Hybrid Rendering of Dynamic Heightfields using Ray-Casting and Mesh Rasterization

Lucas Ammann* Olivier Génevaux* Jean-Michel Dischler*
Université de Strasbourg
LSIIT UMR 7005 CNRS-UDS, France

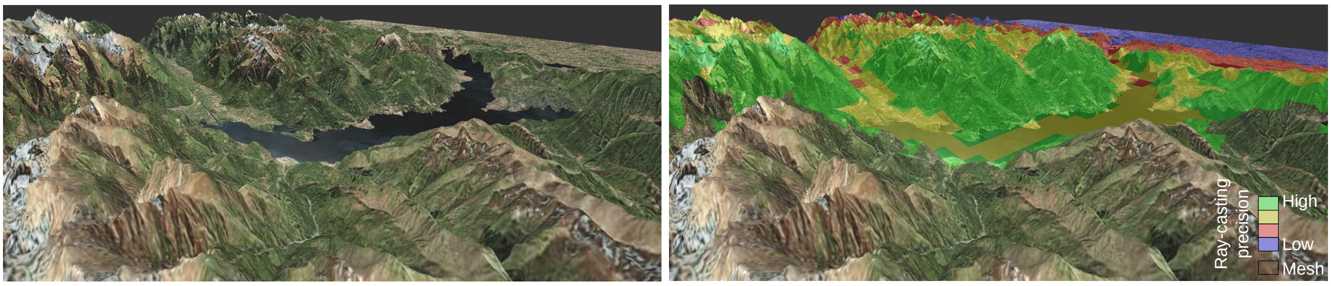


Figure 1: Rendering of the Como Lake (4096 × 4096 heightfield) using our method combining mesh rendering and ray-casting. This frame is rendered at 43 Hz on a viewport of 1900 × 700 px. The right part of the figure depicts the division of the terrain amongst the mesh and ray-casting with its four levels-of-precision.

Abstract

This paper presents a flexible hybrid method designed to render heightfield data, such as terrains, on GPU. It combines two traditional techniques, namely mesh-based rendering and per-pixel ray-casting. A heuristic is proposed to dynamically choose between these two techniques. To balance rendering performance against quality, an adaptive mechanism is introduced that depends on viewing conditions and heightfield characteristics. It manages the precision of the ray-casting rendering, while mesh rendering is reserved for the finest level of details. Our method is GPU accelerated and achieves real-time rendering performance with high accuracy. Moreover, contrary to most terrains rendering methods, our technique does not rely on time-consuming pre-processing steps to update complex data structures. As a consequence, it gracefully handles dynamic heightfields, making it useful for interactive terrain edition or real-time simulation processes.

Index Terms: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Raytracing; I.3.6 [Computer Graphics]: Methodology and Techniques—Graphics data structures and data types; I.3.8 [Computer Graphics]: Applications

1 Introduction

Heightfield rendering at interactive frame rates has always been an important topic in computer graphics, whose main application is terrain visualization, where the common representation is a regular sampling of the altitude. Finding out a good trade-off between speed, visual accuracy and flexibility remains the most challenging issue. Some applications, like video games require very fast frame rates, while others rather require high accuracy, even if frame rates are lower, like scientific visualization of 2D scalar fields or geomatic-related data visualization. Others require that terrains can be edited dynamically in real-time.

Most of the techniques can be broadly classified into two main families: mesh-based techniques, which make use of hierarchical representations, and ray-casting algorithms, which are now mainly executed on GPU. To be practical, mesh-based rendering methods do not render raw data and thus often involve pre-processing steps. As an example, an adaptive mesh reconstruction or the generation of a hierarchical multi-scale structure for applying level-of-details mechanisms are often used. Conversely, GPU ray-casting is able to directly process raw terrain data. The straightforward use of raw data, without heavy pre-processing, is interesting for interactive applications that involve non-static terrains. Interactive terrain edition for graphical content creation in the field of virtual landscape design is an important example of application. During real-time simulation processes, such as terrain erosion or ocean waves, the dynamic aspect is also crucial, as well as for scientific visualization where parameters, like transfer functions, must remain editable in real-time. Our motivation is to improve flexibility by offering a heightfield rendering method which strongly limits pre-processings and which makes no use of complex data structures, thus allowing for dynamic terrain manipulations at interactive frame-rates.

Unfortunately, straightforward ray-casting of raw data, while avoiding any pre-processing, remains too slow to be practical. Speedup techniques further used introduce approximations and lower fidelity: Dick et al. [5] have shown that a mesh representation still performs better than a ray-casting technique for moderate data resolution. Yet, mesh based methods tend to smooth reliefs because they handle large data sets in a hierarchical way such as for geometry clipmaps [14].

In this paper, we address such smoothing by combining a mesh representation and a GPU based ray-casting rendering. The mesh representation uses the full resolution of the data close to the viewer, while ray-casting rendering is performed further away. Ray-casting precision is governed by the view distance and relief characteristics so as too balance quality versus speed, as shown on Figure 1. The use of one or the other rendering technique is driven by a heuristic based on an analysis of the error induced by ray-casting. By fully exploiting the GPU and by evaluating the ray-casting error, our method improves the balance between real-time rendering per-

*email: {ammann,genevaux,dischler}@unistra.fr

formance and high visual accuracy over existing techniques, still allowing the data to be edited or dynamically modified.

