A HYBRID APPROACH FOR HIGH QUALITY REAL-TIME TERRAIN RENDERING AND OPTIMIZED A-PRIORI ERROR ESTIMATION

John Doe
Unknown
{jondoe}@mail.com

Keywords: Terrain rendering, level of detail, longest-edge-bisection, refinement criteria, cached geometry, multi-core CPU.

Abstract: This paper describes a hybrid approach for high-quality real-time terrain rendering which combines the advantages of fine-grain triangle-based CPU and GPU-optimized patch-based algorithms with regard to terrain adaptation and CPU-GPU transfer workload. In more detail, our scheme selectively refines a cluster over a regular domain at the granularity of individual triangles to generate highly-adaptive view-dependent geometry. In contrast to pregenerated patches, a substantial reduction of the number of primitives is achieved. To reduce the CPU-GPU workload inherent in triangle-based algorithms, we exploit frame-to-frame coherences by caching refined geometry on local VRAM in combination with an optimized error evaluation. Finally, a significant increase in the performance is reported.
Additionally, our work proposes a per-vertex preprocessing scheme for necessary error-bounds calculation that massively exploits current multi-core architectures. This scheme is based on the widely applied *longest-edge-bisection* approach. Compared to current computationally expensive, recursive procedures utilizing the aforementioned subdivision scheme, a performance gain of several orders of magnitude is reported.

1 INTRODUCTION

In the last decade, many terrain rendering algorithms have been developed. Due to the close interrelation of rendering quality, rendering time and available resources, a compromise between these aspects has to be found (Löffler et al., 2009). This, in turn, is a complex and challenging task.

To achieve high-quality images in real-time, a multi-resolution data structure is required, which is generated in a pre-process prior to rendering. Since the generation of triangulated irregular networks (TIN) commonly entails an exhaustive preprocess combined with large storage costs, data structures on a semi-regular basis are utilized in most cases. This leads to an improved balance in terms of preprocessing time and height field approximation. In that context, *longest-edge-bisection*(Pajarola and Gobbetti, 2007) represents a widely applied recursive subdivision scheme. With the help of this scheme, refinement criteria are determined on a per-vertex level, representing the object-space error bounds. With this data structure, view-dependent rendering is realized by selectively refining the terrain mesh. This is accomplished at the granularity of either triangles on CPU side or GPU-optimized triangle clusters, called patches(see(Pajarola and Gobbetti, 2007) for a recent survey). Given a particular error threshold, triangle-based CPU algorithms highly adapt the terrain, but this leads to the well-known CPU-GPU transfer bottleneck. To minimize computational overhead, such algorithms can benefit from frame-to-frame coherences. In contrast, patch-based methods reduce the bottleneck by exploiting recent GPU triangle throughput capabilities, but result in a higher number of rendered triangles. Moreover, most patch-based methods select the appropriate patches from scratch each frame, and thus, do not take advantage of frame-to-frame coherences.

The goal of this work is to combine the advantages of both approaches with regard to terrain adaptation and workload reduction. We therefore propose a novel hybrid approach for view-dependent refinement. In more detail, our scheme selectively refines a patch over a regular domain at the granularity of individual triangles to generate highly-adaptive view-dependent geometry. Hence, we are able to drastically reduce the number of primitives with respect to a particular error tolerance. Furthermore, we substantially reduce the CPU-GPU bottleneck by exploiting frame-to-frame coherences in that we cache geometry in the GPU memory. We further check the cache validity by testing one point only instead of evaluating the entire cached geometry. This results in a significant performance gain. Moreover, we significantly accelerate the a-priori refinement criteria determination by adapting the indexing scheme from (Hwa et al., 2004). In this way, we are able to massively exploit multi- and many-core CPU architectures. As a result, preprocessing time could be significantly decreased by several orders of magnitude.

The outline of our approach can be summarized as follows: Given a regular height-field, we first compute the refinement criteria using our novel a-priori error estimation scheme characterized in Section 3. A multi-resolution patch hierarchy is then derived from both the height field and the criteria. During rendering, we select appropriate patches from the hierarchy followed by dynamic view-dependent refinement of the selected patches. Furthermore, we exploit frame-to-frame coherences to reuse cached refinement results. This novel hybrid approach is presented in Section 4 in more detail. The results reported a significant performance increase with respect to both the a-priori error estimation and the view-dependent rendering. The results are discussed in Section 5. Before going into detail, we reflect previous work in the next section.

