Seamless Patches for GPU-Based Terrain Rendering

Yotam Livny
Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel
livnyy@cs.bgu.ac.il

Zvi Kogan
Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel
koganz@cs.bgu.ac.il

Jihad El-Sana
Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel
el-sana@cs.bgu.ac.il

ABSTRACT

In this paper we present a novel approach for interactive rendering of large terrain datasets which is based on subdividing the terrain into rectangular patches at different resolutions. Each patch is represented by four triangular tiles which can be at different resolutions; and four strips which are used to stitch the four tiles in a seamless manner. As a result, our scheme maintains resolution changes within patches and not across patches. At runtime, the terrain patches are used to construct a level of detail based on view-parameters. The selected level of detail only includes the layout of the patches and the resolutions at boundary edges. Since adjacent patches agree on the resolution of common edges, the resulted mesh does not include any cracks or degenerate triangles. The GPU generates the meshes of the patches by using scaled instances of cached tiles and assigning elevation for each vertex from the cached textures. Our algorithm manages to achieve quality images at high frame rates while providing seamless transition between different levels of detail.

Keywords: Terrain visualization, view-dependent rendering, and level of detail

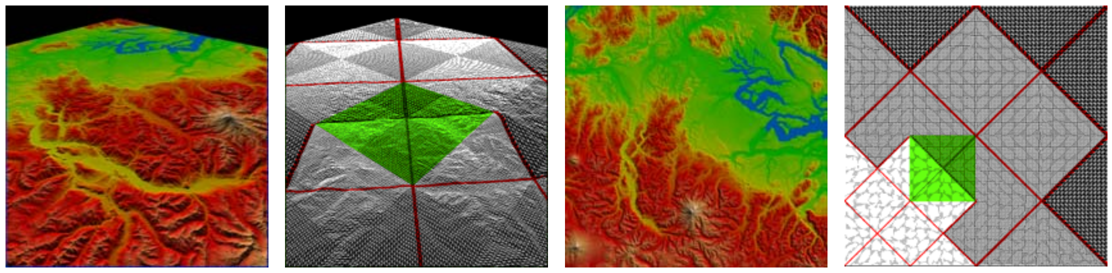

Figure 1: Terrain rendering using seamless patches. (a) A selected view; (b) The wire-frame of (*a*), where the green region marks one patch; (c) Top view of *(a)*; (d) The wire-frame of *(c)* with the same marked patch.

1. INTRODUCTION

Interactive visualization of landscapes and outdoor graphics environments is important for graphics applications such as computer games, flight simulators, and virtual exploration of remote planets. Terrains and height field geometry are vital components of these virtual environments.

The rapid development in acquisition of topographic maps and cartography has led to the generation of large terrain datasets that contain billions of samples. Such terrains exceed the rendering capability of available graphics hardware, thus reducing the geometric complexity of these datasets is mandatory for interactivity. Adjusting the terrain triangulation in a view-dependent manner is a common approach for interactive terrain rendering. Furthermore, adaptive level-of-detail rendering not only simplifies the geometry, but also manages to reduce aliasing artifacts that may result from rendering uniform dense triangulation.

The challenges of interactive terrain rendering have attracted the interest of researchers for several decades and extensive research has been done (see Section 2). Classic level-of-detail rendering schemes generate, usually off-line, multiresolution hierarchies which are used at runtime to guide the selection of appropriate levels of detail based on view-parameters. Some of these approaches utilize temporal coherence among consecutive frames by adaptively simplifying or refining the geometry of a frame to produce the next frame's geometry. Other approaches generate the geometry for each frame independent of its previous frames. These approaches have managed to accelerate the rendering of large terrains, but they were not able to maintain the improvement rate as the GPUs grow faster. In addition, generation of the frame's geometry is performed by executing refine and simplify operations on the CPU, which often fails to complete these computations within the duration of one frame. This geometry, which is transferred to the graphics hardware at each frame, often exceeds the bandwidth of the communication channel and results in unacceptably low frame rates.

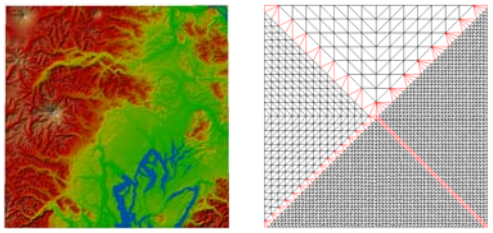
Figure 2: An image of one patch (left) and its wire-frame showing the stitching strips (right).

