the edge V_r, V_3 does not and the half edge collapse is considered valid.
This first boundary for an isolated half edge collapse always computes the correct result. As mentioned earlier, some boundaries can block valid half edge collapses in order to allow for parallel execution. While this is true for the following boundaries for two or three candidates in one triangle, it does not apply here.

4.2 Test 2

Given a triangle where two vertices are candidates for removal, the boundaries above are not valid anymore since they do not take the half edge collapse executed on neighbouring vertices into account. They would only prevent collapses that individually cause foldovers

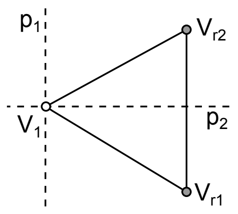

Figure 12: Boundary 2

or inconsistencies. See Figure 9 for an example.
Given a triangle with the vertices V_{r1}, V_{r2}, V_1 with V_{r1}, V_{r2} being candidates for removal, there is no edge that would remain unchanged and therefore no predictable rotational axis for the triangle normal. This results in a boundary consisting of two planes (Figure 12 shows an example from the viewpoint of the camera). They are designed to assign each removal candidate an individual area where a replacement position can be found. Any overlap between these two areas would enable a change of orientation between the triangle normal and the view vector.
The first plane p_1 is defined using the vectors $\overrightarrow{V_{r1} - V_{r2}}$ and $\overrightarrow{E - V_1}$ as well as the point V_1. It is therefore parallel to the edge between the two removal candidates and lies through the remaining vertex of the triangle.
The second plane p_2 is defined to intersect the edge between V_{r1} and V_{r2}. For this purpose a median was chosen. Plane p_2 uses the vectors $\overrightarrow{E - V_1}$ and $\overrightarrow{\frac{V_{r1} + V_{r2}}{2} - V_1}$ as well as the point V_1.
The test for a valid half edge collapse works similarly to the previous one. Given an edge that has to be tested for possible foldovers, it is intersected with both p_1 and p_2 to check if the collapse would cause in issue with this triangle. If an intersection between the edge and either of the planes can be found, it is considered invalid and will not be executed.
These boundaries can block valid half edge collapses and even combinations of valid half edge collapses for V_{r1} and V_{r2}. This is however accepted in order to allow for a parallel execution.

4.3 Test 3

The last case handles triangles where all 3 vertices V_{r1}, V_{r2}, V_{r3} are to be removed. Test 2 places both planes p_1 and p_2 to contain the remaining vertex V_1 of the triangle. Since all three vertices are a candidate for removal here as well, these boundaries are no longer valid.

Given a triangle V_{r1}, V_{r2}, V_{r3} test 3 creates two planes per removal candidate as well. All planes contain the centroid S of the triangle. Plane p_1 for vertex V_{r1} uses the vectors $\overrightarrow{V_{r2} - V_{r1}}$ and $\overrightarrow{E - S}$. Plane p_2 uses $\overrightarrow{V_{r3} - V_{r1}}$ and $\overrightarrow{E - S}$.
Both these planes are parallel to the edges that contain

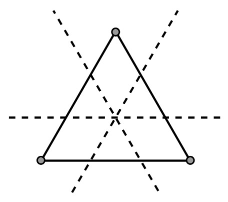

Figure 13: Boundary 3

the removal candidate and are again defined using the view vector of the camera. While for test 2 the two planes were identical for V_{r1} and V_{r2}, here two removal candidates only share a single plane. As a result two planes have to be constructed for each removal candidate, while three are necessary for the entire triangle (Figure 13 shows an example from the viewpoint of the triangle).
Testing of a half edge collapse is done as described in test 2. Given a removal candidate V_{r1} both necessary planes are constructed and an edge e is intersected with both p_1 and p_2. If either p_1 or p_2 intersect e, the edge is not valid for a half edge collapse as it would pose a risk of causing a mesh foldover or a topological inconsistency.

In order to find all valid half edge collapses for a vertex VR, first all triangles have to be found. Then for each triangle the number of removal candidates is determined and the corresponding set of boundaries is constructed. At this point, a set of planes P is on hand. Each edge VR, VS_j is then intersected with all $p_k \in P$. If an edge intersects any plane in P, it is considered invalid.