The key points of our approach can be summarized as follows:

- a hybrid rendering method combining per-pixel ray-casting and mesh rendering, both executed on GPU to provide interactive frame rates.
- a simple heuristic, based on an error evaluation of the ray-casting rendering, to balance both methods and to maintain high visual accuracy.
- a method using only simple data structures, without pre-processing, thus able to handle dynamic heightfields.

The remainder of this paper is organized as follows: in Section 2, we briefly review existing methods for heightfield rendering. In Section 3, we present our rendering method. In Section 4, an analysis of a ray-casting algorithm is proposed and our adaptive mechanism is described in Section 4.2 with a detailed analysis. Finally, some results are shown in Section 5, before concluding in Section 6.

2 Related works

In the following three subsections, we respectively present mesh-based rendering techniques, ray-casting-based rendering and hybrid methods.

2.1 Heightfield rendering with meshes

Historically, the first family of terrain rendering methods is based on the use of triangular meshes. To reduce the amount of triangles without sacrificing the terrain geometry, these methods use different strategies, such as adaptive level-of-details meshing algorithms [7, 8, 13], advanced data loading and caching mechanisms to fully exploit GPU [12, 4, 10], or dedicated compression algorithms [14, 6]. We refer the reader to the overview of Pajarola and Gobbetti [18] for more details concerning these rendering techniques.

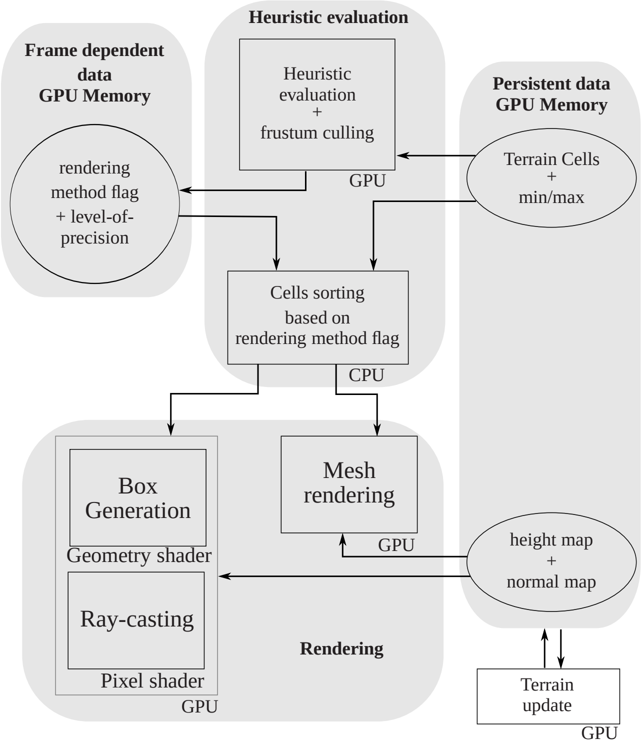
A good compromise between rendering quality and performance is offered by the geometry clipmaps [14]. This simple approach consists in selecting a level-of-detail in world space based on viewer distance using a set of nested rectangular regions. It also proposes a compression method associated to a regular mesh rendering.

2.2 Heightfield ray-casting

The second family of heightfield rendering techniques is based on ray-casting. These methods were initially developed on CPU [16], but with modern hardware, it becomes possible to execute them on GPU.

Mantler and Jeschke [15] propose a rendering method using a per-pixel ray-casting executed on GPU. This method was developed to integrate some other objects, like trees. There is no adaptive mechanism but an empty space skipping algorithm accelerates the ray intersection lookup with the heightfield. Jeong and Han [9] also propose a per-pixel rendering of terrains using ray-casting, where proxy geometry, to initiate ray-casting, is a simplified mesh of the terrain. This process reduces the ray intersection lookup but introduces a pre-processing step to build this proxy. If the heightfield is dynamic, this step needs to be recomputed at each frame.

In addition to previously described techniques, while not specifically targeting terrain rendering, relief mapping algorithms [20, 22, 21] are more focused on adding details to an object surface using a heightfield as input data. This class of algorithms, executed on GPU, computes an approximate intersection between a ray and the heightfield. To perform a more accurate and faster intersection lookup, methods such as [2, 19] rely on heavy pre-processing. But the major drawback is the important computation time, which prevents the data to be dynamic. Tevs et al. [23] propose a technique to accelerate the ray lookup intersection. This algorithm is used and improved by Dick et al. [5] to render large scale terrain split into tiles. In this paper, the authors also demonstrate the viability of ray-casting algorithms to render terrains but conclude that rasterization techniques still remain more efficient for small and medium scale terrains.