To reduce computation load on the busy CPU, several approaches partition the terrain into patches at different resolutions. At runtime these patches are stitched together to generate the appropriate levels of detail, which are then transmitted to the graphics hardware. Stitching these patches in a seamless manner is the main challenge for these approaches. Introducing degenerate triangles and dependencies among patches are used to handle these problems. However, these solutions may introduce visual artifacts or require additional random-access memory references.

To reduce data transmission between CPU and GPU, several algorithms use cached templates and quadric terrain elevation maps to generate geometry within the GPU. These algorithms often rely on triangular templates, which do not fit the rectangular texture interfaces and, hence, impose additional complexity in maintaining and storing textures.

In this paper we present a novel approach for interactive rendering of large terrain datasets, which is designed to prevent the above limitations of previous algorithms. Our approach subdivides the terrain into rectangular patches at different resolutions as shown in Figure 1. Each patch is represented by four triangular tiles that can be at different predetermined discrete resolutions and are stitched together by four strips as shown in Figure 2. Since the number of different resolutions is very small, the number of required patterns of stitching strips is also very small.

At runtime, these patches are used to construct the appropriate level of detail based on view-parameters.

The selected levels of detail do not include any geometry; instead they only include the layout of the patches and the resolutions along their boundaries. The resolutions along the boundaries are used to guide the selection of the adequate tiles and strips to cover each patch without the need to query adjacent patches. Since adjacent patches agree on the resolution of the shared edges, the generated mesh does not include any cracks or degenerate triangles. Scaled templates of the cached tiles are used to generate the geometry, within the GPU, based on the boundary resolution. The vertex and fragment processors fetch and assign elevation and color for each vertex using the cached textures. To handle large terrain datasets, we provide external texture memory support that caches the necessary displacement and color maps in the GPU's memory.

Our approach provides a number of advantages over previous terrain rendering schemes. The level of detail in each patch is determined without querying adjacent patches. Such a scheme saves unnecessary random-access memory references. The rendered mesh does not include any degenerate or sliver triangles, since our approach assures the same triangulation on the two sides of each boundary edge. In addition, it uses an implicit hierarchical representation that maintains the structure of the different patches in runtime. Furthermore, it reduces communication overhead as a result of transmitting only the layout of patches to the GPU at each frame, and using predetermined planar triangular tiles, which are cached in texture memory, to generate the selected level-of-detail representation. Therefore, only elevation values are transmitted to the graphics hardware in each frame.

In the rest of this paper we briefly overview related work in terrain rendering. Then we discuss our novel approach, followed by implementation details and experimental results. Finally, we draw some conclusions and suggest directions for future work.

2. RELATED WORK

In this section we briefly overview related work in level-of-detail terrain rendering. We focus on approaches that utilize the special properties of height-field datasets.

General level-of-detail rendering algorithms represent terrains as triangulated meshes. They usually utilize temporal coherence and manage to achieve the best approximation of the terrain for given view-parameters and triangle budget. However, these algorithms require the maintenance of mesh adjacency and validation of refinement dependences at each frame.

Level-of-detail algorithms for height-field datasets are based on regular grid representation. They utilize the longest edge bisection scheme to simplify memory layout by using a restricted quadtree triangulation [Bao04a, Paj98a], triangle bintrees [Duc97a, Lin96a], or hierarchies of right triangles [Eva01a, Lin02a]. However, updating the mesh at each frame prevents the use of efficient rendering schemes, such as geometry caching.

To utilize efficient rendering schemes, several approaches partition the terrain into square patches at different resolutions. At runtime the appropriate patches are selected, stitched together, and rendered [Hit93a, Paj98a, Pom00a]. Cignoni *et al.* [Cig04a] and Yoon *et al.* [Yoo05a] have developed similar approaches for general 3D models. The main challenge for these approaches is to stitch the boundaries of the appropriate patches seamlessly.

To overcome this communication bottleneck several algorithms have utilized cached geometry. Various approaches cache triangulated regions in texture memory [Cig03a, Cig03b, Lar03a, Lev02a], while others exploit the geometric locality to maximize the efficiency of the cache [Hop99a]. Terrains usually compensate small geometric details by textures and as a result, they are often accompanied by huge texture maps. Tanner *et al.* [Tan98a] have introduced the texture clipmaps hierarchy, and Döllner *et al.* [Dol00a] have developed a more general hierarchy to handle large texture maps. Caching techniques enable fast transfer of geometry and texture to graphics hardware. However, cache memory is limited, thus large datasets may still involve an overhead in communication between CPU and cache memory.