4.4 Half edge collapse selection

After a list of possible half edge collapses has been compiled, one of them has to be chosen and executed. We devised a slightly modified version of the quadric error metric presented by Garland and Heckbert[Gar98a] for this selection.
[Gar98a] presents an approach that uses pair collapse to replace a vertex pair V_1, V_2 with a single vertex V'. For this purpose it defines a vertex error $\triangle(V')$ in relation to V_1 and V_2. The algorithm then tries to find a position for V' that minimizes $\triangle(V')$. The authors however also suggest that a simple solution would be to use either V_1, V_2 or $\frac{V_1+V_2}{2}$ and choose the position with the lowest value of $\triangle(V')$.
The parallel half edge collapse makes use of this error metric. As suggested by the authors of [Gar98a], we use the replacement position with the lowest vertex error according to the quadric error metric. Given a removal candidate VR and a set of neighbours N where all $N_i \in S$ with edges VR, N_i form a valid half edge collapse, the error value according to the quadric error metric is calculated. Then the N_i with the lowest

error value $\triangle(N_i)$ is chosen and the half edge collapse performed.

Since half edge collapses are not executed isolatedly when using the parallel half edge collapse and several edges may be collapsed into a single vertex in parallel, the quadric error metric is not updated with each collapse as suggested by [Gar98a]. Instead the QEM is recalculated for each removal candidate during boundary computation, using the current intermediate result of the mesh.

5 DEADLOCK PREVENTION

While the boundaries described in the previous section allow for parallel execution of half edge collapses, they create the disadvantage of possibly blocking combinations of parallel half edge collapses that would not cause a foldover or a topological inconsistency. This problem cannot only occur for two neighbouring vertices. Several vertices could enter a state where they mutually block half edge collapses, effectively causing an area of the mesh that cannot be simplified even though for each vertex one or more half edge collapses could be performed individually.

If a vertex is to be removed, but all possible half edge collapses are being blocked by the boundaries, an additional computation has to be performed. For each edge e_i that was tested against the boundaries, the blocking planes have to be determined.

If all planes that blocked the half edge collapses are constructed by test 1, the current topology of the mesh around the removal candidate VR does not allow any half edge collapse without causing either a foldover or a topological inconsistency. In this first case the algorithm aborts the search for a valid half edge collapse for VR. The vertex remains in the mesh and is part of the simplified result.

The second case needs a more complex handling. Here one or more planes blocking a collapse are constructed from a triangle with two or three vertices to be removed. As described above, these boundaries could prevent valid half edge collapses from being executed. To avoid that, the algorithm always computes a separate set of planes for VR, using only test 1 for all triangles, effectively ignoring possible parallel half edge collapses. This leads to two results for each half edge collapse. The first one taking parallel half edge collapses into account and potentially blocking valid collapses, the second one to check if the topology allows for a removal of VR at all. If the second result fails as well as the first one, no half edge collapse can be safely performed for VR and the vertex remains in the mesh and no further attempts of removal will be taken.

If at least one half edge collapse is allowed by the second result, the vertex remains a candidate for removal for the next iteration i_{n+1}.

During iteration i_{n+1} the status of all neighbouring vertices of VR is compared to that of the previous iteration i_n. If none of the removal candidates have been subjected to a half edge collapse, a deadlock is assumed. To resolve this the stored vertex error $e(VR)$ is compared to the one of the neighbouring removal candidates. Unless $e(VR)$ is greater than the vertex error of all neighbouring removal candidates, VR is marked as "to be ignored". Ignored vertices are skipped during the selection of removal candidates and although neighbouring vertices do not consider them valid targets for a possible half edge collapse, they are not considered as "to be removed" and therefore do not cause the application of test 2 or 3. This approach allows neighbouring vertices to be subjected to a half edge collapse and so a possible deadlock can be resolved.

Once a neighbouring vertex is reclassified or removed by a half edge collapse, the ignore flag is deleted and VR can be selected as a candidate for removal again.

6 RECLASSIFICATION

As described in section 3, the initial classification performs calculations based on neighbouring vertices but does not consider the possible removal of those. For this reason an additional step is performed in between iterations.

After a vertex VR_i is removed by a half edge collapse, the vertex error of all neighbours $N_j \in R$ of VR_i is invalidated due to the changes to the mesh. A new vertex error that is used to validate the classification is then calculated for all N_j.

After an iteration has been completed, the algorithm determines a list $C = \{C_0, C_1, ..., C_n\}$ containing all vertices $VR \in R$ that have at least one neighbour in S. Since half edge collapses change the edges, this list needs to be regenerated after each iteration. Then the vertex error $e(C_i)$ is updated taking these changed edges into account. If the new vertex error $e(C_i)$ is greater than a certain threshold, C_i is reclassified, removed from R and C and added to S. It is therefore no longer to be removed from the mesh.

