Real-time Terrain Modeling using CPU-GPU Coupled Computation

Adrien Bernhardt, André Maximo, Luiz Velho, Houssam Hnaidi, Marie-Paule Cani

▶ **To cite this version:**

Adrien Bernhardt, André Maximo, Luiz Velho, Houssam Hnaidi, Marie-Paule Cani. Real-time Terrain Modeling using CPU-GPU Coupled Computation. XXIV SIBGRAPI - 24th SIBGRAPI Conference on Graphics, Patterns and Images, Aug 2011, Maceio, Brazil. pp.64-71, 10.1109/SIBGRAPI.2011.28 . inria-00612291

HAL Id: inria-00612291
https://inria.hal.science/inria-00612291

Submitted on 28 Jul 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Real-time Terrain Modeling using CPU–GPU Coupled Computation

Adrien Bernhardt*, André Maximo†, Luiz Velho†, Houssam Hnaidi‡ and Marie-Paule Cani*
*INRIA, Grenoble Univ., France
†IMPA, Brazil
‡LIRIS, CNRS, Univ. Lyon 1, France

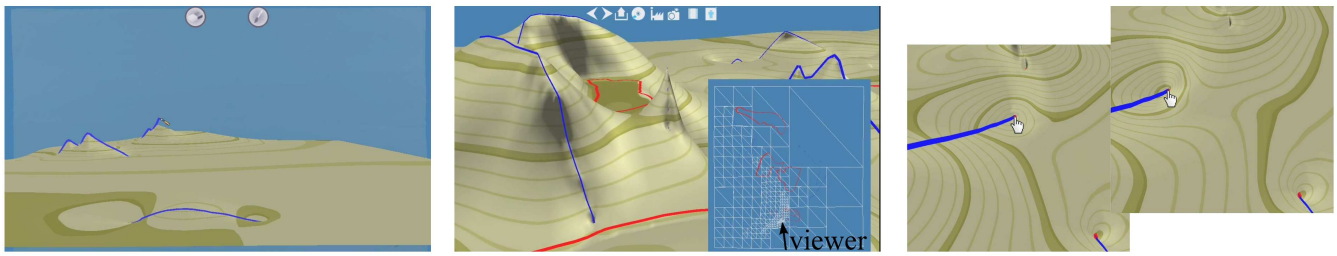


Fig. 1. Sketch-based terrain modeling example in our tool: as each stroke is drawn (left) or manipulated (right), the terrain is tessellated in the GPU to follow the stroke. In the CPU, the quadtree data structure (middle) controls the quadrilateral patches sent to the GPU.

***Abstract*—Motivated by the importance of having real-time feedback in sketch-based modeling tools, we present a framework for terrain edition capable of generating and displaying complex and high-resolution terrains. Our system is efficient and fast enough to allow the user to see the terrain morphing at the same time the drawing editing occurs. We have two types of editing interactions: the user can draw strokes creating elevations and crevices; and previous strokes can be interactively moved to different regions of the terrain. One interesting feature of our tool is that terrain primitives can be interactively manipulated similarly to primitives in vector-graphics tools. We achieve real-time performance in both modeling and rendering using a hybrid CPU–GPU coupled solution. We maintain a coarse version of the terrain geometry in the CPU by using a quadtree, while a fine version is produced in the GPU using tessellation shaders.**

***Keywords*-coupled computation; sketch-based modeling; terrain modeling; GPU tessellation.**

I. Introduction

Recent years have been marked with a striking evolution of sketch-based modeling tools [1], [2], [3], [4], which have the interesting property of benefiting greatly from real-time feedback while editing. As the object to be modeled increases in complexity, the modeling tool becomes more involving to develop. In the special case of terrain modeling, exemplified in figure 1, the challenge is to create in real-time realistic terrain features with intuitive strokes and guided landscape design.

Synthetic terrains are applied in many applications, such as: crowd and flight simulators, real movies using chroma-key effects, film animations, and computer games. Regardless of the application, synthetic terrains often have direct influence in the visual and artistic aspects to be accomplished. The common strategy to produce terrain models is by procedural generation [5], [6] through texture synthesis and height-field captured examples. The idea is to achieve a high level of realism through Digital Elevation Models (DEMs) of real terrains [7]. Although this strategy produces convincing results, it lacks controllability and restricts the result to variations of existing terrain primitives.

One way to improve controllability over the terrain generation is to use image samples from a procedural input [8]. The idea behind this technique is to allow the user to intuitively paint bumps at different resolution levels, enhancing the terrain modeling tool while maintaining realistic results. Another step further usability and control is to permit angles different than 90° when painting details on the terrain [9]. As the tool increases in controllability, the terrain modeling tool becomes more user-controlled and less automatic. The zenith of this process is an interface closer to pen-and-paper sketching [10], allowing total control over terrain primitives exchanging physical realism by artistic freedom.

Parallel to the question of balancing controllability and realistic results, terrain modeling tools can be analyzed by two different computational perspectives: CPU-based modeling strategies [11], [12] and GPU-based shading and solver techniques [13], [14], [15]. On the one hand, the CPU plays the role of a controller and its strategies focus mainly on integrating manual editing with complex deformations and simulations. On the other hand, the GPU offers massive parallelism and its techniques focus on shading landscapes and performing computational intensive tasks, such as solving diffusion equations. The gap between one unit and the other becomes clear when analyzing the data structure involved on each computation. While the CPU allows a broad range

of inter-connected data without penalizing access, the GPU requires coherent memory requests imposing data regularity.

