Patch-Trees for Fast Level-of-Detail Synthesis

Hermann Birkholz
Research Assistant
Albert-Einstein-Str. 21
Germany, 18059, Rostock
hb01@informatik.uni-rostock.de

ABSTRACT

This paper describes a procedure that synthesizes Level-of-Detail (LoD) meshes from a tree of mesh-patches. The patch tree stores the surface of the original mesh in different detail levels. The leaf patches represent the original detail, while lower levels in the tree represent the geometry of their child nodes with less detail.

Such patch trees have a coarser granularity compared to basic approaches like “edge-collapse”. This is because only complete patches can switch their detail, instead of pairs of triangles. On the other hand, it can better utilize the graphics hardware, which is capable to render preloaded patches very fast. The problem of such a patch-based LoD approach is to join the patches of different resolutions together in a smooth mesh. This problem is solved by the use of different versions of the patch borders that depend on the detail level of the neighbor patches.

Keywords

Level-of-Detail, Batched Dynamic Mesh.

1. INTRODUCTION

The drastic increase in speed of graphics hardware demands new strategies for Level-of Detail algorithms in order to make use of the GPU processing power. Common LoD algorithms, such as vertex-trees or triangle-subdivision schemes, only add / remove two triangles in each refinement / simplification step. Used with many available large meshes, most of the frame time is spent for the frame-to-frame mesh update. This is due to the high CPU load of the update process, compared to the GPU load. A coarser update granularity in LoD hierarchies enables generating a higher GPU load, together with a lower CPU load. The use of patches furthermore enables to store the geometry data in the GPU memory and to use efficient geometry descriptions, such as triangle-strips.

The approach in this paper describes the steps to create a patch-tree and how to use it for LoD rendering.

2. RELATED WORK

For the online approximation of triangle meshes there exist many different techniques. All of them create a hierarchy in an offline process, which is used for a fast online approximation. Xia [Xia96] creates merge-trees, which are constructed by a sequence of merge operations on pairs of surface vertices. Each merge operation creates a new node in the merge-tree, which also stores an error value to allow a selective refinement. The leaf-nodes of the tree represent the original vertices of the corresponding mesh. Hoppe [Hop97] independently extended his Progressive Meshes [Hop96] in order to create a comparable hierarchical data structure. Both approaches have a granularity of two triangles per simplification / refinement operation and thus create high CPU loads. Furthermore, each update operation requires an adaption of the triangles in the direct neighborhood and thus another increase in CPU load.

Other algorithms avoid the neighborhood update by substituting the mesh patches. The MT-hierarchy method [Pup96] generates approximations from constrained cuttings through a previously constructed hierarchy. Each update operation in the hierarchy cut is equivalent to a merge of two surface vertices and thus the fine granularity still results in high CPU load. The same is true for the ROAM [Duc97] algorithm, where a hierarchy of right-angled triangles is used to approximate heightfield meshes with a constrained triangle substitution.

The BDAM [Cig03] approach first used higher granularity primitives. It constructs a hierarchy of right-angled triangles like ROAM, but each node

represents a patch of triangles. With the Adaptive Tetra Puzzles [Cig04], the idea of BDAM was ported to arbitrary meshes. Both approaches enable the synthesis of conformal (crack free) mesh approximations with constraints at the patch borders during the simplification. The outer border of the patch which is simplified must not be changed. Thus, patches which differ by only one detail level can be placed side by side without mesh cracks.

In [Cig05] the above approaches are generalized to Batched Multi Triangulations. A sequence of partitions is created for the mesh with a descending number of patches. Each pair of successive partitions in the sequence is cut against each other, in order to determine the parent-child relations in the hierarchy of patches. The geometry in the partitions is simplified according to the number of patches, while the geometry along the cuts is preserved. This hierarchy is used to create approximations by stitching patches of the different partitions together.

3. PATCH-TREE CONSTRUCTION

In this section the basic steps of the patch tree construction are described. A tree of triangle-patches has to be constructed, where each node contains a triangle-patch with the simplified geometry of its child patches. The nodes are further enriched with information about the change of their borders from lower to higher detail.