2 RELATED WORK

In the last decade, many terrain rendering algorithms have been proposed. High-quality real-time rendering, especially terrain rendering, must deal with very large data-sets. Hence, multi-resolution data structures, mostly generated a-priori, are utilized to gain interactive frame rates. For quality purposes, at the same time, object space error metrics are applied. Moreover, the underlying data structures and level–of–detail (LOD) methods applied distinguish most terrain rendering approaches (Pajarola and Gobbetti, 2007). As discussed in (Cignoni et al., 2003), the rapid adaptive construction and display of continuous terrain surfaces can be solved by two types of data structures: First, techniques that are based on general, mainly unconstrained, triangulations (TIN), and second, semi-regular representations. Examples of the former category are described in (Puppo, 1996; Cignoni et al., 1997; Hoppe, 1998). Such class of algorithms generates a highly adaptive and high-quality terrain mesh with regard to the number of triangles / error count (Evans et al., 2001a). However, these algorithms require much more complicated multi-resolution data structures. Moreover, their a-priori generation entails an exhaustive pre-process and high storage costs due to necessary connectivity storage caused by irregularity. A first GPU-based approach for such class of LOD algorithm has been proposed in (Hu et al., 2009).

To gain a better balance of approximation and preprocessing times, structures of the latter category are used for terrain rendering (Pajarola and Gobbetti, 2007). For this class of semi-regular algorithms, several mesh generation algorithms have been developed which use the so-called method of *longest–edge–bisection* for view-dependent refinement or simplification of the terrain mesh (Duchaineau et al., 1997; Pajarola, 1998; Lindstrom et al., 1996; Röttger et al., 1998; Evans et al., 2001b; Evans et al., 2001a). The scheme is as simple as powerful and has been applied in various forms and data structures. (Duchaineau et al., 1997; Evans et al., 2001b) use triangle bintree hierarchies and perform split and merge operations for recursive refinement and simplification, respectively. In order to generate crack-free approximations, force-split operations are introduced, whereas (Evans et al., 2001b) proposed further improvements on efficient representations and mesh traversal. Lindstrom (Lindstrom and Pascucci, 2002) designed a method for view-dependent refinement, while (Lindstrom et al., 1996; Pajarola, 1998) deploy restricted quad-tree triangulations. Especially the algorithm by Lindstrom et al. (Lindstrom and Pascucci, 2002) is a general scheme for terrain mesh simplification. The method treat the entire per-vertex parent-child relationships as an implicit vertex-graph representing the surfaces to be extracted during view-dependent refinement. Refinement criteria are determined and saturated over the hierarchy which represents per-vertex related error of an arbitrary error metric and the radius of the error spheres. Hence, a hierarchy of nested error spheres is built. In this way, only position and refinement criteria are stored per vertex and no additional data structures are required. A recursive algorithm generates a view-dependent triangulation of the height-field. Due to the nesting property, no special constraints must be considered to ensure crack-freeness. Furthermore, the refinement algorithm generates a triangle-strip which is well suitable for recent graphics hardware. The computation scheme as well as the proposed error metric is well designed and has been used for a wide range of algorithms, for instance (Cignoni et al., 2003; Bösch et al., 2009). However, the algorithm generates the triangulation from scratch for the entire height-field. As a consequence, caching parts of the geometry on fast local video memory is not feasible.

In general, all of these algorithms determine on the CPU the preferably minimum number of triangles to be rendered based on decisions at the triangle/vertex primitive level. Despite these high adaptation capabilities given a particular error threshold, the triangle-based approaches lead to the well-known CPU-GPU bottleneck (Pajarola and Gobbetti, 2007; Cignoni et al., 2005).

To overcome this bottleneck and to fully exploit the capabilities of current graphics hardware, it is necessary to select and send clusters of geometric primitives to be rendered with just a few CPU instructions (Cignoni et al., 2005). Following this approach, various GPU-oriented multi-resolution structures have been recently proposed, based on the idea of moving the granularity of the representation from triangles to triangle patches, for instance (Pomeranz, 2000; de Boer, 2000; Cignoni et al., 2003; Hwa et al., 2004; Bösch et al., 2009). The benefit of these approaches is that the needed per-triangle workload to extract a multi-resolution model can be reduced by orders of magnitude. The small patches can be preprocessed and optimized off-line for a more efficient rendering. Especially (Cignoni et al., 2003) combines the advantages of semi-regular structures and TIN-based batches for high performance terrain rendering.