Contribution: In this paper, we present a new modeling framework for terrain editing by *coupling* CPU–GPU computational perspectives. Differently from other approaches, discussed in section II, our modeling tool uses the CPU not only to control the initial level-of-detail (LOD) visualization but also to start and stop the terrain generation done by the GPU. The combination of the different computational granularities makes the data paths between the two units more complex, as explained in section III.

Our approach partitions the terrain with a view- and heightmap-dependent quadtree [16], as shown in figure 1 (middle). This lightweight data structure controls the quadrilateral patches to be rendered, serving two main purposes: it allows a first coarse LOD analysis of the terrain; and it frees the CPU from the costly task of generating the entire heightmap data.

Next, we employ the GPU to solely generate and maintain the terrain data on its own texture memory. The generation is accomplished by a fast multigrid solver [15] specifically designed for terrains. The generated data is constantly used in our approach to morph the original rendering of low-resolution quadtree patches in smooth high-resolution triangles through the graphics-card tessellation shaders. The tessellator allows for a second fine LOD analysis of the terrain, balancing the decision between the units. Although the whole data is updated and used by the GPU, the management is done by the CPU.

The final result of our coupled solution is a real-time terrain modeling tool based on sketches and capable of dynamically generate and visualize multiresolution heightmaps (see section IV). The framework presented is discussed in more details in section V, and section VI concludes our work giving future research directions.

II. Related Work

In many editing tools, especially sketch-based modeling, it is important to have real-time feedback to help improve the editing quality [17], [18], [19]. This importance is emphasized particularly in sketch-based terrain modeling, being able to see the terrain morphing at the same time the drawing editing occurs constitutes a great user experience, an example is shown in figure 1 (left).

The quality of the modeling tool depends mainly on three aspects: controllability, intuitiveness and responsiveness. The first makes the actual modeling possible, the second makes the tool more natural and easy to learn, and the third enhances the creativity process when the tool provides real-time feedback. One interesting example combining these three aspects is the FiberMesh [4]. In this tool, the user is able to create freeform surfaces with 3D curves in real-time with natural and unconstrained design. However, there are two drawbacks in FiberMesh: the high pre-computational cost required when adding or removing 3D curves; and the relative small models that the tool can handle. Like FiberMesh, our system builds the surface by interpolating curves but allowing real-time interaction with more complex models.

Terrain modeling tools also benefit from intuitiveness and natural interactions. The usage of sketch-based terrain modeling exploring this aspect was presented by Watanabe and Igarashi [20], and further inspired the work of Gain *et al.* [11]. In Gain *et al.*'s work, the user can draw strokes defining mountain silhouettes and modify previous strokes forming the base of mountains. Their CPU-based modeling framework generates terrain primitives extracting, at the same time, noise from the user strokes to create more realistic terrains. Although we do not extract noise from the user strokes in our tool, we have two types of editing interactions, in the same spirit of Gain *et al.*'s work: the user can draw strokes creating mountain elevations or crevices; and previous strokes can be interactively moved to different regions of the terrain, as shown in figure 1 (right). Moreover, we use a CPU–GPU coupled method to drastically improve the performance of our tool, generating terrains two orders of magnitude faster than Gain *et al.*'s work.

One work dealing with static real terrains and focused on rendering is the work of Losasso and Hoppe [21]. In their approach, called *geometry clipmaps*, they define a regular-grid hierarchy centered about the viewer instead of a world-space quadtree. This choice simplifies both their pyramid compression scheme and inter-level continuity while the viewer is moving. The geometry clipmaps uses the CPU to (1) construct LOD regular regions (see [21, figure 2]) based on viewer distance, (2) tessellate the regions in fine-to-coarse order and using triangle strips (see [21, figure 3]) to exploit hardware occlusion culling and vertex caching, (3) update the terrain maintained in CPU memory via decompression. There is one simple data path as the GPU is only used to interpolate height values for visual continuity in the vertex shader. In a revised version of this idea [22], the height-value interpolation is improved in the GPU by elevation-map upsampling, terrain detail synthesis and normal-map computation but remaining with a single data path between the units. In our approach, we are interested in dynamically changing terrains and use different techniques: a CPU-based quadtree data structure rather than a nested regular grid; and a GPU-based runtime terrain management for both tessellation and terrain modification.

Realistic terrains can also be created through modeling by employing simulation of natural phenomena. Anh *et al.* [13] presented a GPU-based method for hydraulic erosion simulation capable of simulating water transportation and erosion one order of magnitude faster than CPU-based methods. The work of Šťava *et al.* [23] presents a comprehensive set of GPU-accelerated erosion tools for real-time interactive terrain modeling, indicating the usage of graphics hardware for the solution of physically-based equations. Although in our modeling system we do not have simulation effects, we use the GPU computational power to generate the terrain by solving a different type of physically-based equation.

The terrain modeling in our approach is accomplished by combining the multigrid GPU solver of Hnaidi *et al.* [15] with an adaptive tessellation-based rendering shader capable of handling dynamic heightmaps. The main contribution of Hnaidi *et al.*'s work is to introduce a GPU-based multigrid diffusion

equation solver for terrains relying on Laplacian equations, which interpolates not only heights but also amplitude and frequency noise parameters. Our modeling tool uses Hnaidi *et al.*'s solver to allow an interactive edition and manipulation of complex terrain primitives.