Creating the Patch-Hierarchy

To create the patch-tree, a greedy merge algorithm is used to create the initial patches. Therefore, all original triangles are treated as patches. The patches are then iteratively merged according to the resulting shape, until a desired number of patches is reached. This number of patches depends on the average number of triangles per patch in the patch-tree. The priority of the merge operations is measured by the perimeter of the patch and its surface area:

$$priority = \sqrt{\frac{outline^2 + 4\pi * area}{2}} \qquad (1)$$

This metrics prefers small and compact patches. To take only the patch perimeter into account, as done in [San01], leads to problems at mesh parts that are connected to the rest of the mesh with thin transitions (e.g. neck connects head and body).

Once the initial patches are created, their borders are smoothed with a local shortest path algorithm. This is done in order to facilitate stitching of patches with different triangle resolutions. The algorithm iteratively shortens the path between two patch corners. Therefore the path-vertices, and all vertices that are adjacent to them, are marked as optimization corridor. Now the shortest possible path is found in this corridor and optimized again, as long as optimizations are possible.

Figure 1 shows the result of the smoothing algorithm. The patches in the left image have rough borders. After straightening the borders, they are much smoother as visible in the right image. Furthermore the relatively compact shape of the patches can be recognized, which enables a good distribution of the image space error in later approximations.

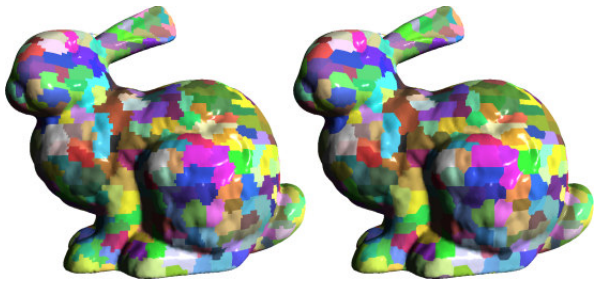

Figure 1: initial (l.) and smoothed(r.) borders

With the smoothed patches, a merge tree is created with a greedy algorithm again, using the same compactness metrics. The resulting unbalanced binary merge tree is then transformed to a balanced final patch-tree. This step is necessary, because we need explicit neighbor info of the patches in each level of the final tree. The leaves of the patch-tree are identical to the leaves in the merge-tree, but all reside in the same level of the tree. The following levels in the patch-tree are extracted from the binary tree, according to the compactness metrics and are provided with half the number of patches compared to the previous level.

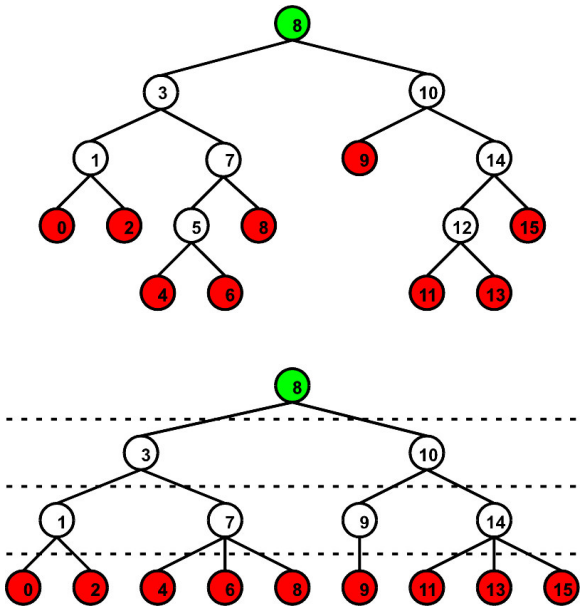

Figure 2: initial (t.) and final(b.) patch-tree

Figure 2 shows a small example. The binary tree (top) that was created by the patch merge algorithm is transformed into a general tree (bottom), whose leaf nodes are all situated on the same tree-level. Some merge tree nodes have been ignored (e.g. node 12) in this process, while other nodes appear multiple times (e.g. node 9).

LOD Creation

In order to enable an extraction of LoD approximations, the non-leaf nodes of the patch-tree have to be equipped with the simplified geometry of their child nodes. The construction of the LoD halves the number of triangles from each level to the next, starting at the leaf level. This ensures the same average number of triangles per patch in the whole patch-tree.