Heuristic evaluation
Frame dependent data GPU Memory
Heuristic evaluation + frustum culling
GPU
Persistent data GPU Memory
Terrain Cells + min/max
rendering method flag + level-of-precision
Cells sorting based on rendering method flag
CPU
Box Generation
Geometry shader
Ray-casting
Pixel shader
GPU
Mesh rendering
GPU
height map + normal map
Rendering
Terrain update
GPU

Figure 2: Diagram of our rendering pipeline. Arrows describe data flow between each step of the whole rendering process and the GPU memory.

2.3 Hybrid techniques

One way to improve the rendering performance is to combine different rendering algorithms, choosing the best performing one for a given situation. In this spirit, Becker et al. [3] were the first to combine different kinds of relief rendering techniques: BRDF, bump mapping and displacement mapping. Smooth transitions are then introduced to avoid popping artifacts when switching between these three representations. Larue et al. [11] also propose a hybrid algorithm to render the small canvas relief of digitized art paintings. They combine relief mapping with bump mapping and introduce an adaptive mechanism which automatically chooses the better rendering technique according to the current viewing conditions and painting relief. We apply such a kind of approach to terrain rendering, and we combine mesh rendering and ray-casting depending on viewing conditions and landscape data too.

3 Hybrid rendering method principles

The principle of our hybrid rendering technique is summarized in Figure 2. The terrain is divided into a regular set of identical square cells. For each cell, the adaptive mechanism, guided by our heuristic that will be described in the next section, chooses the best rendering algorithm to use: mesh or ray-casting. The adaptive mechanism also adjusts the ray-casting precision amongst predefined levels. For each level a given shader is used. To further increases performance, frustum culling is performed on cells during this process. The work-flow is as follows: first, the heuristic is evaluated for each cell on the GPU (see top of figure). Then, a read-back from

GPU memory to the main CPU memory is needed to group together the cells that share a common shader. Finally, the data are displayed precision levels by precision levels (see bottom of figure).

When the content of a cell is rendered using a mesh, vertex displacement mapping is performed, using one quad of mesh stored on GPU memory. This quad is designed to match the number of elevation points enclosed in any cell. This quad is overlaid on cells and the vertex displacement mapping is steadily applied on the vertices of this quad.

When ray-casting is selected, we use the algorithm described by Policarpo et al. [20]. All the front faces of the box bounding the content of the cell are the rendering proxy: for each pixel drawn, a ray is cast and the intersection with the heightfield is computed using a linear search followed by a binary search. Both steps are performed using a fixed number of iterations. Note that a ray can have no intersection with the heightfield during the traversal of the box. Proxy boxes are adjusted to the height extend inside the cell, to keep their footprint as tight as possible. To ensure proper continuity across cells boundaries, maximal and minimal height values are computed using one pixel overlap with neighboring cells.

It could have been possible to apply a per-pixel precision modulation within the ray-casting shader, but we experienced that adaptation at cell level performs better due to performance issues related to dynamic branching on GPU at pixel-level. Therefore, we use a pre-defined fixed number of ray-casting shaders each one corresponding to a different level-of-precision.

Since our goal is to handle dynamic heightfields, no speedup techniques involving many pre-computed data can be used, such as safety zones used for empty space skipping [2, 19].

As can be seen in Figure 2 (right part), our combined rendering method requires only minimal extraneous data on top of the heightfield data: cell subdivision information (e.g. the cell's origin) and minimal/maximal height values of its bounding box. When a terrain is evolving dynamically, both minimal and maximal height values as well as terrain normals evolve. Hence, these values need to be updated accordingly. Fortunately, computing minimal and maximal heights for each cell turns out to be a very light process, which can be done interactively on GPU in parallel to the normals computations. In addition, since the terrain is divided into an independent set of cells, only the altered cells can be updated in case of a local deformation or edition, keeping the operation costless.

Since our method targets direct use of heightfield data, it does not make use of level-of-details techniques. Indeed, using a straightforward approach (direct geometrical mipmapping for example) does not provide a sufficient control on the geometrical simplification as with geometry clipmaps. As a consequence, data over smoothing would be introduced, as shown in section 5. Using a more complex approach for the level-of-details mechanism, based on a fine analysis of the data, may reduce the geometrical over smoothing, but the required complex data structures would be difficult to handle on the GPU, add memory overheads and require pre-processing steps. Interactive performances may be degraded up to an unacceptable level too. As a consequence, the rendering of dynamic data sets would be severely impaired. Nevertheless, as explained in section 4.3, if direct use of heightfield data is not required, implementing such mechanism would still be possible. However, even if such multi-resolution technique would be included, no major performance increase would be achieved since we use an image based rendering method for the faraway areas. Indeed, rendering complexity depends on the number of rendered pixels, and is not directly linked to data size (except for hardware constraints like memory access latency, for example). Yet, some aliasing artifacts resulting from the mismatched data/screen resolution would certainly be alleviated due to surface smoothing.

