Eurographics Symposium on Parallel Graphics and Visualization (2014)
M. Amor Lopez and M. Hadwiger (Editors)

Parallel Progressive Mesh Editing

Evgenij Derzapf[1], Nico Grund[1] and Michael Guthe[2]

[1]Sirona Dental Systems GmbH, Bensheim, Germany
[2]Universität Bayreuth, Visual Computing, Germany

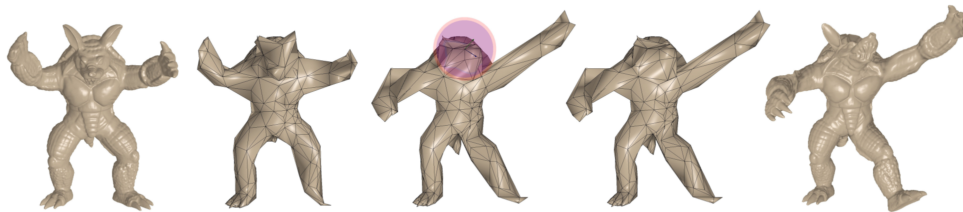

Figure 1: *Editing of the Armadillo progressive mesh. Note how the fine geometric details are preserved by the local encoding of the split operations.*

Abstract

Highly detailed models are commonly used in computer games and other interactive rendering applications. Intuitive editing methods are thus also required in addition to rendering algorithms. Progressive meshes are often employed to improve the rendering performance by reducing the number of rasterized triangles. The classical work flow is to generate a model and then use simplification algorithms to construct the progressive mesh. Thus the whole simplification has to be performed again after editing the model. This does not only require additional processing time but also hinders animations of progressive meshes.

Based on this observation we propose a real-time parallel multi resolution modeling algorithm for progressive meshes. It can be used for real-time editing and animation of complex progressive meshes. Due to the progressive representation we can intuitively modify the overall shape or small scale details. To quickly generate a progressive mesh from a complex triangle model we also propose a massively parallel simplification algorithm that generates all required data structures within a few seconds.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Curve, surface, solid, and object representations I.3.1 [Computer Graphics]: Hardware architecture—Parallel processing

Highly detailed geometric models are very popular in interactive applications such as computer games. These models are usually represented as triangle meshes. To render several of these models at real-time frame rates, level-of-detail (LOD) techniques are commonly used. Mesh editing is used to correct errors or animate the models. In the traditional work flow simplification algorithms are used afterwards to generate the LODs. This requires a re-simplification after editing and restricts animations to space deformations. To solve this problem, multi resolution modeling approaches were proposed that use different resolutions. These subdivision meshes are comparable to static LODs that are simply a set of polygon meshes. Although multi resolution modeling is restricted to geometric deformations, it is still a very valuable tool of the modeling pipeline.

In contrast to the static LODs used for previous multi resolution modeling algorithms, dynamic LODs store a coarse base mesh and a sequence of refinement operations. The most common data structure used in this context are progressive meshes. Using our approach it is possible to edit the mesh at a lower quality while modifications are automatically propagated to finer resolutions. Thus only a few vertices need to be modified, to achieve large scale or small scale deformations, depending on the current LOD. In contrast to progressive meshes, multi resolution modeling re-

© The Eurographics Association 2014.

quires a local geometry encoding to allow editing and the propagation of the geometric modifications. We use a parallel simplification algorithm to generate the progressive mesh and to propagate changes to coarser resolutions. The main contributions of our approach are:

- Real-time editing of large progressive meshes.
- Complete progressive mesh generation on the GPU.

1. Related Work

Mesh editing has been an active field of research over the last three decades. Early approaches focused on editing smooth surfaces while later ones tried to preserve local properties of the mesh. The algorithm of Welch and Witkin [WW94] is based on arbitrary triangle meshes and allows free-form shape design. Taubin [Tau95] later optimized this approach and improved the efficiency. Editing of smooth surfaces is still an active field of research [LF03].

1.1. Multi Resolution Modeling

Meshes containing geometric details require special techniques to preserve these during editing. Commonly, multi resolution representations are used. The details are stored relative to a coarser model for one or more resolutions [FB88, ZSS97, KCVS98, KVS99, GSS99, Gar99]. The user can edit the mesh on a lower quality level and the changes are automatically propagated to the higher levels. This way only a few vertices need to be edited to achieve large changes. Using so-called Laplacian or differential coordinates [LSCo*04, SCOL*04], the fine-scale surface can be reconstructed by solving a linear system containing the modified differential coordinates. To manipulate the mesh, Sorkine et al. [SCOL*04] proposed to use interactive free-form deformation in a region of influence (ROI) or to integrate detail of one surface into another. Marinov et al. [MBK07] mapped a multi resolution deformation framework to the GPU. As the displacement vectors are encoded independently for each component, visible artifacts in highly deformed regions can occur. A combination of this approach with multi resolution meshes was proposed by Manson and Schaefer [MS11] but they only allow to edit a fixed coarse resolution mesh and then transfer the changes to the full resolution and not the other way round. Other approaches include using lower resolution point based models [BSS07] or deforming the surrounding space instead of the model itself as e.g. in [ZSGS12]. The edit granularity is however limited to the resolution of the auxillary representation.