Cook [Coo84a] introduced the displacement maps that represent elevation maps as vertex textures. Other frameworks for displacement maps on programmable graphics hardware have been suggested by [Dog00a, Gum99a, Los04a, Mou02a]. Although these approaches are not implemented at GPU, they are based on designs which prefer many simple computations over a few complicated ones.

The advances in graphics hardware and its programmability have driven the development of a new generation of level-of-detail rendering algorithms. Losasso *et al.* [Los03a] and Bolz and Schröder [Bol05a] used the fragment processor to perform mesh subdivision. Southern and Gain [Sou03a] and Larsen and Christensen [Lar03a] used the vertex processor to interpolate different resolution meshes in a view-dependent manner. Wagner [Wag04a] and Hwa *et al.* [Hwa04a] used GPU-based geomorphs to render terrain patches of different resolutions. Dachsbacher and Stamminger [Dac04a] used GPU programmability to generate and render procedural details for terrains at runtime. Schneider and Westermann [Sch06a] suggested progressive transmission to reduce the data transfer between CPU and GPU.

Geometry clipmaps algorithm [Asi05a] stores the surface triangulation layout in a view-dependent manner. In each frame, the visible part of the triangulation is sent to the GPU and modified according to the uploaded elevation and color maps. However, this algorithm does not perform local adaptivity, and the transition between levels of detail is not smooth and may result in cracks. The cracks problem is resolved by inserting degenerate triangles, but such triangles may generate visual artifacts.

3. OUR APPROACH

In this section we present our novel algorithm for interactive terrain rendering. It partitions the terrain into rectangular patches and utilizes advanced features of graphics hardware, such as programmability, displacement mapping, and geometry caching. Our algorithm involves a light preprocessing stage, in which it generates hierarchies of elevation maps and color textures and stores them in main memory. In our patch scheme, the coexistence of different discrete geometry resolutions within the same patch enables seamless stitching (without cracks or degenerate triangles) of adjacent patches. In each frame our algorithm uses an implicit patch hierarchy to select a set of appropriate patches (for rendering) and determine the resolution on their boundaries based on view parameters. The resolution of each patch is determined based on its four boundary edges and without the need to query its adjacent patches.

Patch Scheme

Previous terrain rendering algorithms use either triangular or rectangular patches for view-dependent level-of-detail rendering. On one hand, algorithms that use rectangular patches assign constant resolution over the entire patch, and hence prevent local adaptivity and impose severe difficulty in stitching adjacent patches. On the other hand, algorithms that use triangular patches enable easier stitching schemes and provide better local adaptivity, but they suffer incompatibility with texture rectangular interface and complicate texture management. Our patch scheme combines the advantages of the two approaches; it subdivides the terrain into rectangular patches which consist of triangular tiles that allow different resolutions to coexist within one patch. Such a scheme provides limited local adaptivity and enables the stitching of adjacent patches in a seamless manner.

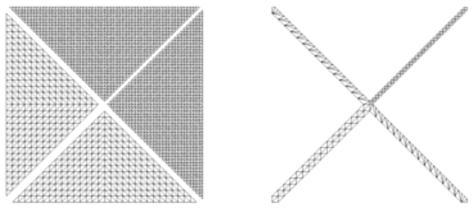
(a) (b)

Figure 3: The components of one patch. (a) The image of four tiles. (b) The image of four strips.

In our scheme, a patch is arranged as four tessellated triangular regions which are determined by the two diagonals of the rectangular patch (see Figure 3). We shall refer to these tessellated triangular regions as *triangular tiles* (or simply *tiles*). The four tiles can have different resolutions which are selected from a predefined set of uniform resolutions. One could treat these tiles as discrete levels of detail of the same tile. Within a patch, the triangular tiles are stitched together by using predefined strips (refer to Figure 4). Since the number of different resolutions for the tiles is usually small – *2* to *4* – the number of different stitching strips is also very small. Six strip types are required to stitch tiles at three different resolutions.

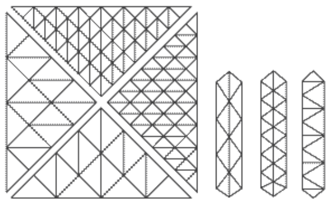
Figure 4: Triangular tiles at two different resolutions and the required stitching strips.

We have chosen to adopt tile resolutions at consecutive powers of two to comply with the mipmap resolutions and meet the requirement of *Claim 1* (see Level of Detail section below).