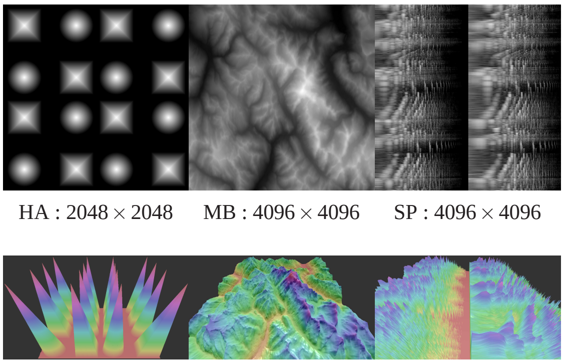


Figure 3: Heightfields used for error evaluation: high amplitude relief (HA), Mont Blanc terrain (MB) and spectrogram (SP).

4 Adaptive mechanism

In this section, we describe our heuristic to automatically choose between the two rendering techniques. Since ray-casting does not guarantee an exact intersection between rays and the heightfield surface, it might introduce some visual artifacts. Hence, our heuristic must also take into account such artifacts to make them invisible or to minimize them. In the next subsection, we first study and evaluate the rendering errors introduced by ray-casting. Based on this study, we then propose an efficient heuristic.

4.1 Ray-casting Error Analysis

Our motivation is to provide a measure that allows us to evaluate the error induced by ray-casting compared to an accurate, full-resolution, mesh rendering with respect to various parameters. The following parameters are known to be involved in the ray-casting rendering quality: viewing distance and angle, terrain characteristics and heightfield intersection lookup precision.

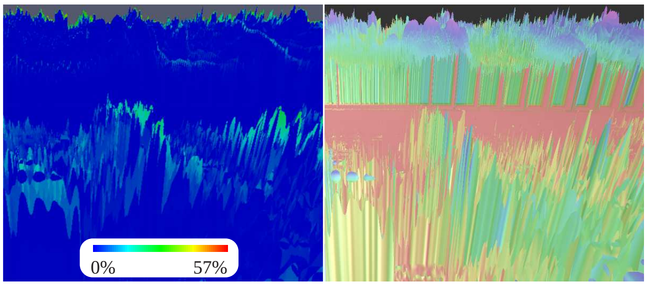
As error measure, we propose to use the distance, along the viewing ray, between the accurate reference mesh surface and the surface rendered using the ray-casting algorithm. Such an error measure is well suited to evaluate ray-casting errors and provides an intuitive interpretation of the rendering accuracy. Indeed, a value of zero means no error at all, compared to the actual mesh rendered terrain, while the value can become infinite if parts are completely missed. We conducted many tests and measurements on two extreme ray-casting cases, with varying view angles (from grazing to orthogonal) and varying view distances: a very low quality case, using only 5 linear steps and 2 binary refinement steps and a high precision case, using 64 linear steps and 25 binary refinement steps.

We also evaluated the error on different data sets: a standard terrain data set (the Mont Blanc, denoted MB), a heightfield from a spectrogram (SP) and a synthetic data set (HA) with high amplitude reliefs (see Figure 3). The spectrogram data set is chosen for its high frequency details to magnify the rendering technique's ability to preserve small-scale terrain details. Moreover it features a strong anisotropy with many parallel narrow valleys and ridges. The synthetic high amplitude data set is used to characterize the silhouette preservation of the rendering method.

To evaluate the overall precision of the rendering, a peak signal-to-noise ratio (PSNR) error is computed, for each frame, as follows:

$$E = 10 \cdot \log_{10}\left(\frac{R^2}{MSE}\right)$$

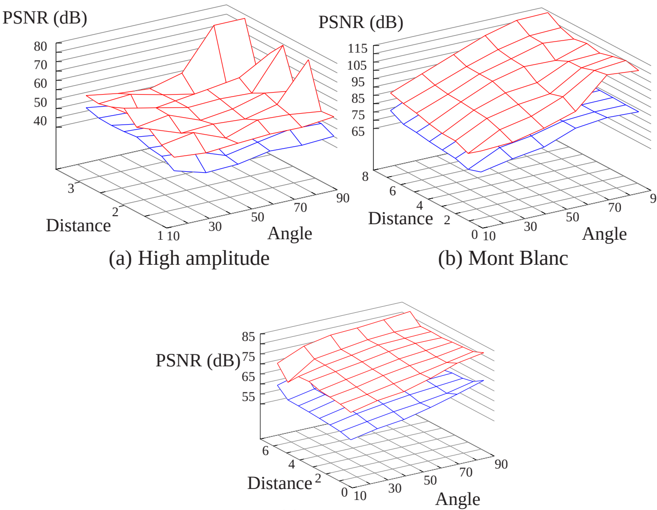
where the mean squared measured error is derived from the errors e_i of all rendered pixels in frame F (with $\#F$, the number of rendered



Error distribution

Rendering result

Figure 4: Geometric error of low precision ray-casting versus reference mesh, on spectrogram data set (SP). Error is relative to the terrain length. PSNR error for this capture is 51.87 dB. 69.5% of the pixels have an error less than 1% of the terrain length; 80% less than 5% terrain length and 89.5% less than 10% terrain length.



(a) High amplitude

(b) Mont Blanc

(c) Spectrogram

Figure 5: PSNR measured for two ray-casting precisions (high precision in red, low precision in blue) on three data sets with varying view-angles and distances to the viewer.

pixels):

$$MSE = \frac{1}{\#F} \sum_{i \in F} e_i^2$$

and correlated to the terrain bounding box typical length R.