III. Real-time Terrain Modeling

The main contribution of this paper is a real-time terrain modeling system, combining a novel CPU–GPU coupled solution with a natural sketch-based interaction. In this section, we describe both the coupled computation and this interaction.

A. CPU–GPU Coupled Computation

Creation of terrain models in real-time involves dealing with dynamically changing data that increases exponentially depending on the terrain resolution. However, high resolution is only interesting in parts of the terrain with a high level of detail. In our terrain modeling framework, presented in figure 2, we make use of this observation dividing the terrain generation process over a multiresolution pyramid. The base level of this pyramid, i.e. coarse resolution, is used whenever the details are low or do not appear in the current point-of-view. The top level of the pyramid, i.e. fine resolution, is used whenever the details are high and dominant in the current shading of the terrain model.

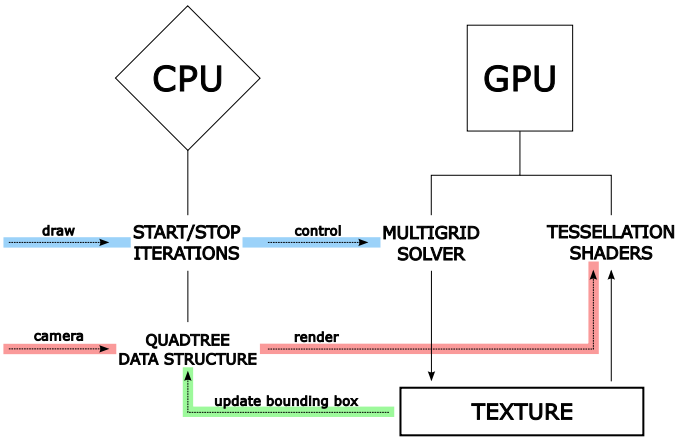


Fig. 2. The terrain modeling framework of our tool. While the CPU controls the multigrid solver depending on drawings and sends terrain patches to be rendered, the GPU generates the terrain in texture memory and tessellates its own produced heightmap. This setting requires three different data paths: adaptive rendering using a quadtree (red); solver iterations control (blue); and updating coarse-resolution details on the quadtree (green).

There are two main user interactions related to the multiresolution pyramid: *draw* of strokes controlling the terrain generation process; and *camera* movement changing the rendering process. These two interactions guide two complementary approaches of our terrain modeling tool. First, a coarse version of the terrain geometry is maintained in the CPU using a quadtree, as shown in figure 1 (middle), where regions closer to the viewer are subdivided more than far regions. This simple and lightweight data structure fits the CPU main role of data control, while allowing it to send adaptive quadrilateral (or *quad*) patches to the GPU (illustrated by the *camera-then-render* path in red in figure 2). Second, a fine version of the terrain is produced in the GPU using two programmable shaders within the tessellator stage. One of them is responsible to subdivide regularly each quad patch, i.e. the quad leaf node sent by the CPU, while the other reads the height values from a texture that changes interactively based on manipulation and drawing of strokes (illustrated by the *draw-then-control* path in blue in figure 2). It is important to note that the different data paths in our framework have an influence in each other, e.g. drawing changes the heightmap that updates the quad patches that changes rendering.

The first phase of our algorithm is to update the quadtree data structure using a level-of-detail mechanism. Our LOD-based approach considers the viewing projection of the bounding box of each quadtree node (see figure 3). This bounding box is constructed by reading the minimum and maximum height values that fall inside the node (illustrated by the *update-bounding-box* path in green in figure 2). We use a linear metric on the number of pixels of the bounding-box projection in order to make our LOD approach both view and height-values dependent. Using this metric, the quadtree node is recursively subdivided until a given criterion is met, effectively increasing the resolution of the current terrain and moving upward in the multiresolution pyramid.

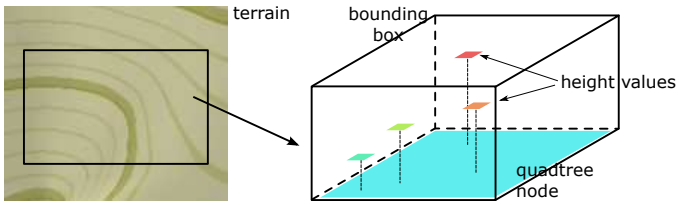


Fig. 3. Illustration of a quadtree node and its corresponding bounding box. The minimum and maximum height values are used to specify the limits inside the current node. The quadtree is refined depending on the projection shape of each node.

In our framework, we have used the rectangular shape of the bounding-box projection to determine the quadtree node subdivision. The recursive creation of child nodes, four at a time, is done in order to guarantee that each projection shape has about the same size (in pixels). Moreover, the screen rectangles are used to determine if a quadtree node is completely occluded and can be discarded, i.e. not sent for rendering as in the view-frustum culling technique. The result is an adaptive quadtree controlling the minimum resolution level imposed to the GPU (depending on each quad node subdivision), which changes interactively depending on camera properties. In the GPU, we use regular subdivision of patches (each patch corresponds to a quadtree node sent for rendering) but a more refined LOD technique could also be applied, allowing a second form of LOD in the top level of the multiresolution pyramid.