A very interesting approach is proposed in (Levenberg, 2002). The CABTT algorithm dynamically extracts triangle clusters from triangle bin-trees during view-dependent rendering and caches the geometry on fast local VRAM. To exploit frame-to-frame coherences, cached geometry is reused. Therefore, an aggregated error measure is determined for each cluster, which drives the splitting and merging of cluster. The algorithm reports a significant performance gain in contrast to ROAM. However, the clusters are not optimized for recent GPUs and a triangle bin-tree must be generated for the height-field.

Recent GPU-bounded terrain rendering algorithms take full advantage of semi-regular data structures and utilize large patches. For an excellent overview, we refer to (Pajarola and Gobbetti, 2007). In general, the consequence of changing multi-resolution granularity is a reduction of the model flexibility: more triangles than necessary will be rendered to achieve a given accuracy. However, less CPU performance is required. Due to the fact that patches are cached in local VRAM, patch-based methods directly benefit from recent GPU throughput capabilities. Hence, this strategy seems to be a good choice.

3 A-PRIORI ERROR ESTIMATION

The first step of our approach is the a-priori estimation of error bounds. We therefore present our novel scheme for parallel refinement criteria determination in this section. A straight forward approach follows the recursive subdivision scheme and computes the refinement criteria of the vertex-graph in a post traversal manner. This is rather slow.

Our goal is to accelerate this process in that we take advantage of recent multi-core CPU architectures. Therefore, we have to parallelize this process. However, parallelizing the recursive process is not trivial due to the dependencies of parent-child relationships. Hence, our general approach is to substitute the recursive by an iterative process. Hence, parent-child relationships must be extractable for arbitrary positions of one particular level without having to traverse the hierarchy node-by-node in a bottom-up recursive fashion. Our solution is to harness parent-child relationships from special entities, called diamonds. Diamonds can be characterized consisting of triangles of the same level, which share their adjacent hypotenuse (Hwa et al., 2004). A diamond is uniquely associated with one vertex (the *diamond vertex*), one edge, and one quadrilateral face. If a diamond is known, the parents and children of it can be described using a special indexing scheme (see Section 3.1). The idea is now to adapt this scheme to the vertex graph structure, as each vertex in this graph represents a diamond. In this way, parent-child relationships of two consecutive levels can be extracted, if those of the coarser level are known(see Section 3.2). Furthermore, this approach enables us to consider all vertexes of one level independently of each other. Hence, we determine the refinement criteria by sequentially treating the levels bottom-up. Within one level, the determination of refinement criteria in a highly-parallel way is feasible.

3.1 Basic Indexing Scheme

Refinement criteria determination implies determination of parent-child relationships. Following the indexing scheme of (Hwa et al., 2004), the face of a diamond d is described by the diamond ancestors $a_0, \ldots, a_4$ (see Figure 1). The hypotenuse of the two

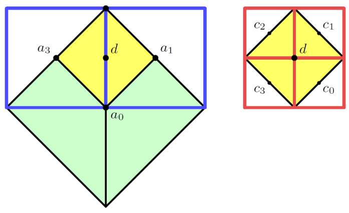

Figure 1: A diamond d (yellow) is shown with its ancestors (left) and its children (right)(Hwa et al., 2004). The green ancestor is a_0, the two parents are a_1 and a_3 (blue outline). ds children are $c_{0...3}$ (red). Note that the orientation is rotated by 45° each level.

adjacent triangles, is characterized by the ancestors a_0 and a_2 and split by d. Using the locations of the ancestors, the locations of the child diamonds c_i with $i = 0, \dots, 3$ can be determined as follows:

$$c_i = (a_i + a_{(i+1) \mod 4})/2 \tag{1}$$

Using this information, we are able to compute and saturate the refinement criteria of a diamond d as described in (Lindstrom and Pascucci, 2002). In the implicit vertex-graph, each vertex represents a diamond. However, ancestors as well as children are described by the recursive indexing scheme, rather than by an explicit definition. In the next section, we describe how to solve parent-child relationships with regard to the position of a vertex in the vertex-graph.