Patch Hierarchy

The patch hierarchy is constructed top-down by subdividing each patch into $R \times R$ children patches, where R is the branching factor of the hierarchy. The branching factor is determined by the number of different resolutions for tiles and equal to the ratio between the smallest and largest resolutions. For example, *2* and *3* resolutions require branching factors of *2* and *4*, respectively. This relation ensures seamless stitching among adjacent patches and absence of cracks.

The patch hierarchy does not store any geometry; instead it stores the position and dimension of each patch with respect to the terrain. Therefore, it easily fits in local memory, even for very large terrains. In practice, there is no need to implement the hierarchy explicitly, and therefore in our current implementation we use implicit hierarchy.

Runtime Rendering

At runtime, the patch hierarchy is used to guide the selection of the various levels of detail based on view-parameters. In each frame, the patch hierarchy is traversed in a top-down manner to select a set of active patches that form the appropriate level of detail. The traversal process starts from the root and for each visited patch τ an error metric is computed. If the error is too large, with respect to the view-parameters, the children of the patch τ are traversed. Otherwise, the resolutions of boundary edges are computed and the patch is added to the stream of active patches.

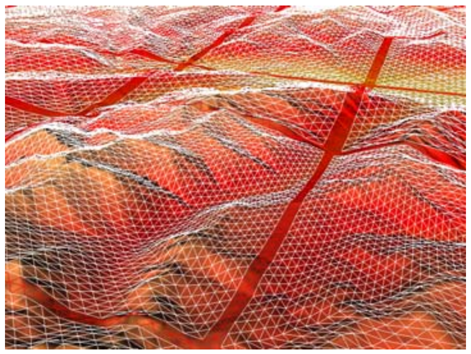
Figure 5: A terrain view with a wire-frame on top. The meshes of triangular tiles appear in white color and the strips appear in red.

Single-unit-size meshes that represent each resolution of the tiles and stitching strips (3 tiles and 6 strips for three different resolutions) are cached in texture memory. In each frame, the active patches are streamed to the graphics hardware for rendering. The light representation of active patches by their 2D enclosing rectangle contributes to the dramatic reduction on the CPU-GPU communication load. The resolutions at boundary edges (of patches) are discretized to match the resolution of the predefined triangular tiles. The resolution at the boundary edges is enough to determine the tiles and strips required to cover the patch τ in a straight forward manner (see Figure 5). The cached instances of the selected tiles and strips are transformed to match the enclosing rectangle of the patch, which selects its tiles without querying any of its adjacent patches. Since each two adjacent patches agree on the resolution of the common edge, the stitching of adjacent patches is smooth and does not include cracks or degenerate triangles.

Tiling a patch with triangular tiles produces a planar mesh without elevation or color components. These components are assigned (for each vertex) by the vertex and fragment processors, which use *x* and *y* coordinates of a received vertex to fetch and assign the appropriate elevation/color from cached textures.

Level of Detail

The level of detail of a patch is represented by the resolutions of its tiles which are determined by the resolution at boundary edges. The resolution of an edge is computed based on its length *l* and the distance *d* from the viewpoint by using Equation 1, where ρ is a precision factor. If ε is larger than 1, the patch is split to its children, otherwise the resolution of the edge is determined by εR_{max} rounded up to the closest resolution, where R_{max} is the highest available resolution.

$$\varepsilon = \rho \frac{l}{d} \qquad (1)$$

The scaling factor is used to resize a tile to match the patch's enclosing rectangle and select the appropriate texture level from which the elevation and color values are fetched.

Claim 1: *The generated mesh does not include cracks, which means that any two adjacent tiles agree on the resolution of the common edge.*

Proof: Without loss of generality we prove the claim for two resolutions and quadtree subdivision. We distinguish between two cases:

I. The two adjacent patches have the same dimensions: Since the two patches have the same dimensions, they have the same enclosing rectangle and share a common edge along an entire side. By selecting the same tile on the two sides of the shared edge, the two patches are stitched seamlessly.