1.2. Mesh Simplification

Mesh simplification is one of the fundamental techniques for real-time rendering of polygonal models. A detailed review of simplification algorithms is given by Luebke [Lue01]. As we focus on modeling, we only discuss those methods that support local modifications.

Garland and Heckbert [GH97] as well as Popović and Hoppe [PH97] introduced the vertex pair contraction. This approach has become the most common technique for the simplification of triangle meshes. The contraction operation is combined with the introduced quadric error metric. It allows a flexible control over the geometric error and can be used to calculate optimal vertex positions. Later Garland and Heckbert extended their approach to handle an arbitrary number of vertex attributes [GH98]. While the generated approximations are superior to vertex clustering at the same number of triangles, the simplification performance is significantly lower. On the other hand, all required levels can be generated using a single simplification sequence from the original model to the coarsest level. Grund et al. [GDG11] proposed a parallel GPU implementation of the quadric error simplification. On a customer level graphics card, it can generate a set of LODs for a model containing over 4 million faces in less than a second.

Pajarola and Rossignac [PR00] introduced compressed progressive meshes that provide a very compact coding but a view-dependent adaption is not possible. They also use a differential coding of vertex attributes but store the offsets in a global coordinate system which is not suitable for editing. Hu et al. [HSH09] proposed a parallel adaption algorithm for progressive meshes. They introduced a relatively compact explicit dependency structure that allows to group vertex splits and half-edge collapses into parallel steps. A more compact progressive meshes data structure for parallel adaption was proposed by Derzapf et al. [DMG10a, DMG10b]. It was later generalized to non-manifold meshes using a compressed progressive mesh data structure [DG12]. In our approach we modify and extend this data structure for real-time editing. A slightly different strategy was used by Maglo et al. [MGH13] where a spatial subdivision is used to split the model into sub-meshes that can be processed independently.

2. Overview

Traditionally, multi resolution editing performs global deformations by decomposing the geometry of a highly-detailed model into a coarse base surface and reconstructing the applied modifications using a displacement in normal direction. In contrast to this we use edge split operations of a progressive mesh to propagate the transformations to finer resolutions. Therefore, we propose an appropriate data structure that contains relevant informations about the connectivity and vertex attributes. The operations are stored in a tree structure that is generated by a parallel simplification of the original mesh. Based on the operation tree, the model can be continuously adapted using parallel vertex split and edge collapse operations. Our proposed algorithm can be divided in two main phases:

1. Simplification of the model and generation of the progressive mesh operation tree.
2. Multi resolution editing of the progressive mesh.

© The Eurographics Association 2014.

To preserve the connectivity during the propagation of the modifications, the local ordering of operations needs to be preserved. This implies for example that a collapse is only allowed, if no other vertex in any adjacent triangle is collapsed or split. The current resolution can be edited using a handle and a euclidean distance based region of influence (ROI). The handle can be translated and rotated and the transformation is applied to the whole ROI (see Figure 1).

2.1. Progressive Mesh

We simplify the original model by collapsing all non-conflict edges in parallel. The original model can then be reconstructed by performing a sequence of parallel vertex split operations. Figure 2 shows the principle of an edge collapse operation col_v and the corresponding split operation spl_v. By applying col_v to an edge defined by the vertices v_t and v_u it is contracted into the vertex v. The new vertex v is computed based on the neighboring triangles. The adjacent faces f_l and f_r of v_t and v_u degenerate and are removed. Since spl_v represents the inverse operation of col_v, the faces f_l and f_r are generated by splitting the vertex v into v_t and v_u. Additionally, an update of the connectivity between the vertices v_t and v_u and the adjacent faces is performed.

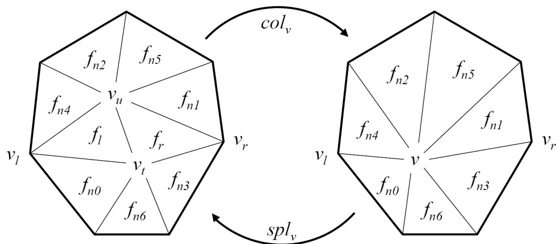

Figure 2: *Edge collapse and vertex split operation.*

2.2. Operation Coding

The data structure used to encode the operations is based on the one proposed by Derzapf and Guthe [DG12]. In our context, the main advantage is that non-manifold meshes are supported by storing the successive topology modifications within the triangles instead of the operations. For this we need to generate so-called final vertex IDs (FVIDs) by enumerating the vertices after computing all edge collapses and storing them in the data structure. This way all possible topology modifications are encoded within the vertex indices of the faces. The faces are then stored in the split operation where they are generated by storing their vertex $\text{FVID}_{0..2}$ for the finest resolution.