The simplification process works as follows:

- Initialize leaf patches with original geometry
- For each lower level
    - Merge geometry of child patches
    - Halve number of triangles in level
- Precompute patch border switches
- Compute split weights for the patches

In order to halve the number of triangles from level to level, an "half edge-collapse" [Kob98] algorithm is used. This ensures common vertices in child and parent patches. The surface error is measured with the QEM [Gar97] and the shape of the patch borders is preserved during the simplification. The simplified geometry then remains in the patches of the actual tree level and is used to initialize the simplification of the next level.

Figure 3: different detail levels

In figure 3, four different levels of the patch-tree are visible. The average surface area of the patches doubles from level to level, while their shape remains compact.

After the geometry of all levels was created, possible changes at the borders of all patches are computed. Therefore each patch references all of its adjacent patches (at least one common vertex). During the synthesis process, patch-refinement operations are applied to the approximated surface. The refinement-operations are restricted, in order to keep the difference between detail levels of adjacent patches below two. This means that refinement-operations are only allowed if all patches in the neighborhood have a higher or the same level. If not, the corresponding patches have to be refined first (forced refinement). Due to the restricted difference in the levels of adjacent patches, only one additional version of each border between two patches must be computed. In this approach, the higher level patch has to adapt to the lower level. The indices of all border vertices of a patch that do not appear in the parent patch, can be changed to the closest border vertex that appears in the parent patch. This ensures a closed triangle mesh without cracks for each approximation. To determine the correct replacement index, the distance to the next left and right border vertex, which also belongs to the parent patch, is compared.

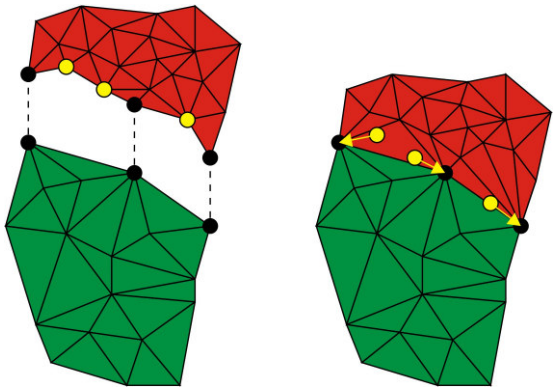

Figure 4: border versions

Figure 4 shows an example for the border adaption. The indices of the yellow vertices in the high level patch have to be changed to match them with the border of the lower level patch. The arrows show the position of the according replacement targets.

The last step computes view-independent refinement priorities, which measure the geometric distortion for a replacement of some child patches by their parent patch. The priority is measured with the average squared distance of the vertices in the child patches to

the parent patch. To compute this average value, all vertices of the child patches are mapped to their parent patch. After that, the accumulated squared distance of all vertices is divided by the number of vertices in the patch. The mapping is realized by locking all common vertices of the child patches and the parent patches. For all other vertices the "mean value coordinates" [Flo03] are determined and used to compute their mapping in the parent patch.

4. LOD SYNTHESIS

The patch-tree can now be used to synthesize approximations of the original mesh. A simple split-only version of the approximation process first inserts the root patch into a priority queue. The priority of the patch is made view dependent by dividing the view independent priority by the squared distance between patch and viewer. Furthermore this first patch is marked and then stored in a list of used patches. Now a loop is started that iteratively splits the patches in the approximation, if it is possible. Therefore the neighbors of the first patch the queue are checked. If all neighbors in the same patch-level are marked, the first patch is removed from the queue and the list of used patches. Furthermore its child patches are marked and put into the list of used patches. If the child patches are not from the leaf level, they are also put into the priority queue, with their view dependent weights.

If any of the neighbor patches in the same level of the patch-tree were not marked yet, the priority of their parent patches (which were surely marked) is lifted to a higher priority than the first patch in the queue. This ensures that patches can only be split if their neighborhood is at least at the same patch-tree level.

The refinement process can be stopped by different events. For the tests in the next section, a triangle threshold was chosen.