II. The two adjacent patches are in different dimensions, which means that the edge belongs to one patch on one side and two patches on the other side (see the edge $\overline{AB}$ in Figure 6): We first show that the tile $\overline{ABJ}$ gets the highest resolution *R2*. The patch $\overline{ABCD}$ has split to four children, which means that one of its edges has required a resolution higher than *R2* (beyond the available resolution); let this edge be $\overline{CD}$. Let *l* be the length of the edge $\overline{AB}$ and the distances of the edges $\overline{AB}$ and $\overline{CD}$ from the viewpoint are d_{far} and d_{near}, respectively. Based on Equation 1, $\rho \cdot l > d_{near}$ holds as a result of assuming that the patch $\overline{ABCD}$ has split into its four children, then:

$$\frac{\rho \cdot l}{d_{near}} > 1 \Rightarrow \frac{\rho \cdot l}{l + d_{near}} > \frac{1}{2} \Rightarrow \frac{\rho \cdot l}{d_{far}} > \frac{1}{2}; d_{far} \le l + d_{near}$$

Therefore, the edge $\overline{AB}$ is assigned the resolution *R2*, and the edges $\overline{AE}$ and $\overline{EB}$ are assigned the resolution *R1*. Our algorithm assigns resolution *R1* to edges with error values ε in the range *[0, 0.5]* and *R2* to those with error values in the range *(0.5, 1.0]*. For that reason, the difference between adjacent patches is at most one level (in the case of two different resolutions). ■

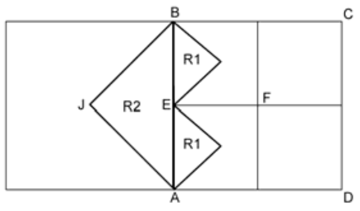

Figure 6: Stitching tiles at two different levels of detail.

Texture Pyramid

Terrain datasets are usually represented by elevation maps and color textures, which store the properties of vertices in the original terrain. We use multiple-level texture pyramids at successive powers of two (similar to mipmaps) to support level-of-detail rendering. These texture pyramids are used at runtime to achieve faithful sampling of the textures for the vertices of each tile. Since these multiple-level pyramids are similar to mipmaps, we could let the hardware construct them. Then at runtime, the vertex processor determines from which level to select the values. However, such an approach does not work when the terrain size exceeds the capacity of the base level of the mipmaps [Los04a].

For large terrains, the multiple-level texture pyramids are constructed once by the CPU before being transferred for caching in the texture memory. We start with the original texture, which represents the most detailed level, and each new level is generated from the previous one by reducing the resolution by half at each dimension. The pixels in the generated level are computed by interpolating the four corresponding pixels of the previous level.

Note that elevation and color values of a vertex are selected from different levels of the hierarchies based on the geometric level of detail of the processed tile.

4. IMPLEMENTATION DETAILS

In our current implementation we do not construct the patch hierarchy explicitly; instead, an implicit representation is used. The root of the hierarchy is the coarsest level of detail that fits in texture memory and matches the interactive rendering capability of the graphics hardware. Therefore, the height of the hierarchy can be easily determined based on the hardware capabilities and the predefined branching factor. Note that the 2D bounding rectangle of the root is the same as that of the original terrain. Also

Dataset Size	View Region	ρ Factor	With Frustum Culling					Without Frustum Culling			
			Triangles	Traversed	Rendered	Culled	FPS	Triangles	Traversed	Rendered	FPS
4Kx4K	Edge	ρ_0	172696	109	46	36	283	331428	121	91	156
4Kx4K	Middle	ρ_0	138668	109	33	49	380	403896	141	106	138
16Kx16K	Edge	ρ_0	180731	100	50	27	271	338240	123	93	153
16Kx16K	Middle	ρ_0	148200	126	39	58	354	389197	151	107	135
4Kx4K	Edge	$2\rho_0$	354248	137	66	37	138	763864	189	142	69
4Kx4K	Middle	$2\rho_0$	330480	133	56	44	156	1133796	265	199	52
16Kx16K	Edge	$2\rho_0$	422358	179	82	74	112	915190	213	160	56
16Kx16K	Middle	$2\rho_0$	414966	152	70	63	112	1359642	340	231	43

Table 1: Runtime performance.

recall that the patch hierarchy does not store any mesh geometry or pixel information. The subdivision of a patch into its children is performed by several shift instructions within the CPU. The traversal of the implicit patch hierarchy is performed similar to the explicit one and is often more efficient as a result of avoiding random memory access to fetch the children patches. We found that traversing the patch hierarchy is negligible compared to the rendering time as shown in the CPU column on Table 2.

View-frustum culling is performed by the CPU during the traversal which determines the set of active patches. For each patch τ which requires further subdivision to reach the appropriate level of detail, we test its children patches against the view-frustum only if τ intersects the boundary of the view-frustum. If the patch τ is entirely included within the view-frustum, then all its children patches are also within the view-frustum. If τ intersects the view-frustum's boundary, we test and mark each of its children patches as to whether it is inside, outside, or intersecting the view-frustum. Outside patches are culled and they are not processed further.