In summary, each operation encodes the attributes of v_t and v_u, the refinement criteria, the generated faces including later modifications and the subsequent operations. In contrast to [DG12] we do not compress the operations but keep their uncompressed form to allow editing. The local ordering is preserved by storing the maximum of the simplification error and the errors stored for the neighbor vertices plus a small epsilon. This way we only need to check the errors of neighboring vertices to enforce a strict local ordering of split and collapse operations.

2.3. Local and Global Attributes

In compression approaches, the vertex offsets are often encoded in the local coordinate system of the split vertex. While this improves compression rates, a transformation of the split vertex directly applies to all its decendents. To support a smooth propagation of the transformation to neighboring split vertices, the local coordinate system is averaged from all adjacent vertices (see Figure 3). Local position, normal and tangent (P_i, N_i, T_i) are computed as weighted average of v and all vertices adjacent to v_t or v_u. Then the position and normal offsets (P_Δ, N_Δ) are encoded in the local coordinate system spanned by N_i and T_i. Note that only one degree of freedom remains for the tangent and it is thus encoded as rotation about N.

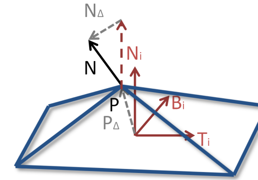

Figure 3: *Vertex attributes encoded relative to the local coordinate system interpolated from neighboring vertices.*

3. Progressive Mesh Generation

The progressive mesh generation is based on the parallel edge collapse simplification algorithm of Grund et al. [GDG11]. Figure 4 gives an overview of the extensions and modifications necessary to construct a progressive mesh that allows both parallel adaption and real-time editing.

After loading the initial mesh, the attributes and indices are transferred to the GPU and stored in a *vertex buffer* and *index buffer*. Then the edge data structure is filled as in the original simplification algorithm. Additionally, we store the edge indices for each face since we require them later to guarantee a fixed collapse neighborhood. In contrast to the previous simplification algorithm, we use memoryless simplification [LT98] which results in computing the vertex quadrics inside the simplification loop.

All adjacent faces and boundary edge quadrics are accumulated to calculate the vertex quadrics. The quadric error optimization is identical to the original method with the exception that the stored simplification error ε_v of vertex v is:

$$\varepsilon_v = \max\left(\varepsilon_{quadric}, (1+\varepsilon_{float}) \max_{i \in neighbors(v)} \varepsilon_i^s\right),$$

© The Eurographics Association 2014.

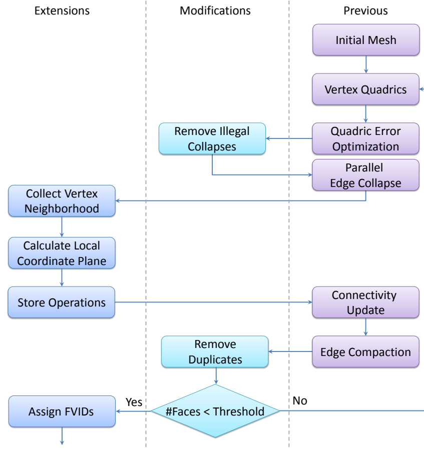

Figure 4: *Progressive mesh generation including the extensions (left) and modifications (middle) of the previous simplification algorithm (right).*

where $\varepsilon_{quadric}$ is the quadric error (i.e. the sum of the squared distances to the adjacent faces' planes), ε_i^s is the previous error stored for vertex i, and the multiplication with $1+\varepsilon_{float}$ assures that the error stored for collapsing v_t and v_u to v is larger than the error of previous neighboring collapses. This property is later used during adaption and modeling to guarantee a fixed neighborhood for each operation. This also necessitates a modification of the overlapping collapse removal step that requires the edge references stored for each face. Fixing the neighborhood means that no two vertices sharing a common edge can be collapsed in parallel. For static LODs, preventing a collapse of two adjacent edges was sufficient. In addition, the collapse can also not be performed, if v_l or v_r are not connected to v_u or v_t by another triangle despite f_l and f_r. This condition is necessary because otherwise v_l or v_r will be missing in the neighborhood of v_u and v_t when performing the vertex split.

After performing the collapses, the operations can be stored in the progressive mesh data structure. This is done by first collecting the neighborhoods for all operations and then encoding the attribute offsets in the local coordinate systems. Finally, we update the connectivity and remove the collapsed and duplicate edges. The edge references stored in the faces are also updated during the compaction of the edge buffer.