This operation may be executed very often during runtime. In addition this error metric for reclassification should not deviate from the metric used in the initial vertex analysis to prevent a potential reclassification of a majority of vertices in R. For those reasons a variation of the metric presented in section 3 is applied here.

The initial classification uses the average of distances between the tangent plane of a vertex V that is determined by the stored vertex normal and not changed during the simplification and its neighbours N. At this point it has to be taken into account that one or more neighbouring vertices may be marked for removal and therefore do not provide useful data, since

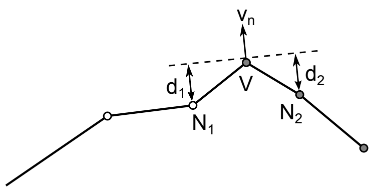

Figure 14: Reclassifying vertices

they may not be part of the final, simplified mesh.
The metric used here still relies on the distance between the tangent plane of V and N_i, it does not rely on the average though and calculates the maximum value instead.

Figure 14 shows an example for a possible reclassification. Given that all vertices to the left of V are elements of S, while the ones to the right are contained in R, only the neighbour N_1 is included in the error metric for the recalculation. d_2 calculated using $N_2 \in R$ is omitted.
Since this newly calculated error value differs from the initial classification, an adapted threshold has to be defined for the reclassification. If the newly calculated error value is greater than the threshold, the vertex is reclassified as described above, otherwise it remains in R and the next iteration tries to find a possible half edge collapse to remove it from the mesh.

7 RESULTS

For early testing a GPU accelerated implementation of the parallel half edge collapse using CUDA on a Nvidia Geforce GTX 670 GPU was used.
For testing purposes, the model of the Stanford Bunny was simplified. The original version of this mesh is made up of 35 947 vertices forming 69 451 triangles.
Three different simplifications of the original mesh were calculated by adapting the threshold for the initial classification. For these early results the main focus was the overall processing time for the simplification, the resulting triangle count of the simplified mesh and the number of iterations necessary until all vertices marked for removal had been processed.
Figure 15 shows the comparison between the wireframe of the three simplified models derived from the original.
The three cases resulted in models made from 48 831, 29 014 and 17 565 triangles respectively. The first case was completed in 1.9 ms, while cases 2 and 3 required 3.3 and 4.4 ms.
As to be expected, the necessary number of iterations increases as more vertices are marked for removal and hence removed from the mesh. This can be explained by the restriction that only edges between vertices in R and S are considered possible half edge collapses. Marking more vertices for removal, therefore reducing the amount of vertices in S, reduces the number of possible half edge collapses and results in additional

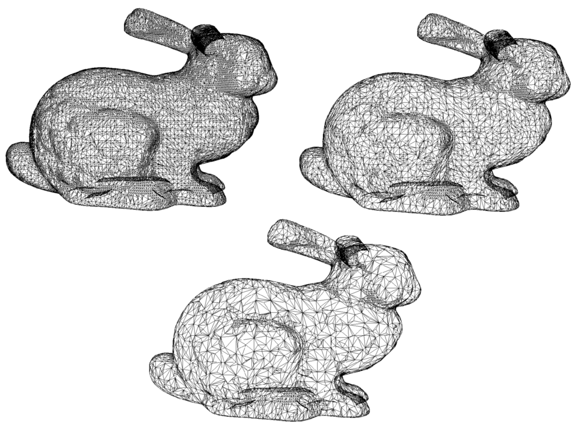

Figure 15: Simplification results

iterations.
While the parallel half edge collapses managed to remove all marked vertices in 2 iterations for case 1, 5 and 8 are respectively needed in the other two cases, causing the increase in processing time.
Another potentially limiting factor that could cause additional iterations are deadlocks. Since these are not resolved until the following iteration, a small number of mutually locking vertices can have a large impact on the runtime. For runtime critical applications of the parallel half edge collapse a threshold for a minimum amount of vertices in R should be considered. Since the half edge collapse and therefore the parallel half edge collapse preserves manifold connectivity at every step, the simplification can be safely aborted after each iteration.

These early tests have also shown that the use of the artificially introduced vertex error can have a great impact on runtime. While there was no difference in runtime measurements for the first test case, there is a clearly visible difference for test cases with a higher number of necessary iterations. Executing test case 3 without the modified vertex error increased the number of iterations by a factor of 4, simultaneously causing a great increase in overall runtime for the simplification as well.