The meshes that represent the different resolutions tiles and strips are cached in texture memory. At runtime, these meshes are used to tile the selected patches. Since the number and the size of these meshes are small (3 tiles and 6 strips are required to support three different resolutions within a patch), we store four orientations of each tile and each strip to avoid rotation and mirroring of these meshes at runtime.

To handle large terrain datasets we have implemented an out-of-core support similar to the one proposed by Losasso and Hoppe [Los04a]. This scheme stores the texture pyramids in main memory and caches in texture memory only the portions necessary for rendering. The updates of the cached textures are performed in an active manner by loading "L-Shape" regions into texture memory, as early as they are required.

In earlier algorithms, the CPU needs to send three coordinates for an uncached vertex to place it in the model space. Our algorithm utilizes hardware supported displacement mapping, and thus the CPU sends only the elevation value for each vertex. The other two coordinates are generated by the GPU in a parametric manner using the terrain grid structure. This technique reduces the data transfer at runtime from three coordinates to one coordinate for each vertex. The elevation components are uploaded into the vertex texture using Fragment Buffer Object extensions (FBO).

5. RESULTS

We have tested our implementation on an AMD Athlon 3500 with 1GB memory, and an *nVidia* GeForce 7800 GTX graphics card with 256M texture memory using Puget Sound and Grand Canyon terrain datasets. In this section we report and analyze selected entries of these results.

The performances of our algorithm are summarized in Table 1. For each dataset we view different regions of the terrain to capture the various processing patterns. In each row we report the terrain size, viewed region, precision factor, and performance with and without view-frustum culling. We record two options for the viewed regions: *edge* and *middle,* which refer to flying near an edge and inside the terrain, respectively. The precision factor (see also Equation 1) $2\rho_0$ selects more detailed levels than the levels selected by ρ_0. In the performance columns we report the number of rendered triangles (*Triangle* column), the number of traversed patches (*Traversed* column), the number of rendered patches (*Rendered* column), the number of culled patches (*Culled* column), and the frame rates. The view-frustum culling doubles the performances when flying on the edge of the terrain and triples it in general. Our algorithm manages to achieve quality images at high frame rates, as can be seen in Table 1. The frame rates depend mainly on the number of triangles. The first row shows 156 FPS without view frustum culling for about 330K triangles and 91 rendering patches, and the sixth row reports the same FPS with view frustum culling and 56 patches. Such observation reveals that patch selection is negligible with respect to the total rendering time. Note that patch selection also includes view-frustum culling and transmission of active patches list.

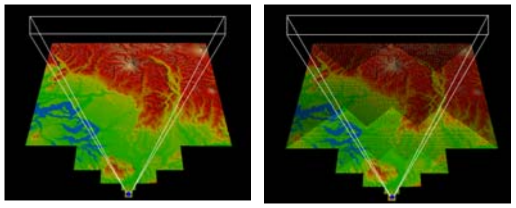
Figure 7: View-frustum culling: A shaded view (left) and its wire-frame representation (right).

We compared the results of our algorithm with the results of three known terrain rendering algorithms. To bring all the results to a common base, we have estimated the expected performance of these algorithms on our machine based on the reported results and the used machine's hardware. To present a reliable approximation, we measure only triangles that are actually processed by the graphics hardware. On comparable hardware we expect that BDAM [Cig03a], Clipmap [Asi05a], and the algorithm suggested in [Hwa04a] will achieve about *46M*, *44M*, and *43M* textured triangles per second, respectively. Our algorithm manages to achieve *53M* textured triangles per second on average. These numbers show that the simplicity of our GPU code with the advantages of displacement map functionality provides encouraging performance.

Configuration			Time		
Triangles	Rendered	Culled	CPU (μs)	GPU (ms)	FPS
88986	18	34	17.54	1.85	583
96246	22	13	22.57	1.98	545
119054	27	18	26.88	2.58	418
148200	39	58	30.67	3.04	354
180731	50	27	34.84	3.98	271
230372	47	23	39.06	5.68	190

Table 2: Hardware performance analysis

The contribution of the CPU and the GPU to the performance of the algorithm is shown in Table 2. The first three columns of each row represent the configuration of a frame, which includes the number of rendered triangles, rendered patches, and culled patches. The fourth and fifth columns report the CPU and the GPU processing time, respectively. The CPU load is tiny and has almost no influence on the frame rates for two main reasons – the selection of the active patches (by the CPU) is very light and the CPU runs parallel to the GPU. These conclusions are also supported by the results shown in Table 2. These results also show that our algorithm will benefit from the current trend in improving GPU rates.