When the number of faces falls below a specified threshold, the simplification stops and the FVIDs are assigned to the vertices and operations. Note, that not all buffers are required during the complete algorithm and can be freed as soon as they are not used anymore.

4. Editing

Editing is based on modifying the attribute offsets of vertices encoded in split operations or the attributes of base mesh vertices. If a vertex is edited, its global attributes are changed and need to be mapped to modified local attributes. Modeling is based on the framework of Bendels et al [BKS03] that allows translations and rotations. Instead of using geodesic distances, we only use the Euclidian distance due to performance reasons. The editing is transferred to finer levels by encoding the split offsets in the local coordinate system of the neighbor vertices. For the coarse levels, the global attributes need to be recomputed after editing. Thus we need to constantly convert between local and global attributes.

4.1. Local and Global Attributes

The interpolated local coordinate systems of the neighbor vertices are used as reference coordinate system. As only the collapse target and its one ring are available, we can only use these. Therefore, we need to guarantee that the neighbor vertices are the same for a split and its inverse collapse operation. This is assured by the definition of the local ordering of the operations (see Section 2.2). Figure 5 shows the neighborhood used as reference for vertex v_t. Note that vertex v is weighted by a factor of two when computing the refernce coordinate system, as it is closest to v_t.

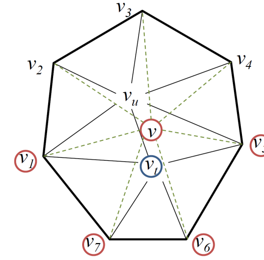

Figure 5: *Vertices in the split neighborhood of v_t. The ones used to interpolate the reference coordinate system of v_t are marked in red.*

Given the attribute offsets in the global coordinate system and the local coordinate system of the neighborhood, the local offsets (P_l, N_l and α) are calculated as:

$$\begin{aligned} P_l &= (T_i \quad B_i \quad N_i)^t P_\Delta \\ N_l &= (T_i \quad B_i \quad N_i)^t N_\Delta \\ T_\perp &= normalize(T_i - N(N \cdot T_i)) \\ \alpha &= \arctan \frac{T_\perp \cdot (N \times T)}{T_\perp \cdot T}, \end{aligned}$$

where P, N and T are position, normal and tangent of the vertex v_t or v_u, $B_i = N_i \times T_i$ the interpolated bitangent and α the rotation of T about the normal of the local coordinate system. The required position and normal offsets are:

$$\begin{aligned} P_\Delta &= P - P_i \\ N_\Delta &= N - N_i. \end{aligned}$$

On the other hand, the global attributes of the vertex can be

© The Eurographics Association 2014.

calculated as:

$$
\begin{aligned}
P_\Delta &= (T_i \quad B_i \quad N_i)\, P_l \\
N_\Delta &= (T_i \quad B_i \quad N_i)\, N_l \\
P &= P_i + P_\Delta \\
N &= N_i + N_\Delta \\
T &= T_i \cos(\alpha) + (N_l \times T_i)\sin(\alpha).
\end{aligned}
$$

If no tangent vectors are specified for the mesh, we initially assign default tangent vectors (orthogonal to $(001)^t$) to the base mesh vertices and set the local offsets α to zero.

4.2. Edit Propagation

After editing the changes need to be propagated to the coarser meshes. As noted earlier, the propagation to finer resolutions is handled automatically during the split operations. During editing, all modified vertices are marked by setting their modified flag to one. This value means that the local coordinate system and the global attributes where changed. Then the edited vertices are used for the calculation of the interpolated coordinate system. By adding the attribute offsets, which are encoded in the local coordinate system, finer details can be rebuilt. For the propagation to coarser resolutions, the local coordinate systems of the modified vertices need to be recalculated during the collapses. As we use memoryless simplification and guarantee a fixed neighborhood for every collapse, we can simply recompute the edge quadrics. Then the quadric is again minimized to calculate the new vertex attributes in the coarser mesh. The new simplification error is also calculated as discussed in Section 3. Due to storing the attribute offsets in the local coordinate system of the neighborhood, we need to recompute the collapse for every vertex that is adjacent to a modified one. This can be checked by simply traversing all triangles in parallel and setting the modified flag of each triangle vertex to two if it is zero and at least one of the others in the triangle has a modified flag of one. A modified value of two now means that the vertex has a new local coordinate system but still its old global attributes. Note that we do not need to prevent race conditions here as we only change the flag from zero to two. The modification flag is then set for all vertices for which the collapse was recomputed to transfer the editing up to the base mesh. This implies that the progressive mesh has to be collapsed down to the base mesh before saving.