3.2 Our General Scheme

To process the refinement criteria in parallel, we must identify the ancestors of a vertex in the vertex graph to adapt the index scheme from (Hwa et al., 2004) described above. For that reason, we position special *diamond kernels* on appropriate vertexes. A *diamond kernel* with a kernel configuration k and the kernel position $\mathbf{k_p}$ provides a computation rule to identify the four ancestors for a diamond vertex with respect to several parameters. These parameters are described in the following.

First, due to alternating diamond orientations, each level l can be categorized into an $even = (n \mod 2 == 0)$ and an $odd = (n \mod 2 == 1)$ level considering the vertex graph, where $2 * n$ levels exist with regard to height field sizes of $2^n + 1$. Hence, we introduce two different diamond kernels for *even* and *odd* levels, respectively. The configurations are illustrated in Figure 2.

Since different levels form different diamond sizes, the distance $s = 2^{l/2}$ of parent and child diamond vertexes is required as the second parameter. Dependent on the level l, the third parameter, bias b, represents the specific permutation of the ancestors.

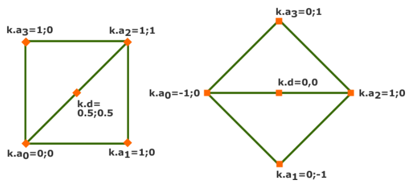

Figure 2: Configuration of the diamond kernels for *even* (left) and *odd* (right) levels, respectively.

As can be seen, the permutation of the ancestors a_0 and a_2 can be neglected, since the order of vertexes is only relevant for rendering in the sense of back-face culling. Hence, we derive two cases: Bias $b = 1$ represents a horizontally aligned diamond, whereas $b = 0$ is a vertically aligned diamond. b can be determined as follows:

$$b = \left\{ \begin{array}{lll} (k_{p.x} + k_{p.y}) & \mod 2 & l = even \\ k_{p.y} \mod 2 & & l = odd \end{array} \right\} \tag{2}$$

With the help of the parameters described above, the locations of the ancestor a_i and the diamond vertex d can now be determined as follows, where *makeIdx* realizes the mapping of a position to an index:

$$\begin{aligned} j &= (i+b) \mod 4 & (3)\\ a_i &= makeIdx((\mathbf{k_p} + \mathbf{k}.\mathbf{a_j}) * s) & (4)\\ d &= makeIdx(\mathbf{k_p} * s + \mathbf{k}.\mathbf{d} * s) & (5) \end{aligned}$$

Due to this scheme, all vertexes on one level can be processed independently of each other as long as the child vertexes have already been processed. As shown in Figure 3, we can now determine the refinement criteria by exploiting diamond parent-child relationships in a bottom-up fashion. For each level, we apply the *diamond kernels* and compute the refinement criteria as exemplarily shown in the figure. Due to fact that the kernel attributes only depend on the current level and the position of the kernel, we can exploit the power of current multi-/many-core architectures to carry out computation in a parallel fashion. Results report a significant performance gain in contrast to the naive approach as shown in Section 5.

4 OUR HYBRID APPROACH

After describing our novel scheme for efficient error estimation (see Section 3) as the first step of our approach, we introduce our novel hybrid method for view-dependent terrain rendering. The goal is to significantly reduce the number of triangles for a given

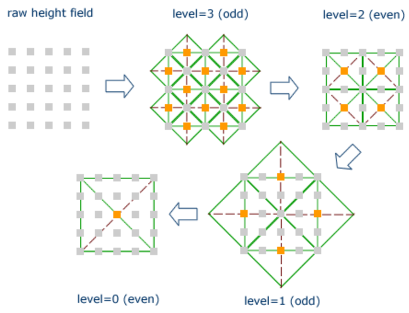

Figure 3: Processing sequence of a 5x5 height-field. Starting at the finest level, for each diamond on a level the refinement criteria are computed and saturated by maximizing the error over the child vertexes. The vertexes on a level are processed in parallel. This drastically increases the processing performance.

error tolerance and the vast triangle workload during the rendering process. The idea is to combine a triangle-based CPU method with a GPU-optimized patch-based approach. An overview of the pipeline is illustrated in Figure 4.