Figure 7 shows the shaded and wire-frame representations of a terrain view after applying view-frustum culling. Figures 8 and 9 were generated from a Puget Sound terrain dataset using our algorithm at different precision factors ρ. In each figure, image (a) shows a shaded view that depicts image quality, image (b) shows the wire-frame representation that illustrates the triangular tiles in white color and the stitching strips in red.

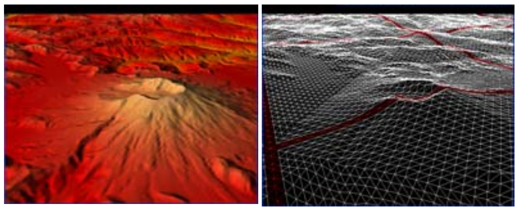
Figure 8: A terrain view at $\rho = \rho_0$. (a) A shaded surface. (b) Tiles in white and strips in red.

6. SUMMARY AND FUTURE WORK

We have presented a novel approach for interactive terrain rendering that reduces the load on the CPU, utilizes texture memory, and leverages advanced features of the GPU. The terrain is subdivided into rectangular patches on the fly. Each patch is represented by four triangular tiles at different resolutions which are stitched together using four strips. At runtime the CPU selects the appropriate patches based on view-parameters and determines the resolution at their boundaries. The different tiles and stitching strips are cached in texture memory and used to tile each patch according to its boundary resolution. Multiresolution levels of color textures and displacement maps are also cached in texture memory and used by the vertex and fragment processors to assign the elevation and color for each vertex.

Our approach balances computation load among the CPU and GPU and dramatically reduces the communication between them. Adjacent patches are stitched in a seamless manner without cracks or degenerate triangles, since they agree on the resolution of the common edge. Furthermore, each patch determines its own resolution without querying its adjacent patches; it simply selects the different tiles that comply with its boundary resolution. The use of tiles provides limited local adaptivity which contributes to the smoothness of the generated mesh.

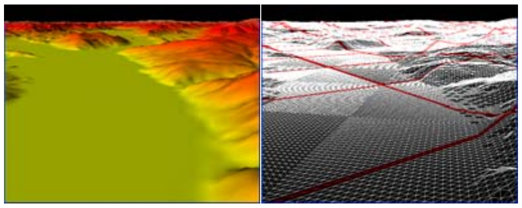
Figure 9: A terrain view at $\rho=2\rho_0$. (a) A shaded surface. (b) Tiles in white and strips in red.

Our algorithm performances are strongly influenced by the speed of vertex pipelines. The algorithm relies on the vertex fetch operation which enables the vertex processor to access texture memory. However, the fetch operations within vertex processors are not yet optimized. We predict that future development in vertex processor hardware will lead to impressive improvement on the performance of our algorithm.

We see the scope of future work in extending the idea of independent patches to general 3D models. Such development will provide view-dependent rendering for large datasets in a seamless manner without imposing dependencies among adjacent patches. Moreover, our suggested approach generates patch geometry within the GPU, and hence can not utilize temporal coherence among consecutive frames. Utilizing temporal coherence within the GPU could contribute to further performance improvements.

7. ACKNOWLEDGMENTS

Our research and particularly this work are supported by the Lynn and William Frankel Center for Computer Sciences, the Israel Ministry of Science and Technology, and the Tuman fund.

8. REFERENCES

[Asi05a] Asirvatham A. and Hoppe H. Terrain rendering using GPU-based geometry clipmaps. GPU Gems 2, pages 27–45, 2005.

[Bao04a] Bao X., Pajarola R., and Shafae M., Smart: An efficient technique for massive terrain visualization from out-of-core. In Proceedings of Vision, Modeling and Visualization '04, pages 413–420, 2004.

[Bol05a] Bolz J. and Schröder P. Evaluation of subdivision surfaces on programmable graphics hardware. Submitted, 2005.

[Paj98a] Pajarola R., Large scale terrain visualization using the restricted quadtree triangulation. In Proceedings of Visualization '98, pages 19–26, 1998.

[Cig03a] Cignoni P., Ganovelli F., Gobbetti E., Marton F., Ponchio F., and Scopigno R. BDAM – batched dynamic adaptive meshes for high performance terrain visualization. Computer Graphics Forum, 22(3):505–514, 2003.

[Cig03b] Cignoni P., Ganovelli F., Gobbetti E., Marton F., Ponchio F., and Scopigno R. Planet-sized batched dynamic adaptive meshes. In Proceedings of Visualization '03, pages 147–155, 2003.