The second phase of our algorithm comprehends the following modules (depicted in figure 2): the *multigrid solver*; and the *tessellation shaders* module. The former provides the height values of the terrain to be used by the latter module.

The multigrid solver module, based on the work of Hnaidi *et al.* [15], computes the heightmap of the terrain through a sequence of increasing resolution textures, up to an arbitrary size, using a two-step algorithm. The first step solves a Poisson-based diffusion system to interpolate gradient constraints. The resulting gradients and height constraints are then used by the second step to compute the final heightmap employing a similar diffusion system. Both systems are solved using a multigrid strategy, detailed in figure 5, which first solves a coarse-resolution system to then increase the resolution and start solving again. While the dyadic upsampling is done once per level, the number of solver iterations is normally the same for each level and depends on the desired accuracy. The input of the multigrid solver module is given by stroke drawing and is explained in the next section. The output of this module is a multiresolution terrain stored as a mipmap texture pyramid (see figure 4). While the GPU is responsible for this module, the CPU is able to start and stop solver iterations within drawing and rendering events, keeping the GPU busy.

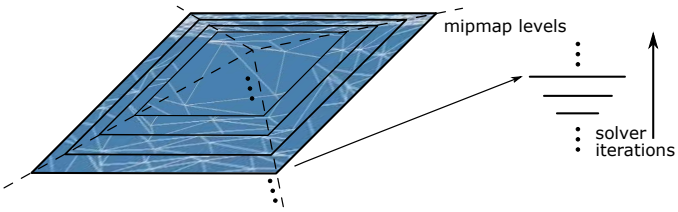


Fig. 4. The multiresolution texture scheme of our tool. The terrain is stored as a mipmap texture pyramid that changes over time, depending on the multigrid solver iterations done so far.

The tessellation shaders module uses the mipmap-pyramid texture generated by the solver module in a producer-consumer strategy, synchronized and controlled by the CPU. After each new resolution level is constructed by the solver, a shader uniform value is updated to the current maximum level. The CPU also controls the minimum level through the quadtree refined nodes sent as quad patches to the GPU.

The tessellation shaders module employs two stages of the graphics-hardware pipeline: the tessellation *control shader*, used to determine the subdivision level done by the tessellator fixed functionality; and the tessellation *evaluation shader*, used to specify properties of the vertices created by the tessellator. In our terrain modeling tool, the control shader uses a constant uniform value determined by the CPU to subdivide regularly each quad patch. Depending on the subdivision, the resolution of the patch can increase or remain the same. The evaluation shader reads the appropriate level of the mipmap-pyramid texture to place each generated vertex at the correct height position. The multiresolution texture is also used by the CPU, but only a small resolution of it (we use 64×64 from the 6th solver iteration) since the quadtree minimum leaf size is much bigger than the texel from the highest resolution texture.

In the tessellation shaders, we use a straightforward strategy for subdivision that does not consider different tessellation levels across patch boundaries. This may lead to T-junctions, i.e. edges connected forming a T. In our experiments, we generate a large number of triangles through the tessellation, making this possible T-junctions imperceptible. However a more elaborated strategy for low-level tessellation can use irregular subdivision ac-cross patches to avoid T-junctions. Our CPU-based LOD guarantees that visible quad patches have at most 2 neighbor patches (invisible patches can be arbitrarily big) and this fact can be used to construct a bit-mask for each patch, to be sent to the GPU, imposing a twice coarser subdivision for patches with bigger neighbors. This approach can be thought as a second form of LOD done by the GPU that is local-geometry dependent.

The integration of the first CPU phase and the second GPU phase is done after the quadtree is refined and before rendering. The leaf nodes of the current quadtree are transformed into quad patches, where each patch contains its corresponding mipmap level and texture position. From this patch, the GPU is able to further increase (but not decrease) the terrain resolution in the tessellation control shader, up to a specified resolution depending on the last solver iteration. This coupled solution defines the entire multiresolution pyramid, where the base level is determined by the CPU and the top level is computed and generated by the GPU.

B. Sketch-based and Interactive Edition

Interaction with terrain models in real-time requires primitives that are easy to create, manipulate and understand. Creating mountains or plains by stipulating each small piece of the terrain can be a cumbersome and slow work. Our terrain modeling system allows intuitive and natural interaction through simple sketches drawn over the current camera viewing plane. These sketches act by *pulling* and *pushing* the terrain surface. The sketches can be created, moved or deleted; effectively changing the terrain. Depending on the desired primitive to create or modify, the user can move the camera to choose the best view of the terrain. The camera zoom can also be used to interact with terrain primitives at different scales.

There are two main primitives supported by our sketch-based modeling tool: mountains and crevices primitives (drawing up or down the terrain level), and plateaux primitives (drawing in the same terrain level). These primitives can be seen in a terrain example shown in figure 6.

To interactively morph the terrain as each primitive is drawn or manipulated, the modeling strokes are converted into constraints used by the GPU multigrid solver in the second phase of our algorithm (explained in the previous section). The 3D strokes are projected and rasterized on top of the terrain creating an input 2D image used by the multigrid solver to specify its constraints. This process is further detailed in figure 5. The 3D curves before rasterization represent terrain primitives and are similar to curves in a vector-graphics tool.