Based on the a-priori error estimation, the next step is to construct a semi-regular multi-resolution data structure of patches. For each patch, we must determine the error metric for view-dependent refinement. For this purpose, we exploit the per-vertex refinement criteria from the first step. The construction of this multi-resolution hierarchy is performed a-priori and is described in Section 4.1. During view-dependent rendering, we select appropriate patches. For each selected patch, a triangulation is generated by view-dependent refinement at the granularity of triangles. Due to this method (see Section 4.2), we drastically reduce the triangle count per patch in contrast to the regular patch triangulation while granting a particular error tolerance. The triangulation of a patch is cached on local VRAM for fast rendering. We reuse this cached triangulation to exploit frame-to-frame coherences. Therefore, the validity of the cached triangulation has to be evaluated for subsequent frames as described in Section 4.3. Instead of inefficiently checking the entire triangulation in the sense of evaluating each vertex, we introduce a method for testing one point only. As a consequence of our general approach, the triangle count as well as triangle workload is substantially reduced. Hence, rendering performance increases by an order of magnitude as reported in Section 5.

4.1 Patch Hierarchy

In an a-priori step, we construct a multi-resolution hierarchy of patches based on longest–edge–bisection.

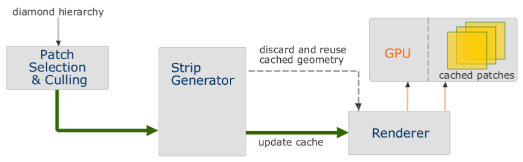

Figure 4: Overview of the rendering pipeline of our novel hybrid approach.

In this case, a cut through the hierarchy represents a coarse grain triangulation of the terrain, in that each precomputed patch triangle represents a regular subtriangulation. Assuming patches over a regular domain, view dependent refinement of a patch can be deployed by refining the patch triangle (see Figure 5).

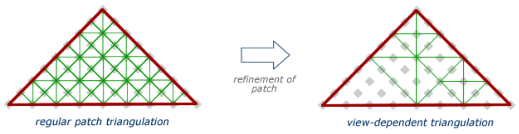

Figure 5: A patch in a semi-regular hierarchy over a regular domain. View-dependent refinement can be applied by refining the patch triangle.

For each patch, refinement criteria must be determined to select appropriate patches during view-dependent rendering. Therefore, the refinement criteria from the a-priori error estimation serve as a basis. A patch must be split, only if a vertex introduced by its children becomes active and hence, part of the triangulation. Consequentially, we iterate through such vertexes and estimate the maximum error of the patch. Thus, we can use the algorithm of (Lindstrom and Pascucci, 2002) to determine if a patch is active or not. Due to the nested properties of the vertexes, the patch hierarchy is also nested.

We choose a *diamond-graph* of tiles as proposed in (Hwa et al., 2004) where each tile holds the vertex or vertex attributes of two triangular patches with an adjacent hypotenuse. The data structure guarantees crack-freeness, and patch related vertex data can be handled and transferred to the GPU efficiently. Furthermore, we harness frame-to-frame coherences to cache vertex data on the GPU efficiently. During the rendering, we dynamically generate an optimal view-dependent triangulation for selected patches, which is described in the next section.

4.2 View-Dependent Triangulation

To reduce the number of triangles per patch, we refine a patch at the granularity of individual triangles. We therefore adapt the the algorithm of (Lindstrom and Pascucci, 2002) for triangular patches over a regular grid of the size $2^n + 1$. The algorithm generates a triangle index-strip during the recursive traversal of the implicit vertex graph. Hence, for a given patch triangle (vl, va, vr), we determine the midpoint $vm = (vl + vr)/2$ and recursively refine the triangles (vl, vm, va) and (va, vm, vr), respectively. The resulting strip is transferred to the graphics board. As the vertex-data still resist in the local video memory, we are able to fully exploit the rendering performance of recent GPUs. Furthermore, CPU-GPU transfer is reduced to small index-strip packages, in contrast to a per-vertex transfer. We perform the refinement of patches and the rendering concurrently. However, the refinement from scratch of each visible patch in each frame is a tedious task.