5. Adaption Algorithm

The adaption algorithm is based on the progressive mesh rendering algorithm of Derzapf and Guthe [DG12]. The algorithm is subdivided into several consecutive steps to implement the adaption on massively parallel hardware. In the first step we update the state of the vertices as in the original algorithm, but a global simplification error is used instead of view dependent refinement criteria during editing. Additionally, we remove overlapping split and collapse operations to guarantee the fixed neighborhood. Then all edge collapses are performed in parallel. Another modification is that we recalculate the local coordinate system of the corresponding split operation if the vertex v was marked as modified (see Section 4.2). The memory management required before splitting is unmodified. For the split operations we additionally need to calculate the global attributes of the vertex (see Section 4.1). Finally, the index update and buffer compaction of the original algorithm are performed.

The dynamic data structures required for adaption and editing are listed in Table 1. The vertex buffer containing position and attributes and the index buffer storing the connectivity of the adapted mesh are required for rendering and thus separated from all other data. The modification flag is used to mark all vertices for which the split operations need to be updated. In the following we discuss the extensions of the progressive meshes adaption algorithm in detail.

buffers	elements	bytes per entry
active faces	index VBO	12
	FVIDs	12
active vertices	vertex VBO (×2)	$8k$
	vertex ID (×2)	8
	modified flag (×2)	2
	collapse target (×2)	8
	next split & collapse (×2)	16
temporary	vertex count	4
	face count	4
	vertex prefix sum	4
	face prefix sum	4
	vertex quadric	$2k^2+6k+4$

Table 1: *Elements of the dynamic data structure, where k is the number attributes and additions are marked bold.*

5.1. Illegal Operation Removal

After updating the state of all active vertices, we remove splits and collapses that cannot currently be performed. The algorithm traverses all triangles and checks the vertex states. For the split operations, only the vertex with the highest simplification error can be split in each face f. In addition, no vertex of f can be collapsed, if any other is marked for splitting. Finally, a collapse operation can only be performed, if its simplification error is the lowest in the triangle. The complete removal of illegal operations is shown in Algorithm 1.

```
foreach face f in parallel do
    if any_vertex_marked(f, split)
        simplification_error_max = get_max_split_error(f)
        unmark_dependent_splits(f, error_max)
    if any_vertex_marked(f, collapse)
        simplification_error_min = get_min_collapse_error(f)
        unmark_illegal_collapses(f, error_min)
```

Algorithm 1: Parallel removal of illegal operations.

© The Eurographics Association 2014.

	Original model			Simplification				Rendering			
model	v_{max}	f_{max}	IFS	PM	memory	time (s)	Op/s	spl. Op/s	coll. Op/s	upd. Op/s	fps
Horse	48,485	96,966	2.7 MB	5.4 MB	41.3 MB	0.3	162k	595k	443k	384k	>60
Armadillo	172,974	345,944	7.9 MB	19.1 MB	147.3 MB	0.6	288k	1181k	810k	631k	>60
St. Dragon	437,645	871,414	19.9 MB	48.3 MB	372.2 MB	1.4	313k	1201k	882k	674k	30–60
Welsh Dragon	1,105,352	2,210,673	50.5 MB	122.3 MB	941.6 MB	3.2	346k	986k	381k	336k	11–60
Dragon	3,609,455	7,218,906	165.2 MB	399.0 MB	1872.6 MB	10.4	347k	760k	379k	335k	3–60

Table 2: *Models used for evaluation with memory consumption for the progressive mesh (PM), simplification operations per second (Op/s), split (spl.), collapse (coll.) and update (upd.) performance.*

5.2. Parallel Edge Collapses

When no neighboring vertices of the edge are marked, the collapse operation simply moves vertex v_t to its old position v. Otherwise, the weigthed average of the adjacent vertex attributes is required. We calculate this by counting the number of adjacent vertices and accumulating their attributes. If the vertex was marked as modified, the local coordinate system of the next split operation is re-calculated and the modified flag is propagated to the parent vertex and its neighborhood. Removing vertex v_u and the degenerated faces is handled in later stages. Algorithm 2 shows the parallel edge collapses including operation update after editing and preparing removal of the collapsed vertices and faces.

```
foreach face f in parallel do
    v1, v2, v3 = get_vertices(f)
    atomic_add_acc_adjacent_sum(v1, v2, v3)
    atomic_add_adjacent_number(v1, v2, v3, 2)
    q = face_quadric(f)
    atomic_add_vertex_quadric(v1, v2, v3, q)
foreach vertex v_u in parallel do
    v = get_target(v_u)
    if marked(v, modified) || marked(v_u, modified)
        LCS_VT = optimize_quadric(v)
        LCS_VU = optimize_quadric(v_u)
        update_split(v, LCS_VT)
        update_split(v_u, LCS_VU)
    else
        restore_attributes(v)
    collapse_vertices(v, v_u)
```

Algorithm 2: Parallel edge collapse algorithm.