Figure 4 shows an example of the spatial error distribution of low precision ray-casting for a grazing view angle. As visible on this figure, it results in large errors, since some intersections are missed at crest lines. Consequently, relief silhouettes are altered. These artifacts are reduced to become negligible when ray-casting precision increases.

As we can see on Figure 5, the major influence on ray-casting quality is the viewing angle. If the terrain is smooth, the error is almost linearly dependent on this angle. Conversely, the distance does not introduce rendering artifacts: the rendering quality is quite constant depending on the distance. The relief influence becomes important when the relief amplitude is high, as it is the case with our high amplitude data set: errors are always present even with a near orthogonal view angle.

From our experiments, we see that the size of the terrain does not have a direct impact concerning the rendering performance, which is presented in Table 1. This is related to the fact that ray-casting is mainly dependent on the number of pixels that are rendered. Rendering performance is also directly linked to the precision of the ray/heightfield intersection lookup steps. As shown on Figure 4, if the precision is not sufficient, rendering artifacts become strongly visible in the foreground. These artifacts are also more pronounced when the viewing position is moving, since it results in flickering effects. As a consequence, low precision ray-casting must be used carefully, and a good balance must be determined between quality and performance.

Data set	High amplitude	Mont Blanc	Spectrogram
(Data size)	(2048^2)	(4096^2)	(4096^2)
Low precision RC	430	360	238
High precision RC	76	47	36
Full mesh rendering	26	6.9	7

Table 1: Rendering speed (in Hz) for a grazing view angle on a viewport of 1024 × 768.

4.2 Heuristic

Based on our observations, we now introduce a heuristic to choose between the two rendering techniques in a way that enhances the rendering quality without sacrificing the rendering performance.

As we showed previously, ray-casting can achieve high rendering quality or high rendering performance depending on the intersection lookup precision. Moreover mesh rendering with a number of triangles matching the full resolution of the data offers the most accurate rendering. At close range, mesh rendering is also the most efficient rendering method since a single triangle projects on numerous pixels, actual performance depending on graphics hardware. Taking this fact into account, we propose to use a threshold based on terrain screen coverage to choose between mesh rendering and ray-casting. Thus, we define the threshold T according to hardware capabilities: when mesh rendering becomes more efficient than ray-casting, the heuristic should select mesh rendering. Intuitively, this threshold should be compared to the triangle's area on screen.

For each cell, the associated box is projected onto the screen. Then, its footprint area is approximated by computing, in screen space, the area A of the bounding rectangle of the 8 corners of the box. This value summarizes the distance from the camera and the maximum relief variation (represented with the minimum and maximum height values) inside the cell. In fact, since we are interested in the triangles footprint on screen, A is divided by the number of terrain samples S contained within the cell (thus $S = n^2$ where n is equal to the number of samples on the cell width). If A/S is greater than the threshold parameter T, then mesh rendering should be used:

$$\begin{cases} A/S > T \Rightarrow \text{mesh rendering} \\ A/S \leq T \Rightarrow \text{ray-casting rendering} \end{cases}$$

If ray-casting is chosen for the current cell, then its precision must be further determined.

To improve rendering performance, we want to use the least possible ray-marching steps. But, as described in Section 4.1, ray-casting introduces some artifacts at grazing view angles, which we want to avoid, as much as possible. As previously shown, height amplitude of a terrain cell is also an important parameter: if this cell represents an important part of the view, ray-casting precision should be increased. In addition, the ray-casting absolute precision can be decreased according to the distance without affecting the rendering quality.

Thus, we incorporate the viewing angle, the distance between camera and cell, and the height amplitude of the cell in our heuristic. For a given box, we propose the following formula to compute the level-of-precision:

$$L = \frac{min(d_b, d_{max})}{d_{max}} - (h \cdot \cos\theta)$$

where h is the height of the box (elevations of the heightfield are supposed to be contained between 0 and 1), θ is the angle between the viewing ray and the vertical at the center of the box and d_b is the distance between the center of the box and the camera. The value d_{max} is computed with $d_{max} = k \cdot d_{BB}$, where d_{BB} represents the maximal possible value for d_b, which corresponds to the diagonal length of the terrain bounding box in world space. The heuristic is decomposed into two terms. The first one controls the variations due to the viewing distance. The second one controls the variations related to the viewing aspect (i.e. the viewing angle compared to the heightfield base plane). The heuristic depends on a parameter k which includes a terrain size normalization factor and which finally modulates levels-of-precisions distribution according to the distance between the cell and the camera.

Finally, L values are evenly distributed amongst the ray-casting levels-of-precisions, $L = 0$ representing the highest level-of-precision.

4.3 Implementation notes

To test our method, we implemented it using OpenGL on a standard PC. All measures were performed using a NVIDIA GeForce GTX 280 graphics card with 1 GB of graphical memory.