Fortunately, view-dependent triangulation can be reused for subsequent frames (Levenberg, 2002). Therefore, the triangulation must be validated. If a triangulation is still valid, we are capable of rendering the existing triangulation persisting on the graphics board. Hence, a drastic reduction of CPU-GPU communication and a significant gain in rendering performance is expected (Levenberg, 2002). Even though, this requires a per-vertex evaluation of the current triangulation. The method guarantees a valid cache reuse, but requires expensive CPU resources at the same time. To reduce CPU workload, we use a conservative rather than the exact validation. We estimate the cache validity by determining a region in which the triangulation is valid in any case. Due to a simple region check - described in the next section - we are allowed to substantially reduce per-triangle CPU workload and CPU-GPU communication.

4.3 Optimized Error Evaluation

A triangulation can be treated as a cut through the implicit vertex-graph. All vertexes on the cut represent the vertex frontier (Hoppe, 1997). All vertexes on and "above" the cut are active and thus part of the current triangulation. A cut is still valid, only if the vertex frontier is unmodified. In detail: all vertexes on the frontier must be active, while their children must be inactive. In our case, we would need an additional data structure to store the vertex frontier. As described above, the CPU-workload is expensive in the sense of evaluating the entire vertex frontier. As a consequence, we try to estimate the cache validity by determining a *region of validity* for a triangulation. The idea is to confine the validity evaluation to checking the new view-point against this region.

Using the refinement criteria, each vertex v_i blows up a spherical region $s_i = (\mathbf{p}_i, r_i^s)$. If the view-point $\mathbf{e}$ is located in that sphere, the vertex is active. Hence,

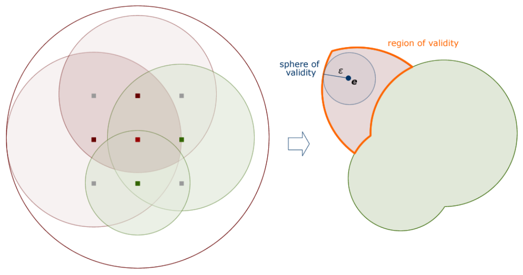

Figure 6: The red region are the spheres of the active vertexes and the green ones of the inactive vertexes. The difference of the intersection of all red and the union of all green spheres builds the *region of validity* for a view-point. Only if subsequent view-points are in that region, the triangulation is valid. ε and the current view-point $\mathbf{e}$ represent an upper estimate of this region.

the *region of validity* of a cut can now be described as the difference of the following two regions (see Figure 6):

1. the intersection of all spheres of the vertexes of the active frontier
2. the union of all spheres of the active frontier child vertexes

However, the resulting region is very expensive to evaluate. Hence, we try to simplify this region to a conservative estimation rather than the exact validation. During the view-dependent refinement, we identify the *sphere of validity* for the current view-point. The sphere is characterized by the current view-point $\mathbf{e}$ and the minimal distance ε between the view-point and the *region of validity* described above. In more detail, ε is the minimal distance between the view-point $\mathbf{e}$ and any sphere s_i. To figure out the radius r_i^s of the sphere s_i with regard to the viewing parameter K (Lindstrom and Pascucci, 2002), following equation is used:

$$r_i^s = \frac{\delta_i}{K} + r_i \tag{6}$$

The resulting *sphere of validity* is an upper estimate of the region described above. Hence, we guarantee that a triangulation is always valid, if subsequent view-points are inside this sphere. As a consequence, we only need to check $\varepsilon < ||\mathbf{e} - \mathbf{e}'||$ for the new viewpoint $\mathbf{e}'$ to validate the triangulation. If a triangulation is valid, the cached geometry is rendered, otherwise a re-triangulation is initiated. As reported in the next section, this increases the rendering performance.

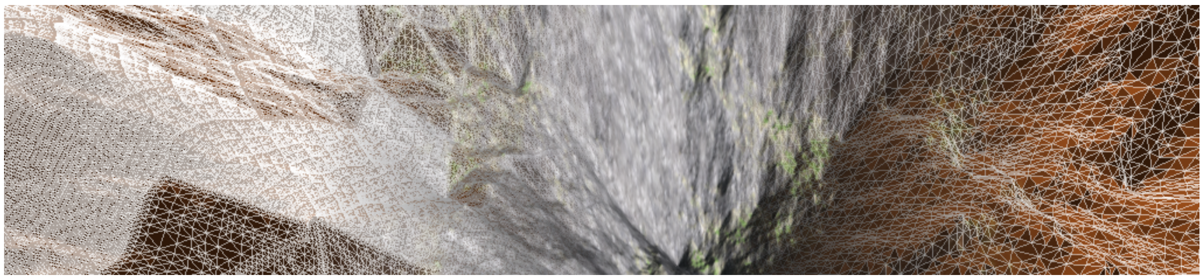

Figure 7: Comparison of preprocessed and dynamic triangulation of patches. left: static pre-generated patch triangulation; middle: textured representation; right: dynamic triangulation generated by our hybrid approach. Notice the significant reduction of triangle count on the right side.