The projection of 3D curves depends on the primitive been drawn. In the case of a plateaux primitive, the stroke is projected on the horizontal plane defined by the first point of the drawing that touches the terrain. In the case a mountain

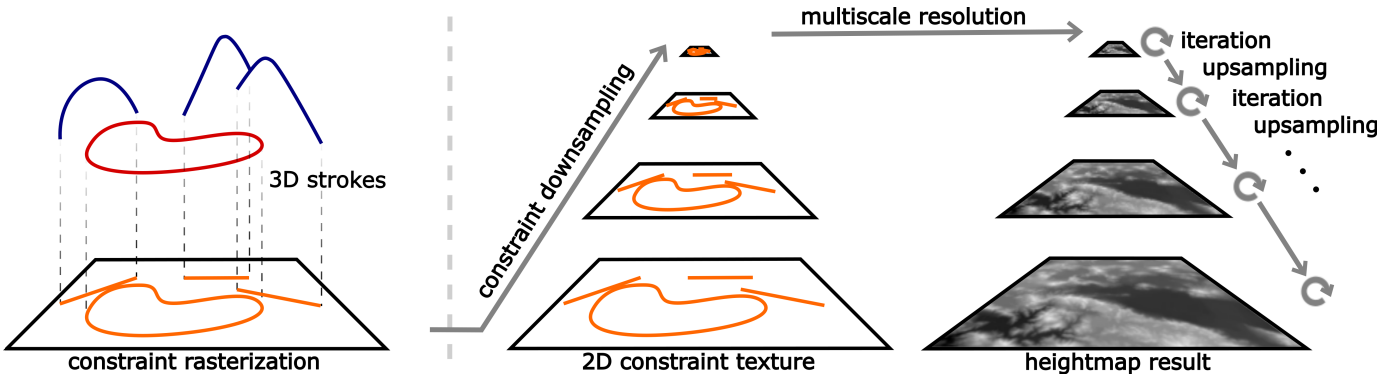


Fig. 5. The landscape primitives specified by 3D strokes are converted to constraints through rasterization and stored on a 2D constraint texture. This texture is downsampled to be used in a *solve-then-upsample* process, i.e. the multigrid solver, where it first solves for the small resolutions before extrapolating the solution for finer resolution systems. The final solution is the heightmap of the terrain in a multiresolution pyramid stored as a mipmap texture.

primitive, the first and last point of the drawing touching the terrain determine a vertical plane where the stroke is projected. However, in our real-time modeling scenario, the last point is unknown while the drawing editing occurs. One simple solution is to project the extremity of the stroke been drawn onto the terrain to obtain a dynamic last point. Although this solution successfully changes the terrain, the result is confusing to the user since both terrain and drawing are changing dynamically (one stroke point updates the terrain that updates where the next stroke point is projected). Instead of projecting the stroke extremity to the dynamic terrain, we use the depthmap of the scene when the user starts drawing as an approximation of the current terrain. By projecting onto this depthmap, with the appropriate transformations, the moving last point is now based on a fixed terrain, matching what the user expects to modify.

The manipulation of 3D curves can be done independently from each other and from new drawings. The first or last points used to compute the projection plane for primitives can be manipulated with drag-and-drop interaction changing the projection and thus the primitive. These points act as handles that can be moved morphing the terrain to a different configuration.

The creation and manipulation of different primitives are handled by the CPU. The solver in the GPU receives the final projection by reading from the 2D constraint texture (see figure 5). The different type of strokes and the number of terrain primitives affect little the performance of the solver.

IV. Results

We have tested our modeling system with different heightmap resolutions and number of solver iterations. Table I presents the terrain creation solver and tessellation-based rendering timings and GPU memory consumption for the tested terrain size and respective number of iterations. The timings are given using a 1024×768 pixel viewport and considering the camera constantly moving. All timings were performed in an Intel Core2Quad CPU and an nVidia 480 GTS GPU using OpenGL 4.1.

Terrain Size	Iterations	Creation (ms)	Tess. (ms)	GPU (MB)
512×512	45	23	4.8	16.9
$1K \times 1K$	49	28	5.2	57.3
$2K \times 2K$	53	35	5.9	141.8
$4K \times 4K$	56	44	6.7	716.7

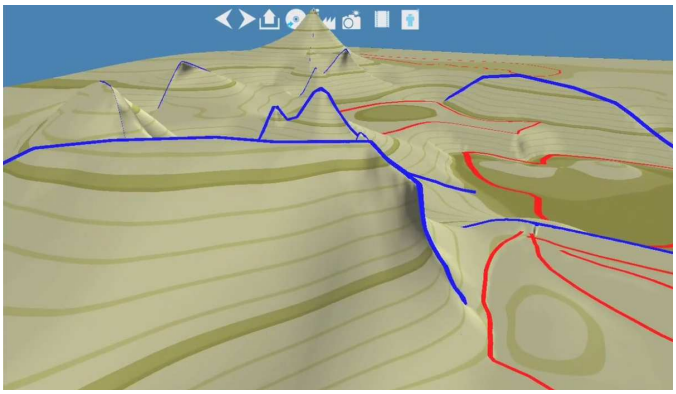
TABLE I
TERRAIN MODELING COMPUTATIONAL TIMINGS AND GPU MEMORY CONSUMPTION AT DIFFERENT RESOLUTIONS.