To waste less triangles in invisible regions, a view frustum culling has been implemented. Each priority computation also tests the bounding sphere of the patch against the planes of the view frustum. By means of the results, the patch is tagged as completely visible, partial visible or completely invisible. The priority of completely invisible patches is always set to zero. Completely visible or invisible patches can inherit this property to their child patches to save culling tests. This ensures the distribution of most triangles within the view frustum.

Frame-to-frame coherency can also be exploited by the help of another priority queue, which stores possible merge operations of patches. But due to the low tree size, the split-only method works fast even for large meshes.

5. RENDERING

Before rendering, the borders of all used patches have to be adapted. Therefore all neighbors of each used patch in the same patch-tree level are checked for a mark. The correct border vertex indices are chosen according to the acquired neighbor information. If the neighbor patch is marked, the original indices are used. Else the previously determined parent versions must be used. The resulting mesh is crack free, because the possibly different triangulations on the patch borders have been repaired now.

Figure 5 shows the "Bunny" mesh without (top) and with (bottom) crack removal. The holes in the upper image are colored in bright red. The bottom image does not show holes, because the patch borders which caused the holes were updated.

The adapted patches can now efficiently be rendered as vertex-, normal- and triangle-arrays. This process can even be accelerated by dividing the patches into a constant partition, which remains in GPU memory, and the changeable parts. Thus only the few changeable parts must be transferred to the GPU, while most of the mesh already resides in the GPU memory.

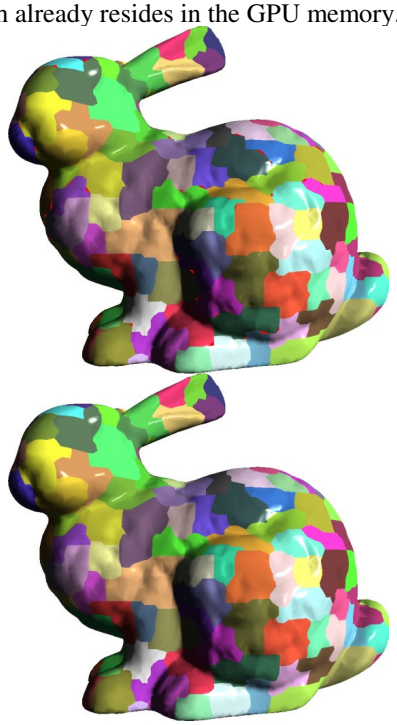

Figure 5: with cracks (l.)/removed cracks (r.)

6. RESULTS

The patch-tree based LoD synthesis has been tested with several meshes. The average patch size was adjusted to 100 triangles to avoid strong popping effects. All meshes were partitioned into compact patches.

The "Bunny" mesh consists of almost 70,000 triangles, while the "Armadillo" mesh uses almost 346,000 triangles and the "Rough Planet" even almost 2,1 Mio triangles. Figure 6 shows the leaf patches of the "Armadillo" and the "Rough Planet" mesh.

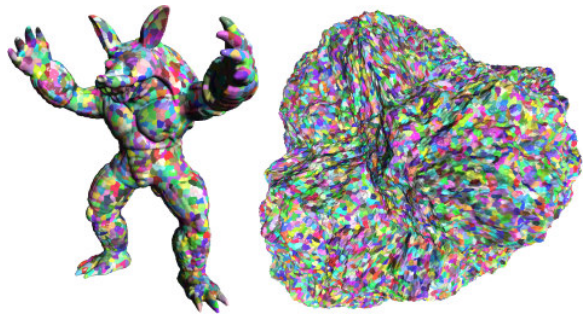

Figure 6: leaf patches of other test meshes

All meshes are rendered with simple vertex-, normal- and triangle-arrays with approximately 15 Mio. triangles per second, independent from the size of the approximated mesh (P4 – 2GHz, 1GB Ram, ATI Radeon 9800 Pro). With the use of float-buffers on the GPU and a patch size of 1000 triangles, the rendering throughput raised up to 25 Mio. triangles/second. The number of triangles for the approximation was adjusted to 500,000 triangles at a frame rate of 50 fps.