5 RESULTS

We implemented our approaches using C++ and OpenGL. We tested height fields with respect to a single dimension of 1025, 2049, 4097 and 8193 to measure the performance of our novel a-priori error estimation scheme. Furthermore, two different hardware configurations were utilized. These include a single-core Pentium 4 3 GHz and a Core 2 Quad 2,83 GHz (quad-core) in order to evaluate hardware parallelism capabilities. The respective preprocessing times are shown in Table 1.
The results clearly show that using our parallel error estimation scheme decreases preprocessing times in any case. On both configurations, our approach diminishes preprocessing time linearly with increasing height field sizes. However, on the single-core environment our approach accelerates the processing by factor 4-5. Exploiting hardware parallelism the factor even rises up to 10-12. The results clearly show a significant improvement of preprocessing performance under all circumstances. As a result, our approach is well-suited for on-the-fly processing of height fields.

The results of our hybrid approach have been captured on the quad-core configuration equipped with a GeForce GTX 280. We chose a screen-resolution of 1280x800. For a quantitative evaluation, we compared our hybrid method with a pre-generated patch-based method. More precisely, a static patch size of 65 was considered a good selection according to (Bösch et al., 2009). As described in (Lindstrom and Pascucci, 2002), rendering and dynamic tessellation were driven concurrently. We compared height field sizes of 1025 (Crater), 2049 (Puget Sound) and 4097 (Kauai). The results are reported in Table 2.
As figured out, our approach significantly reduces the triangle count per frame. In more detail: our hybrid approach enables us to decrease the triangle count by 84% with regard to the minimum error tolerance. For instance, the pre-computed patches require up to 4,6 million triangles whereas our approach only needs 758 thousand primitives. With small error tolerances in general, high resolution patches must be selected, which results in a vast amount of rendered primitives. In contrast, an entire significant performance gain is achieved using our method, particularly with increasing error tolerances. This is due to the frequent cache reuse capability of our algorithm. In terms of minimum tolerances, the necessary high re-triangulation effort affects rendering performance and caching capabilities.

CPU	Height field size							
	1k		2k		4k		8k	
	rpcs	*ppcs*	*rpcs*	*ppcs*	*rpcs*	*ppcs*	*rpcs*	*ppcs*
Single – Core	1,43	0,28	5,17	1,18	20,03	4,71	65,3	16,9
Quad – Core	0,58	0,05	2,27	0,20	9,18	0,76	36,8	3.18

Table 1: Performance comparison of preprocessing times in seconds for the four height field sizes in regard to the two hardware configurations. Note that *rpcs* means the straight forward recursive computation scheme, while *ppcs* represents our novel parallel computation scheme.

Error tolerance	Height field size					
	1k		2k		4k	
	fps	num_Δ	fps	num_Δ	fps	num_Δ
	Patch-based approach					
0,5	284	1082k	143	2732k	82	4642k
1,0	286	1079k	191	1885k	121	3001k
2,0	291	1037k	261	1223k	212	1522k
	Our hybrid approach					
0,5	299	947k	172	665k	113	758k
1,0	427	472k	343	343k	274	339k
2,0	570	175k	566	147k	497	136k

Table 2: Results of the *quad-core* test environment with regard to a patch-based implementation in comparison with our hybrid approach. The table shows the average frame rate fps and the average number of rendered triangles per frame num_Δ depending on several error tolerances and height field sizes.

6 CONCLUSION

We presented a novel approach for a-priori error estimation scheme exploiting hardware parallelism which is widely applicable. Furthermore, we proposed a novel hybrid approach for view-dependent refinement. This algorithm incorporates the advantages of both patch-based and triangle-based data structures with respect to high adaptation capabilities and reduced workload. In more detail, we suggested selecting appropriate patches from a predetermined patch hierarchy and refining them on a per-triangle basis. Additionally, an optimized cache validity evaluation for reuse of geometry persisting in the GPU memory was introduced. The results reported improved preprocess time, reduction of triangles per frames as well as a gain in rendering performance.