5.3. Parallel Vertex Splits

As in Derzapf and Guthe [DG12] we first compact the split operations to improve thread utilization. Then the global position of v_t and v_u are calculated from the weighted average of the local offsets stored in the operation. Again, the weighted average of the adjacent vertex attributes is required (see Section 2.3). For splitting, the new faces are first added to the mesh. Then the global attributes are calculated. Algorithm 3 shows the parallel vertex split.

```
compact(splits)
foreach split vertex v in parallel do
    v_u = v + 1
    split_vertex(v, v_u)
    append_faces(v, face_sum[v])
foreach face f in parallel do
    v1, v2, v3 = get_vertices(f)
    atomic_add_acc_adjacent_sum(v1, v2, v3)
    atomic_add_adjacent_number(v1, v2, v3, 2)
foreach split vertex v in parallel do
    LCS_VT = acc_adjacent_sum(v) / adjacent_number(v_t)
    LCS_VU = acc_adjacent_sum(v_u) / adjacent_number(v_u)
    calc_attributes(v, v_u, LCS_VT, LCS_VU)
```

Algorithm 3: Parallel vertex split algorithm.

6. Results

Our test system consists of a 3.333 GHz Intel Core i7-980X CPU with 6 GB DDR3-1333 main memory and an NVIDIA GTX580 (841/4204MHz). We used CUDA to implement the parallel simplification and editing and use OpenGL for rendering. Table 2 gives an overview of the generated progressive meshes and the runtime performance during modeling. All models use position and normal as vertex attributes ($k = 6$). Note that increasing k needs more memory The original meshes contain v_{max} vertices and f_{max} faces. As the progressive meshes are uncompressed, they require approximately twice the memory than the original models (IFS). The maximum memory consumption is between 16 times (for small models) and 11 times (for larger ones) larger than that of the original model. This maximum is reached at the beginning of the simplification. During rendering the maximum is approximately $\frac{2}{3}$ since the edge data structures are not required anymore. Similarly to the simplification, the maximum is reached when the model is refined to full resolution. Compared to the approach of Grund et al. [GDG11] the preprocessing performance is by a factor of 5.8 lower. The two main reasons for this are the larger neigborhood that reduces the number of parallel collapses by a factor of two and the generation of the progressive mesh data structure. In addition, the memoryless simplification is computationally more expensive than the normal quadric error simplification used in that algorithm. The adaption performance is even by a factor of 9.6 lower than that of Derzapf and Guthe [DG12].

© The Eurographics Association 2014.