For each cell, only heightfield data (elevation and normal) and the elevation bounds are required. These data are stored in two textures whose layout match the terrain one. This layout was chosen to keep our implementation simple but this prevents us from handling extremely large data sets, as our implementation only allows heightfields which entirely fit into GPU memory to be rendered. Nevertheless, since the data are accessed cell by cell (for visible cells) during rendering, it would be possible to further add a streaming mechanism to dynamically upload required data into the GPU memory on demand.

As previously described in section 3, in order to allow dynamic heightfields rendering, no pre-processing is performed. Consequently, no optimization technique for ray-casting, like empty space skipping mechanisms [2, 19, 23], is included. Yet, it is conceivable to integrate such kind of techniques. However, memory footprint would be increased and handling dynamic heightfields would be restricted, or even become impossible.

4.4 Heuristic parameters analysis

As the heuristic defined previously is based on several parameters, the influence of these parameters is now analyzed.

The cell size has an influence on the rendering performance. We obtain the highest frame rates for box sizes varying between 50^2 and 128^2 samples, as shown on Figure 6. In fact, when box size becomes too small, rendering performance decreases: too many cells must be rendered using ray-casting which involves increasing overdraw: some pixels are drawn multiple times in neighboring boxes. When box size increases, the meshed surface becomes more important and slower to render (when many cells are rendered using this algorithm). Moreover, a higher overdraw also occurs during the ray-casting pass.

Table 2 shows the influence of the number of different ray-casting levels-of-precision. To measure these rendering timings, we use the Puget Sound data set (4096 × 4096 px) with a box size of 75 × 75 pixels. The scene is rendered using a varying number of levels-of-precision: the number of steps ranges from 64 to 6 for linear search, and from 50 to 5 for binary search. This parameter only affects ray-casting, so the mesh rendering timing is identical for all configurations. Moreover, the time elapsed to evaluate the level-of-precision heuristic is constant since the computation is the same. On the contrary, the ray-casting rendering pass timing increases with the number of levels-of-precision. This overhead cannot clearly be traced back to a specific part of the rendering, but might be the consequence of increasing shader context switching. Experimental results show that rendering quality does not significantly increase along with the number of levels-of-precision since the lowest and highest precision levels both stay the same. These results also show that four levels-of-precisions provide the best balance between rendering speed and quality.

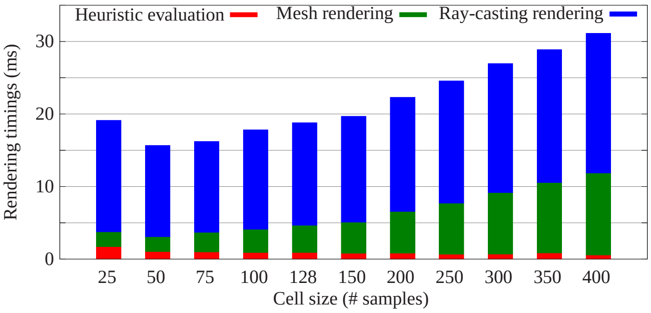


Figure 6: Influence of cell size on rendering timing for the Mont-Blanc data set (MB).

# LOP	Heuristic evaluation	Mesh rendering	Ray-casting	Total rendering
2	0.38	2.38	10.80	13.61
4	0.42	2.33	12.13	14.95
6	0.38	2.33	13.04	15.80
8	0.38	2.33	13.83	16.60
10	0.38	2.35	14.66	17.45

Table 2: Influence of the number of levels-of-precisions (LOP) on the rendering performances. Timings are in ms.