7.1 Future work

The parallel half edge collapse can provide fast simplification with good results. A main bottleneck of the approach can be posed by the restriction, that only edges between vertices in R and S are considered a possible half edge collapse. A majority of the vertices of a mesh being marked for removal or a disadvantageous topology can cause a reduction in parallel half edge collapses and force the execution of additional iterations. Future work can focus on several approaches to resolve these issues.

An improved vertex analysis/classification could provide a way to improve parallelism and reduce processing time. A more precise initial classification could provide additional removal candidates in each iteration, therefore reducing the number of necessary iterations and improving performance.
The difficulty caused by this approach lies within the prediction of chosen half edge collapses. Preselecting a precise set R without adapting to the outcome of simplification operations has the potential to result in lower quality simplifications than adapting after each iteration.
Another significant improvement could be made by allowing vertices in R that do not have a neighbour in S to be collapsed rather than having to wait for one or more iterations until a neighbouring vertex $N \in S$ is provided. This would also increase the freedom of the simplification, possibly offering a greater number of half edge collapses per vertex to choose from and improve the overall result of the simplification.

8 CONCLUSION

The parallel half edge collapse is designed to provide fast simplification with good results. The emphasis on lack of communication between half edge collapses as well as the focus on fast error metrics potentially allows for short processing times, enabling the parallel half edge collapse for real time applications using view-dependent simplification and thus further improve the result of the simplification.
The parallel design enables an implementation on modern GPUs, further reducing the time needed to calculate the simplified meshes.

On the other hand however the usage of the half edge collapse instead of the more generic edge collapse can prove to be a limiting factor. The possibly limited number of selectable half edge collapses and the focus on error metrics designed for fast computation and updating can impact the result. This can have a negative influence on the quality of the overall result of the simplification, which is a trade-off that has to be accepted for real time use.

9 REFERENCES

[Cla76a] Clark, J. H. Hierarchical geometric models for visible surface algorithms, Com. of ACM 19, No. 10, pp.547-554, 1976

[Ros92a] Rossignac, J., and Borrell, P. Multi-resolution 3D Approximations for Rendering Complex Scenes, Modeling of Computer Graphics: Methods and Applications, pp.455-465, 1992

[Sch03a] Schaefer, S., and Warren., J. Adaptive vertex clustering using octrees, Proceedings of SIAM Geometric Design and Computing 2003, Vol. 2, pp.491-500, 2003

[Low97a] Low, K.-L., and Tan, T., S., Model simplification using vertex-clustering, SI3D Proceedings 1997, pp.75-ff., 1997

[Sch92a] Schroeder, W., J., Zarge, J., A., and Lorensen, W., E. Decimation of triangle meshes, ACM SIGGRAPH Computer Graphics Vol. 26, No. 2, pp.65-70, 1992

[Hop93a] Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., A., and Stuetzle, W. Mesh optimization, ACM SIGGRAPH Proceedings 1993, pp.19-26, 1993

[Hu09a] Hu, L., Sander, P., V., and Hoppe, H. Parallel view-dependent refinement of progressive meshes, I3D 2009 Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games, pp.169-176, 2009

[DeC07a] DeCoro, C., and Tatarchuk, N. Real-time mesh simplification using the GPU, I3D 2007 Proceedings of the 2007 Symposium on Interactive 3D Graphics Vol. 2007, pp.161-166, 2007

[Hop96a] Hoppe, H. Progressive meshes, ACM SIGGRAPH 1996 Proceedings, pp.99-108, 1996

[Lue02a] Luebke, D., Watson, B., Cohen, J., D., Reddy, M., and Varshney, A. Level of Detail for 3D Graphics, 2002

[Xia97a] Xia, J., C., El-Sana, J., and Varshney, A. Adaptive real-time level-of-detail-based rendering for polygonal models, IEEE Transactions on Visualization and Computer Graphics Vol. 3, No. 2, pp.171-187, 1997

[Gar98a] Garland, M., and Heckbert, P., S. Surface simplification using quadric error metrics, SIGGRAPH Proceedings 1997, pp.209-216, 1997

[Hu10a] Hu, L., Sander, P., and Hoppe, H. Parallel view-dependent level of detail control, IEEE Transactions on Visualization and Computer Graphics Vol. 16, No. 5, pp.718-728, 2010

[Pap15a] Papageorgiou, A., and Platis, N. Triangular mesh simplification on the GPU, The Visual Computer: International Journal of Computer Graphics Vol. 31, Issue 2, pp.235-244, 2015