[Cig04a] Cignoni P., Ganovelli F., Gobbetti E., Marton F., Ponchio F., and Scopigno R. Adaptive tetrapuzzles: efficient out-of-core construction and visualization of gigantic multiresolution polygonal models. ACM Trans. Graph., 23(3):796–803, 2004.

[Coo84a] Cook L., Shade trees. Computer Graphics, 18(3):223–231, 1984.

[Dac04a] Dachsbacher C., and Stamminger M. Rendering procedural terrain by geometry image warping. In Eurographics Symposium in Geometry Processing, pages 138–145, 2004.

[Dog00a] Doggett M., and Hirche J. Adaptive view dependent tessellation of displacement maps. In HWWS '00: Proceedings of the workshop on Graphics hardware, pages 59–66, 2000.

[Dol00a] Döllner J., Baumann K., and Hinrichs K. Texturing techniques for terrain visualization. In Proceedings of Visualization '00, pages 227–234, 2000.

[Duc97a] Duchainear M., Wolinsky M., Sigeti D., M. Miller, C. Aldrich, and Mineev-Weinstein M., ROAMing terrain: Real-time optimally adapting meshes. In Proceedings of Visualization '97, pages 81–88, 1997.

[Eva01a] Evans S., Kirkpatrick D., and Townsend G. Right triangulated irregular networks. Algorithmica, 30(2):264–286,2001.

[Lin96a] Lindstrom P., Koller D., Ribarsky W., Hodges L., Faust N., and Turner G. Real-time, continuous level ofdetail rendering of height fields. In SIGGRAPH '96,pages 109–118, 1996.

[Lin02a] Lindstrom P. and Pascucci V. Terrain simplification simplified: A general framework for view-dependent out-of-core visualization. IEEE Transactions on Visualization and Computer Graphics, 8(3):239–254, 2002.

[Gum99a] Gumhold S. and H¨uttner T. Multiresolution rendering with displacement mapping. In HWWS '99: Proceedings of the workshop on graphics hardware, pages55–66, 1999.

[Hit93a] Hitchner L. and McGreevy M. Methods for user-based reduction of model complexity for virtual planetary exploration. In SPIE 1913, pages 622–636, 1993.

[Hop99a] Hoppe H. Optimization of mesh locality for transparent vertex caching. In SIGGRAPH '99, pages 269–276, 1999.

[Hwa04a] Hwa L. M., Duchaineau M., and Joy K. I.. Adaptive 4-8 texture hierarchies. In Proceedings of Visualization 2004, pages 219–226, 2004.

[Lar03a] Lario R., Pajarola R., and Tirado F. Hyper-block quadtree based triangulated irregular networks. In Proceedings of IASTED VIIP, pages 733–738, 2003.

[Lev02a] Levenberg J. Fast view-dependent level-of-detail rendering using cached geometry. In Proceedings of Visualization '02, pages 259–266, 2002.

[Los03a] Losasso F., Hoppe H., Schaefer S., and Warren J. Smooth geometry images. In Eurographics Symposium in Geometry Processing, pages 138–145, 2003.

[Los04a] Losasso F. and Hoppe H. Geometry clipmaps: terrain rendering using nested regular grids. ACM Trans. Graph., 23(3):769–776, 2004.

[Mou02a] Moule K. and McCool M. Efficient bounded adaptive tessellation of displacement maps. In Proceedings of Graphics Interface, pages 171–180, 2002.

[Pom00a] Pomeranz A. Roam using triangle clusters (RUSTiC). Master's thesis, 2000.

[Sch06a] Schneider J. and Westermann R. Gpu-friendly high-quality terrain rendering. Journal of WSCG, 14(1-3):49–56, 2006.

[Sou03a] Southern R., and Gain J., Creation and control of real-time continuous level of detail on programmable graphics hardware. Computer Graphics Forum, 22(1):35–48, 2003.

[Tan98a] Tanner C., Migdal J. and Jones M. The clipmap: a virtual mipmap. In Proceedings of SIGGRAPH '98, pages 151–158, 1998.

[Wag04a] Wagner D. Terrain geomorphing in the vertex shader. Shader Programming Tips & Tricks with DirectX 9, 2004.

[Yoo05a] Yoon, S. Salomon, B. and Gayle, R. Quick-VDR: Out-of-core view-dependent rendering of gigantic models. IEEE Transactions on Visualization and Computer Graphics, 11(4):369–382, 2005.