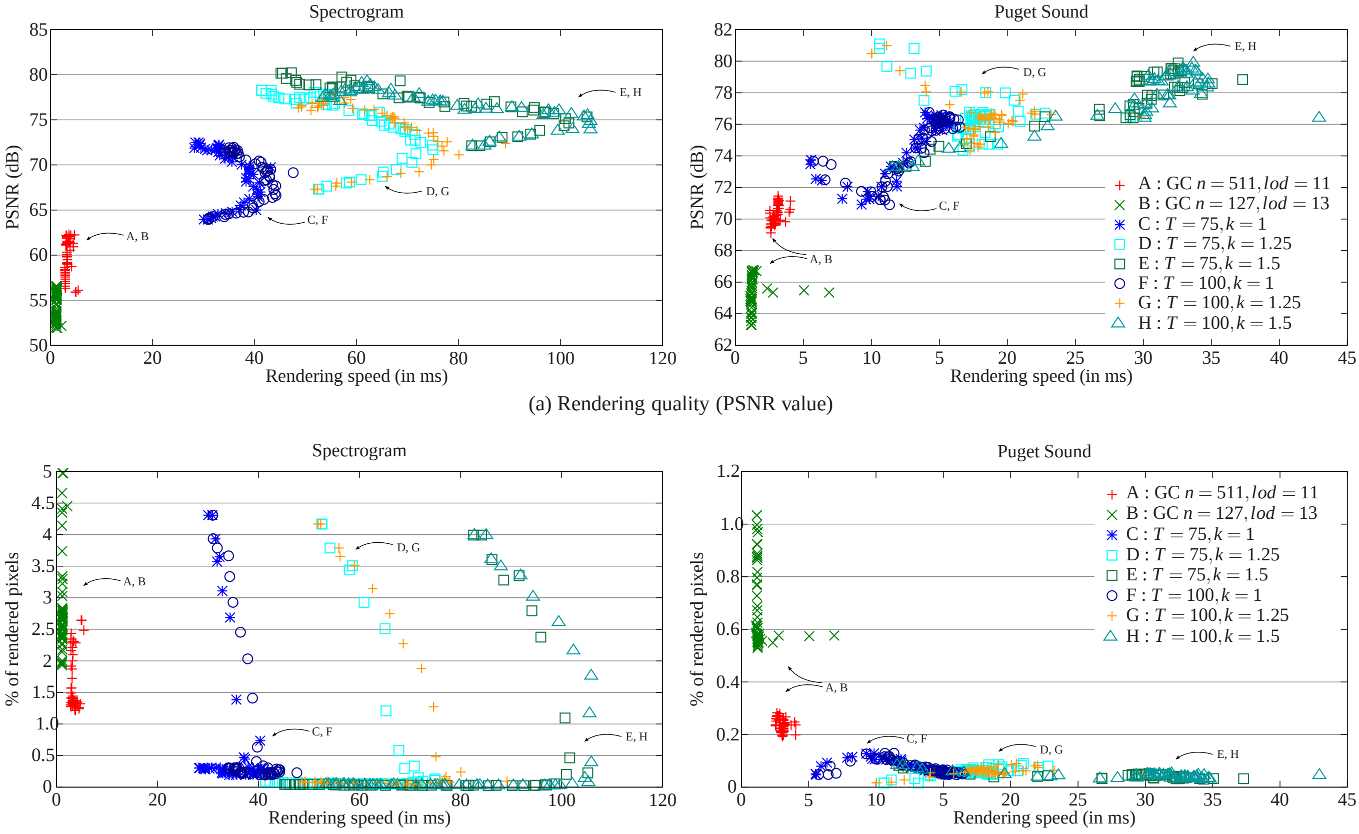
Our heuristic uses different parameters to allow an adaptation to the data and allows a user to set the trade-off between rendering precision and speed. The parameter T, used to define whether or not a mesh is used, mainly selects the foreground area of the heightfield, and allows one to improve the rendering quality for these area: when T increases, the mesh coverage of the heightfield becomes more important. The parameter k, used to adapt the heuristic to the relief characteristics, acts on the precision distribution of the ray-casting. If k is low, more parts of the heightfield are rendered using a low level-of-precision, so rendering error increases.

Figure 7 shows the influence of these parameters on the rendering speed and quality. Quality is expressed with PSNR error and terrain silhouette rendering precision (wrong pixels). This figure shows values measured on several frames of a fly-through over two different data sets. As we can see, the rendering quality is strongly linked to the parameter k which adapts the ray-casting distribution. Indeed, when using a low k parameter, more cells are rendered using a low-precision ray-casting, so the rendering quality is decreased but the rendering speed is increased. With regard to the parameter T, its impact on rendering quality and speed is less important than k. The rendering speed is slightly lower when this parameter increases but the quality is not significantly enhanced.

However, we can notice, on very high frequencies data sets like the spectrogram data set, that ray-casting misses more pixels, especially for grazing view angles. This is confirmed on Figure 7 with a higher proportion of wrong pixels on some frames.

5 Results and comparison with geometry clipmaps

Some rendering results and a thorough comparison with geometry clipmaps [14] are now presented. The ability of our method to han-



(a) Rendering quality (PSNR value)

(b) Wrong pixels

Figure 7: Comparison between geometry clipmaps (GC) and our method for varying parameters. (a) PSNR error value (y axis) is expressed depending on the rendering speed (x axis). Each point of the plots represents PSNR and speed values for a single frame of a fly-through over the spectrogram data set (left) and over the Puget Sound (right). (b) proportion of wrong pixels expressed depending on the rendering speed for the same frames.

dle dynamic heightfields is also presented.

5.1 Rendering quality and performances

Figure 1 shows a rendering result for the Como Lake data set of resolution 4096 × 4096 px. The colors highlight the distribution of the levels-of-precision of the ray-casting algorithm. It clearly shows that our heuristic is guided by relief features to define the precision of the ray-casting. We used $T = 70$ and $k = 1.30$ for a cell size of 128^2 samples and four levels-of-precision (linear/binary steps): 55/25, 32/15, 16/10, 8/5. Using this configuration and a viewport of 1024 × 768 px, a frame rate between 40 and 150 Hz is obtained, depending on the viewing position. When the same data are rendered using a full resolution mesh, with the same viewing conditions, the frame rate varies between 5 Hz and 9 Hz.

Compared to a reference mesh rendering, on Figures 8 and 9, our rendering method well preserves small reliefs and, consequently, does not smooth reliefs with high frequencies details.