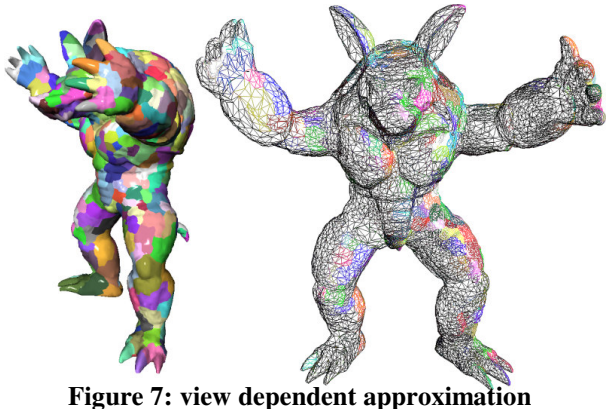

Figure 7: view dependent approximation

Figure 7 shows a 80,000 triangle approximation view (left) of the „Armadillo" mesh and an overall view (right). The effect of the adaptive triangulation can be seen here. The left hand shows the highest detail, while the right part of the mesh, which has a higher distance to the viewer, is triangulated significantly coarser. Due to the low number of triangles per patch, the transition of detail is relatively smooth.

7. CONCLUSION

This paper introduced patch-trees, which can be used to synthesize LoD meshes. The algorithm is relatively easy to implement and shows good results. Due to the selectable granularity (average patch size), it is possible to reduce the CPU load in the synthesis period and thus enables highly detailed approximations. The results can still be improved with a more GPU oriented version of the rendering process. Furthermore a "Out of Core" version of the patch-trees should be easily implementable.

8. REFERENCES

[Cig03] Cignoni, P., Ganovelli, F., Gobbetti, E., Marton, F., Ponchio, F., and Scopino, R. BDAM: Batched dynamic adaptive meshes for high performance terrain visualization. Computer Graphics Forum, 22(3):pp.505–514, Sept. 2003.

[Cig04] Cignoni, P., Ganovelli, F., Gobbetti, E., Marton, F., Ponchio, F., and Scopino, R. Adaptive tetrapuzzles: efficient out-of-core construction and visualization of gigantic multiresolution polygonal models. ACM Trans. Graph., 23(3):pp.796–803, 2004.

[Cig05] Cignoni, P., Ganovelli, F., Gobbetti, E., Marton, F., Ponchio, F., and Scopino, R., Batched Multi Triangulation, Proceedings IEEE Visualization, IEEE Computer Society Press, 2005.

[Duc97] Duchaineau, M.A., Wolinsky, M., Sigeti, D.E., Miller, M.C., Aldrich, C., and Mineev-Weinstein, M.B. ROAMing terrain: Real-time optimally adapting meshes. Proceedings IEEE Visualization, IEEE Computer Society Press, 1997.

[Flo03] Floater, M.S., Mean value coordinates. Computer Aided Geometric Design, Elsevier Science Publishers B. V., 2003

[Gar97] Garland, M., Heckbert, P.S., Surface Simplification Using Quadric Error Metrics, SIGGRAPH '97 Conf. Proc., pp. 209-216, 1997

[Hop96] Hoppe, H., Progressive Meshes, Computer Graphics:pp.99-108, 1996

[Hop97] Hoppe, H. View Dependent Refinement of Progressive Meshes, Computer Graphics, 1997

[Kob98] Kobbelt, L., Campagna,, S., Vorsatz, J., and Seidel, H.-P., Interactive multi-resolution modeling on arbitrary meshes, In Proceedings of the 25th annual conference on Computer graphics and interactive techniques, pp. 105-114, 1998

[Pup96] Puppo, E. Variable Resolution terrain surfaces, Proc. Of 8th Canadian Conference of Computational Geometry, 1996

[San01] Sander, P.V., Snyder, J., Gortler, S.J., and Hoppe, H. Texture Mapping Progressive Meshes, Computer Graphics Proceedings, ACM Press, 2001

[Xia96] Xia, J.C. and Varshney, A. Dynamic view-dependent simplification for polygonal models. Proceedings IEEE Visualization, IEEE Computer Society Press, 1996