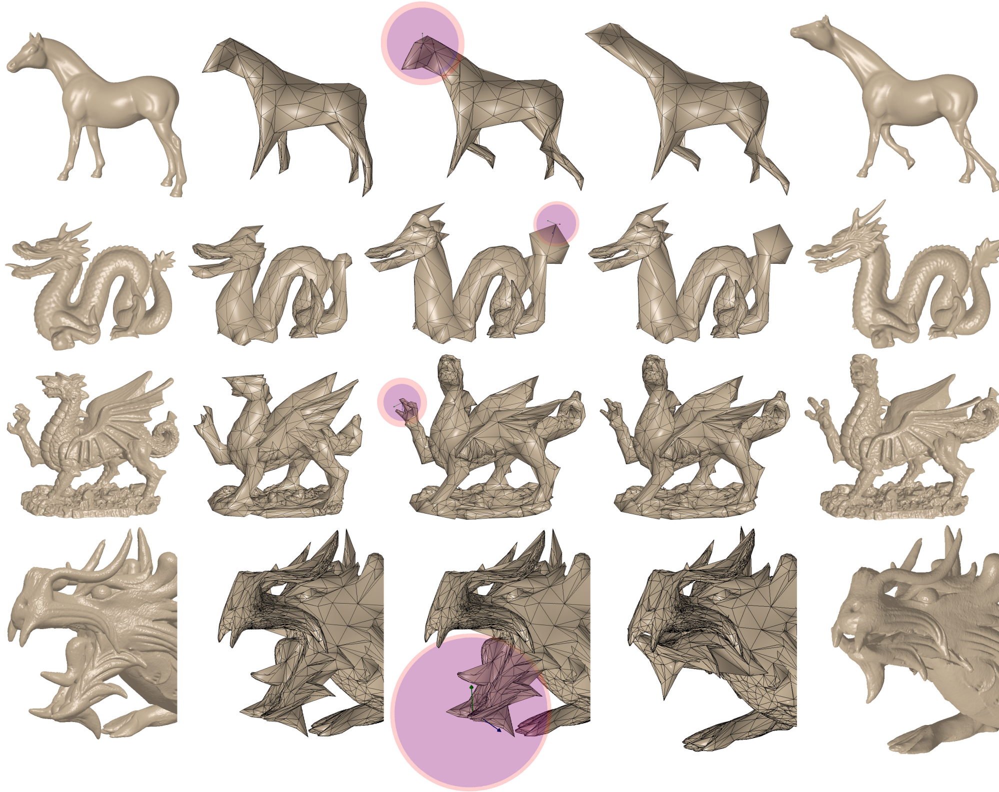

Figure 6: *Several progressive meshes used in our evaluation. From left to right: original models, three frames captured during editing where the purple spheres show the ROIs, and the final proressive meshes refined to full resolution.*

This is partially again due to the fact that we introduce neighborhood dependencies which reduce the number of parallel operations by a factor of approximately 6. On the other hand, the uncompressed data require more memory bandwidth and the transformation from local to global coordintes also slows down the adaption. The reduction of the number of operations and thus decreasing performance for larger models is due to the fact that fewer operations were performed in parallel. The reason for this is that these models were not pre-simplified and therefore contained many coplanar faces. In our current implementation this blocks many collapses as only those are performed that have a smaller error than all of their neighbors. Although we broke the possible deadlock by adding a small random number to the error, a more sophisticated solution would lead to more parallel collapses and thus increase the split and collapse performance. In addition, the frame rate drops to approximately 12 fps for meshes containing one million vertices when drastically changing the approximation error. When the adaption finishes, the frame rate however returns to 60 fps. In total, adaption and propagation scale almost linear in the number of cores.

The modeling performance of our approach is approximately on par with algorithm of Marinov et al. [MBK07] with real-time frame rates for all our test models. In contrast to their approach we are however able to edit a mesh at different resolutions and the modifications are transferred directly to all LODs. Previous multi resolution modeling approaches like the method of Zorin et al. [ZSS97] are only suitable for models with subdivision connectivity. In Laplacian mesh editing [SCOL*04] a smoothed surface is used for editing and the topology is transferred back to the surface afterwards. The drawback of that approach is that modifications can only be performed for small ROIs with at most 100K vertices at interative frame rates. Our algorithm is on the other hand able to handle models containing up to serveral million triangles and edit them at real-time frame rates, independent of the ROI size. Figure 1 and 6 show some of the generated progressive meshes during editing.

7. Conclusion and Limitations

We have proposed a parallel progressive mesh generation and editing algorithm. Its input is an indexed face set from which it first generates a progressive mesh data structure. The progressive mesh can then be edited at any resolution. The modifications are automatically propagated to finer resolution using an encoding of the split operations based on

© The Eurographics Association 2014.

local coordinate systems. In addition to that, the coarser resolutions – up to the base mesh – are updated using memoryless simplification. This leads to a valid progressive mesh during the complete editing pipeline.

The main limitation of our algorithm is that the model size is currently limited to a few million triangles. This could be alleviated using out-of-core stream simplfication or possibly even compression techniques as in [DG12]. Another limitation is that the local ordering of operations is fixed after initial simplification. While this is necessary for animations, a partial re-simplification might be desirable after huge deformations.

Like all multi resolution modeling techniques, our algorithm is limited to geometric modifications of the vertices. Changing the connectivity of the mesh requires rebuilding the progressive mesh data structure. While it would be possible to locally re-compute the operations after large deformations, changing the mesh connectivity would require longer processing times than geometric modifications.

References

[BKS03] Bendels G. H., Klein R., Schilling A.: Image and 3d-object editing with precisely specified editing regions. In *Vision, Modeling and Visualisation 2003* (2003), pp. 451–460. 4

[BSS07] Boubekeur T., Sorkine O., Schlick C.: Simod: Making freeform deformation size-insensitive. In *Symposion on Point Based Graphics* (2007), pp. 47–56. 2

[DG12] Derzapf E., Guthe M.: Dependency free parallel progressive meshes. *Computer Graphics Forum 31*, 8 (2012), 2288–2302. 2, 3, 5, 6, 8

[DMG10a] Derzapf E., Menzel N., Guthe M.: Parallel view-dependent refinement of compact progressive meshes. In *Eurographics Symposium on Parallel Graphics and Visualization* (2010), pp. 53–62. 2

[DMG10b] Derzapf E., Menzel N., Guthe M.: Parallel view-dependent out-of-core progressive meshes. In *Vision, Modeling, and Visualization* (2010), pp. 53–62. 2

[FB88] Forsey D. R., Bartels R. H.: Hierarchical b-spline refinement. In *Proc. of the 15th annual conference on Computer graphics and interactive techniques* (1988), SIGGRAPH '88, ACM, pp. 205–212. 2