For these results we have drawn several strokes, changing the terrain shape and inducing the GPU multigrid solver, and considered the timing to create and tessellate the maximum resolution. The timings show that our system can compute the solver and tessellate high-resolution terrains in real-time (at about 20 frames per second) allowing for still higher resolutions when working with adaptive tessellation.

Analyzing the performance results further, the ability to control incrementally the creation of the terrain constitutes another important characteristic of our approach. The solver iterations can be used to specify the resolution levels and terrain features precision, which affects both timing and memory consumption. More iterations generate improved and higher resolution terrains, at the cost of more GPU time and memory. The memory usage in our system refers to the mipmap-pyramid texture stored in the GPU.

One terrain example modeled with our tool can be seen in figure 6. In this example appears different terrain primitives drawn: mountains, plateaux and crevices. In addition to terrain primitives, our tool allows several user interactions, including sketching and manipulation of previous strokes and camera movement.

Adding the GPU to the process of creating and tessellating the terrain (not only for rendering) makes our approach to run at 27.8 ms for a 512×512 grid. While Gain *et al.* [11] report 2.3 seconds on a similar CPU to generate the same terrain grid. That is, our approach generates terrains two order of magnitude faster than Gain *et al.*'s work employing one single unit.



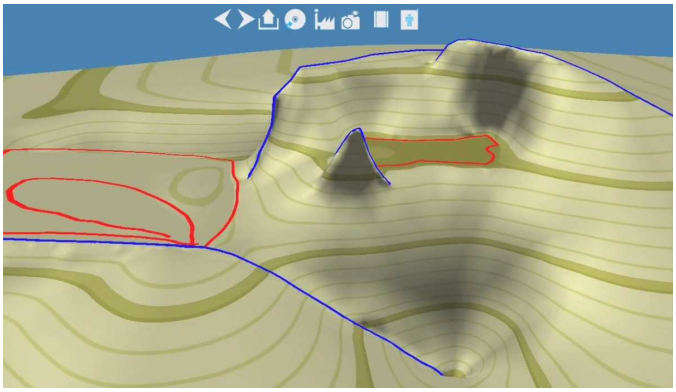
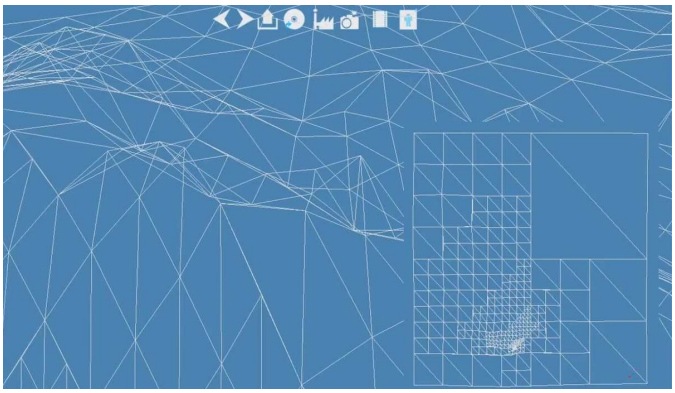


Fig. 6. Terrain example generated by our modeling system (north and south views). The blue strokes were drawn to create mountains and crevices, and the red strokes were drawn to create plateaux in the midst.



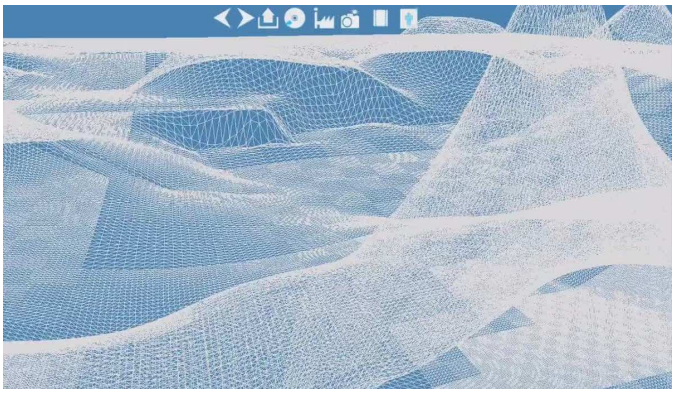


Fig. 7. Illustration of coarse (top) and fine (down) version of a terrain tessellated by the GPU. The coarser resolution corresponds to the quadtree (shown on the right), that is the minimum resolution level imposed by the CPU, and the finer resolution is the maximum resolution level.

Compared against Hnaidi *et al.*'s approach [15], our coupled solution introduces the usage of tessellation shaders and a CPU-based method to control and synchronize between drawing and rendering events. One direct advantage of this approach comes from the high-cost memory bandwidth between the CPU and the GPU; by using the CPU *only* for control and synchronization, the communication between the units becomes small (few kilobytes in our system) even though the GPU is handling large amounts of data (hundreds of megabytes for high-resolution terrains). Without the CPU–GPU coupled solution, the modeling framework changes from computation-bound (regarding FLOP count) to I/O-bound (regarding data transfers), compromising the real-time performance of the terrain modeling tool.

Another advantage is that while the multigrid GPU solver acts in a global scope, generating and improving the entire terrain, the CPU quadtree adaptation and the GPU tessellation (see an illustrative example in figure 7) act in a local scope, refining the terrain on demand for rendering. This strategy alleviates the burden on the CPU, in terms of editing and visualization, and on the GPU, in terms of control and synchronization, providing a comprehensive sketch-based terrain modeling tool. Without one unit or the other, the tool is less capable of dealing with high-resolution and dynamically changing terrains.