We see the scope of future work in extending our scheme for out-of-core processing. Moreover, we would further harness heterogeneous hardware platforms in general (for instance, OpenCL). This includes the a-priori error estimation and view-dependent refinement. A further point of interest is to adapt our hybrid method to TINs due to the high adaption capabilities.

REFERENCES

Bösch, J., Goswami, P., and Pajarola, R. (2009). RASTeR: Simple and Efficient Terrain Rendering on the GPU. In *Eurographics 2009 - Areas Papers*, pages 35–42. Eurographics Association.

Cignoni, P., Ganovelli, F., Gobbetti, E., Marton, F., Ponchio, F., and Scopigno, R. (2003). BDAM - batched dynamic adaptive meshes for high performance terrain visualization. *Computer Graphics Forum*, 22:505–514.

Cignoni, P., Ganovelli, F., Gobbetti, E., Marton, F., Ponchio, F., and Scopigno, R. (2005). Batched multi triangulation. *IEEE Visualization, 2005. VIS 05*, pages 207–214.

Cignoni, P., Puppo, E., and Scopigno, R. (1997). Representation and visualization of terrain surfaces at variable resolution. In *The Visual Computer*, pages 50–68.

de Boer, W. H. (2000). Fast terrain rendering using geometrical mipmapping.

Duchaineau, M. A., Wolinsky, M., Sigeti, D. E., Miller, M. C., Aldrich, C., and Mineev-Weinstein, M. B. (1997). Roaming terrain: real-time optimally adapting meshes. In *IEEE Visualization*, pages 81–88.

Evans, W., Kirkpatrick, D., and Townsend, G. (2001a). Right-triangulated irregular networks. *Algorithmica*, 30(2):264–286.

Evans, W., Kirkpatrick, D., and Townsend, G. (2001b). Right-triangulated irregular networks. *Algorithmica*, 30:264–286.

Hoppe, H. (1997). View-dependent refinement of progressive meshes. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 189–198, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.

Hoppe, H. (1998). Smooth view-dependent level-of-detail control and its application to terrain rendering. *Visualization Conference, IEEE*, 0:35.

Hu, L., Sander, P., and Hoppe, H. (2009). Parallel view-dependent refinement of progressive meshes. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pages 169–176. ACM New York, NY, USA.

Hwa, L. M., Duchaineau, M. A., and Joy, K. I. (2004). Adaptive 4-8 texture hierarchies. In *VIS '04: Proceedings of the conference on Visualization '04*, pages 219–226, Washington, DC, USA. IEEE Computer Society.

Levenberg, J. (2002). Fast view-dependent level-of-detail rendering using cached geometry. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 259–266, Washington, DC, USA. IEEE Computer Society.

Lindstrom, P., Koller, D., Ribarsky, W., Hodges, L. F., Faust, N., and Turner, G. A. (1996). Real-time, continuous level of detail rendering of height fields. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 109–118, New York, NY, USA. ACM.

Lindstrom, P. and Pascucci, V. (2002). Terrain simplification simplified: A general framework for view-dependent out-of-core visualization. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):239–254.

Löffler, F., Rybacki, S., and Schumann, H. (2009). Error-Bounded GPU-Supported Terrain Visualisation. In *WSCG'09 Communication Papers Proceedings*, pages 47–54. University of West Bohemia.

Pajarola, R. (1998). Large scale terrain visualization using the restricted quadtree triangulation. *Visualization Conference, IEEE*, 0:19.

Pajarola, R. and Gobbetti, E. (2007). Survey on semi-regular multiresolution models for interactive terrain rendering. *The Visual Computer*, 23(8):583–605.

Pomeranz, A. (2000). *ROAM Using Surface Triangle Clusters (RUSTiC)*. PhD thesis, UNIVERSITY OF CALIFORNIA.

Puppo, E. (1996). Variable resolution terrain surfaces. In *Proceedings of the 8th Canadian Conference on Computational Geometry*, pages 202–210. Carleton University Press.

Röttger, S., Heidrich, W., and Seidel, H.-P. (1998). Real-time generation of continuous levels of detail for height fields. pages 315–322.