[Gar99] Garland M.: Multiresolution modeling: Survey & future opportunities. *Proc. of the Eurographics '99 – State of the Art Reports* (1999), 111–131. 2

[GDG11] Grund N., Derzapf E., Guthe M.: Instant level-of-detail. In *Vision, Modeling, and Visualization (VMV2011)* (2011), pp. 293–299. 2, 3, 6

[GH97] Garland M., Heckbert P. S.: Surface simplification using quadric error metrics. In *Proc. of the 24th annual conference on Computer graphics and interactive techniques* (1997), SIGGRAPH '97, pp. 209–216. 2

[GH98] Garland M., Heckbert P. S.: Simplifying surfaces with color and texture using quadric error metrics. In *Proc. of the conference on Visualization* (1998), pp. 263–269. 2

[GSS99] Guskov I., Sweldens W., Schröder P.: Multiresolution signal processing for meshes. In *Proc. of the 26th annual conference on Computer graphics and interactive techniques* (1999), SIGGRAPH '99, pp. 325–334. 2

[HSH09] Hu L., Sander P. V., Hoppe H.: Parallel view-dependent refinement of progressive meshes. In *I3D '09: Proc. of the 2009 symposium on Interactive 3D graphics and games* (2009), pp. 169–176. 2

[KCVS98] Kobbelt L., Campagna S., Vorsatz J., Seidel H.-P.: Interactive multi-resolution modeling on arbitrary meshes. In *SIGGRAPH* (1998), pp. 105–114. 2

[KVS99] Kobbelt L., Vorsatz J., Seidel H.-P.: Multiresolution hierarchies on unstructured triangle meshes. *Comput. Geom. 14*, 1-3 (1999), 5–24. 2

[LF03] Le Feuvre L.: Modelling and deformation of surfaces defined over finite elements. In *Proc. of the Shape Modeling International* (2003), IEEE Computer Society, p. 175. 2

[LSCo*04] Lipman Y., Sorkine O., Cohen-or D., Levin D., Rössl C., peter Seidel H.: Differential coordinates for interactive mesh editing. In *Proc. of Shape Modeling International* (2004), Society Press, pp. 181–190. 2

[LT98] Lindstrom P., Turk G.: Fast and memory efficient polygonal simplification. In *IEEE Visualization* (1998), pp. 279–286. 3

[Lue01] Luebke D. P.: A developer's survey of polygonal simplification algorithms. *IEEE Comp. Graph. Appl. 21* (2001), 24–35. 2

[MBK07] Marinov M., Botsch M., Kobbelt L.: Gpu-based multiresolution deformation using approximate normal field reconstruction. *Journal of Graphics, GPU, and Game Tools 12*, 1 (2007), 27–46. 2, 7

[MGH13] Maglo A., Grimsteadb I., Hudelota C.: Pomar: Compression of progressive oriented meshes accessible randomly. *Computers & Graphics 37* (2013), 743–752. 2

[MS11] Manson J., Schaefer S.: Hierarchical deformation of locally rigid meshes. *Computer Graphics Forum 30*, 8 (2011), 2387–2396. 2

[PH97] Popović J., Hoppe H.: Progressive simplicial complexes. In *Proc. of the 24th annual conference on Computer graphics and interactive techniques* (1997), SIGGRAPH '97, pp. 217–224. 2

[PR00] Pajarola R., Rossignac J.: Compressed progressive meshes. *IEEE Transactions on Visualization and Computer Graphics 6*, 1 (2000), 79–93. 2

[SCOL*04] Sorkine O., Cohen-Or D., Lipman Y., Alexa M., Rössl C., Seidel H.-P.: Laplacian surface editing. In *Proc. of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing* (New York, NY, USA, 2004), SGP '04, ACM, pp. 175–184. 2, 7

[Tau95] Taubin G.: A signal processing approach to fair surface design. In *Proc. of the 22nd annual conference on Computer graphics and interactive techniques* (1995), SIGGRAPH '95, ACM, pp. 351–358. 2

[WW94] Welch W., Witkin A.: Free-form shape design using triangulated surfaces. In *Proc. of the 21st annual conference on Computer graphics and interactive techniques* (1994), SIGGRAPH '94, ACM, pp. 247–256. 2

[ZSGS12] Zollhöfer M., Sert E., Greiner G., Süßmuth J.: Gpu based arap deformation using volumetric lattices. In *Eurographics (Short Papers)* (2012), Andújar C., Puppo E., (Eds.), pp. 85–88. 2

[ZSS97] Zorin D., Schröder P., Sweldens W.: Interactive multiresolution mesh editing. In *Proc. of the 24th annual conference on Computer graphics and interactive techniques* (1997), SIGGRAPH '97, pp. 259–268. 2, 7

© The Eurographics Association 2014.