V. Discussions

The majority of procedural editing and feature extraction tools for terrain modeling are based on a top-view interaction [7], [10], [17]. Although interesting, this approach limits the artistic freedom of the tool. Our modeling system is based on a 3D camera with a first-person perspective and sketch-based drawing interaction over the moving view plane. This method allows the user to draw landscape silhouettes at any distance and with more freedom.

The drawing interaction is only convincing when the tool responds in real-time. Inspired by FiberMesh [4], our tool has controllability and real-time responsiveness. However, to deal with high-resolution terrains, we use a different type of solver to generate the model. While FiberMesh uses a direct solver with a high pre-computational cost, we use a fast multigrid solver in the GPU that is suitable for our goal.

Our GPU-based solver hints for a multiresolution hierarchy of terrains in the form of a texture pyramid. This pyramid is constructed top-down and can be used profitably for level-of-detail control on both the CPU and the GPU (as explained in section III). Aiming at only visualizing static and real terrains, the technique from the geometry clipmaps [21], [22] uses a better solution for LOD control, building a compressed multiresolution pyramid that is accessed bottom-up. However, we are interesting in a flexible modeling interface for dynamic

terrain generation and the geometry clipmaps' technique is not appropriate.

Between creating and tessellating the terrain in the GPU, the creation is more costly (see table I) since we use a simple regular subdivision approach in the tessellation control shader. In the case of a complex second form of LOD in the GPU, to allow even higher terrain resolutions, the bottleneck may become the terrain tessellation.

The two modules of the CPU, controlling the solver iterations and building the quadtree data structure, depend on the user input, drawing of strokes and moving the camera, and are intrinsic serial computations. Although the two modules can not be implemented by the GPU, they could be suppressed, making the modeling tool more simple to implement but less interactive.

VI. Conclusions

We have presented a real-time terrain modeling tool combining two strategies in the GPU with a lightweight CPU-based data structure. Our tool is capable of dynamically generate heightmaps with adaptive resolution, that is, our tessellation shaders are able to generate on-the-fly different parts of the terrain at different resolutions.

One interesting aspect of our CPU–GPU coupled computation solution is the usage of the CPU to control the GPU solver iterations and stop at a certain resolution and then resume computing when the GPU is idle. Another interesting feature of our method is the balance between terrain generation in the CPU and in the GPU – we can control this balance by simply changing the quadtree refinement. With these features, the user can draw strokes and see at the same time the terrain morphing to the drawing. Terrain primitives, such as mountains and clefts, are controlled seamlessly in our framework.

Although with interesting features, our modeling framework has a wide variety of future work directions. First, there is room for optimizations in the GPU solver: instead of performing the initial iterations in the GPU, the CPU is more adequate to compute these low-resolution mipmaps and use them in its quadtree data structure (avoiding transfering data back), while the GPU is not underutilized with small images. Second, the current usage of the mipmap-pyramid texture in the tessellation evaluation shader indicates the possibility of using anisotropic filtering with a gradient vector depending on the solver. Third, the tessellation shaders allow a second fine LOD analysis of the terrain that is interesting to explore. Finally, adding normal and fractal texture painting or simulation effects in our sketch-based modeling tool can increase the realism of the generated terrains. In short, we believe this work has the potential to become an effective terrain modeling tool in practical applications.

Acknowledgments

We would like to acknowledge the grant of the first author provided by the PPF Interactions Multimodales, and the grant of the second author provided by Brazilian agency CNPq (National Counsel of Technological and Scientific Development). We also would like to thank the anonymous reviewers for their valuable comments and suggestions.

References

[1] T. Igarashi, S. Matsuoka, and H. Tanaka, "Teddy: a Sketching Interface for 3D Freeform Design," in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH '99. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1999, pp. 409–416. [Online]. Available: http://dx.doi.org/10.1145/311535.311602

[2] R. D. Kalnins, L. Markosian, B. J. Meier, M. A. Kowalski, J. C. Lee, P. L. Davidson, M. Webb, J. F. Hughes, and A. Finkelstein, "WYSIWYG NPR: Drawing strokes directly on 3D models," *ACM Transactions on Graphics*, vol. 21, no. 3, pp. 755–762, 2002. [Online]. Available: http://dx.doi.org/10.1145/566654.566648

[3] R. Schmidt, B. Wyvill, M. C. Sousa, and J. A. Jorge, "ShapeShop: Sketch-based Solid Modeling with BlobTrees," in *ACM SIGGRAPH 2007 courses*, ser. SIGGRAPH '07. New York, NY, USA: ACM, 2007. [Online]. Available: http://doi.acm.org/10.1145/1281500.1281554

[4] A. Nealen, T. Igarashi, O. Sorkine, and M. Alexa, "FiberMesh: Designing Freeform Surfaces with 3D Curves," in *ACM SIGGRAPH 2007 papers*, ser. SIGGRAPH '07. New York, NY, USA: ACM, 2007. [Online]. Available: http://doi.acm.org/10.1145/1275808.1276429

[5] G. J. P. de Carpentier and R. Bidarra, "Interactive GPU-based Procedural Heightfield Brushes," in *Proceedings of the 4th International Conference on Foundations of Digital Games*, ser. FDG '09. New York, NY, USA: ACM, 2009, pp. 55–62. [Online]. Available: http://doi.acm.org/10.1145/1536513.1536532

[6] E. Yeguas, R. Muoz-Salinas, and R. Medina-Carnicer, "Example-based Procedural Modelling by Geometric Constraint Solving," *Multimedia Tools and Applications*, pp. 1–30, 2011, 10.1007/s11042-011-0795-0. [Online]. Available: http://dx.doi.org/10.1007/s11042-011-0795-0

[7] H. Zhou, J. Sun, G. Turk, and J. M. Rehg, "Terrain Synthesis from Digital Elevation Models," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, pp. 834–848, 2007. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/TVCG.2007.1027

[8] K. Perlin and L. Velho, "Live Paint: Painting with Procedural Multiscale Textures," in *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH '95. New York, NY, USA: ACM, 1995, pp. 153–160. [Online]. Available: http://doi.acm.org/10.1145/218380.218437

[9] E. Keller, *Introducing ZBrush*. Sybex, 2008.

[10] J. M. Cohen, J. F. Hughes, and R. C. Zeleznik, "Harold: a World Made of Drawings," in *Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, ser. NPAR '00. New York, NY, USA: ACM, 2000, pp. 83–90. [Online]. Available: http://doi.acm.org/10.1145/340916.340927

[11] J. Gain, P. Marais, and W. Straßer, "Terrain Sketching," in *Proceedings of the Symposium on Interactive 3D Graphics and Games*, ser. I3D '09. New York, NY, USA: ACM, 2009, pp. 31–38. [Online]. Available: http://doi.acm.org/10.1145/1507149.1507155

[12] R. Smelik, T. Tutenel, K. J. de Kraker, and R. Bidarra, "Integrating Procedural Generation and Manual Editing of Virtual Worlds," in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, ser. PCGames '10. New York, NY, USA: ACM, 2010, pp. 2:1–2:8. [Online]. Available: http://doi.acm.org/10.1145/1814256.1814258

[13] N. H. Anh, A. Sourin, and P. Aswani, "Physically based Hydraulic Erosion Simulation on Graphics Processing Unit," in *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*, ser. GRAPHITE '07. New York, NY, USA: ACM, 2007, pp. 257–264. [Online]. Available: http://doi.acm.org/10.1145/1321261.1321308

[14] A. Peytavie, E. Galin, J. Grosjean, and S. Merillou, "Arches: a Framework for Modeling Complex Terrains," *Computer Graphics Forum*, vol. 28, no. 2, pp. 457–467, 2009. [Online]. Available: http://dx.doi.org/10.1111/j.1467-8659.2009.01385.x

[15] H. Hnaidi, E. Guérin, S. Akkouche, A. Peytavie, and E. Galin, "Feature based Terrain Generation using Diffusion Equation," *Computer Graphics Forum*, vol. 29, no. 7, September 2010. [Online]. Available: http://liris.cnrs.fr/publis/?id=4974

[16] R. A. Finkel and J. L. Bentley, "Quad Trees a Data Structure for Retrieval on Composite Keys," *Acta Informatica*, vol. 4, pp. 1–9, 1974. [Online]. Available: http://dx.doi.org/10.1007/BF00288933

[17] L. Olsen, F. F. Samavati, M. C. Sousa, and J. A. Jorge, “Sketch-based Modeling: A Survey,” *Computers & Graphics*, vol. 33, no. 1, pp. 85 – 103, 2009. [Online]. Available: http://dx.doi.org/10.1016/j.cag.2008.09.013

[18] T. Igarashi, S. Matsuoka, and H. Tanaka, “Teddy: a Sketching Interface for 3D Freeform Design,” in *ACM SIGGRAPH 2007 courses*, ser. SIGGRAPH ’07. New York, NY, USA: ACM, 2007. [Online]. Available: http://doi.acm.org/10.1145/1281500.1281532

[19] T. Igarashi and J. F. Hughes, “A Suggestive Interface for 3D Drawing,” in *ACM SIGGRAPH 2007 courses*, ser. SIGGRAPH ’07. New York, NY, USA: ACM, 2007. [Online]. Available: http://doi.acm.org/10.1145/1281500.1281531

[20] N. Watanabe and T. Igarashi, “A Sketching Interface for Terrain Modeling,” in *ACM SIGGRAPH 2004 Posters*, ser. SIGGRAPH ’04. New York, NY, USA: ACM, 2004, pp. 73–. [Online]. Available: http://doi.acm.org/10.1145/1186415.1186500

[21] F. Losasso and H. Hoppe, “Geometry Clipmaps: Terrain Rendering Using Nested Regular Grids,” *ACM Trans. Graph.*, vol. 23, pp. 769–776, August 2004. [Online]. Available: http://doi.acm.org/10.1145/1015706.1015799

[22] A. Asirvatham and H. Hoppe, “Terrain Rendering using GPU-based Geometry Clipmaps,” in *GPU Gems 2*, M. Pharr and R. Fernando, Eds. Addison-Wesley, March 2005. [Online]. Available: http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter02.html

[23] O. Šťava, B. Beneš, M. Brisbin, and J. Křivánek, “Interactive Terrain Modeling using Hydraulic Erosion,” in *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA ’08. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2008, pp. 201–210. [Online]. Available: http://portal.acm.org/citation.cfm?id=1632592.1632622