A major drawback using hybrid techniques is popping artifacts which occur at the transitions between different rendering techniques and levels-of-precision. However, with our method, given sufficient precision on ray-casting no popping artifacts are visible. Our method does not introduce cracks between cells rendered using different levels-of-precisions. All cells rendered as a mesh use the same full resolution terrain samples thus, no T-junction is ever created at the boundary between two meshed cells. At mesh-ray traced boundaries, no crack is produced either because ray tracing operates on the full resolution data too: entry point of the ray in the proxy box is always coherent regarding to the mesh rendered in the neighboring cells because both use the same sampling of the terrain and the same linear interpolation between samples. It is an advantage compared to level-of-details mechanisms which combine meshes with different resolutions: they frequently need to deal with cracks at the boundary of terrain areas rendered using different resolutions and rely to complex re-meshing systems or adaptive tessellations.

Since no geometrical level-of-details mechanism is used, our method does not intrinsically mitigate aliasing artifacts. However, experimental results show that artifacts are limited to the less accurate levels-of-precisions. In fact, cells rendered with mesh or with a high ray-casting level-of-precisions do not suffer from aliasing. As the full mesh is used for the closest part of the terrain, suitable parameters can keep the triangles big enough to avoid these artifacts. Still, in the background, minor aliasing cannot be easily alleviated because no antialiasing mechanism is used.

5.2 Comparison with clipmaps

We compare our method to the GPU handled geometry clipmaps [1], with the following parameters: $n = 511$, 11 levels-of-details, and an alpha transition of $n/10$. Since the clipmaps only use a distance based level-of-details scheme, small relief features are over smoothed, as we can see on Figure 9. The error is mainly located on the crest lines. This is also confirmed on the top of Figure 7 where we can see better PSNR error values for our rendering technique. To enhance the over smoothing of the geometry clipmaps and its effect on the PSNR measure, we also use more degraded settings

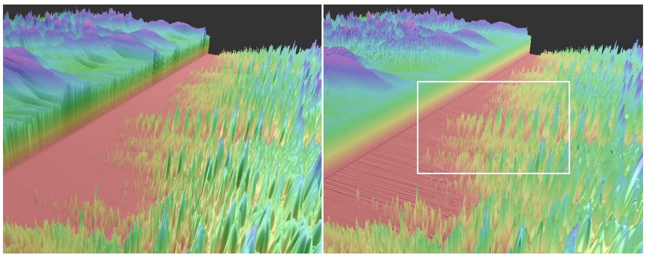
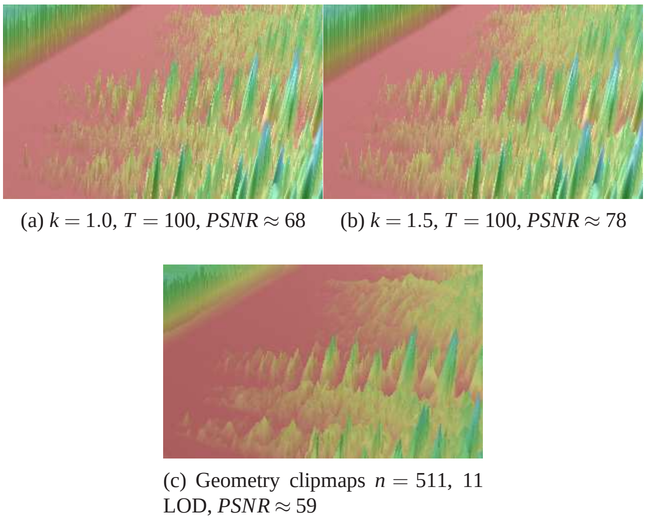


Figure 8: Rendering comparison between our method (left) and a reference mesh rendering (right).



(a) $k = 1.0$, $T = 100$, $PSNR \approx 68$

(b) $k = 1.5$, $T = 100$, $PSNR \approx 78$

(c) Geometry clipmaps $n = 511$, 11 LOD, $PSNR \approx 59$

Figure 9: Rendering quality comparison between our method (a) and (b) for varying k parameter values and the geometry clipmaps (c). Excerpt from Figure 8.

for the geometry clipmaps ($n = 127$ and 13 levels-of-details).

Since our heuristic takes into account relief features, it better preserves them. For data with high relief amplitudes or high relief frequencies, silhouette errors are also mostly avoided with our technique, while they remain more present with geometry clipmaps, as illustrated on Figure 9. In fact, compared to a reference mesh rendering (Figure 8), some pixels are not rendered and some others are rendered at a wrong place, especially on crest lines. These visual observations are corroborated by the numerical PSNR error, as shown on the top of Figure 7. The bottom of the Figure 7 also quantifies this phenomena. As we can see on this figure, our method, depending on the heuristic parameters, minimizes these rendering errors.

Several experiments on various terrain data sets have shown that our method provides better results on data sets with high frequencies details: geometry is well preserved without any smoothing of the small details, maintaining real time rendering frame rates.

5.3 Dynamic heightfields

One key point of our method is its straightforward handling of dynamic heightfields. Our method does not introduce any dependencies between cells for the rendering. As a consequence, local deformations of the data remain local to the data and do not propagate to the whole data. Global modifications are also possible and could be performed in real time, depending on the data set size and the complexity of the deformation process. No height amplitude constraints are applied to the deformations since the elevations remain between 0 and 1.

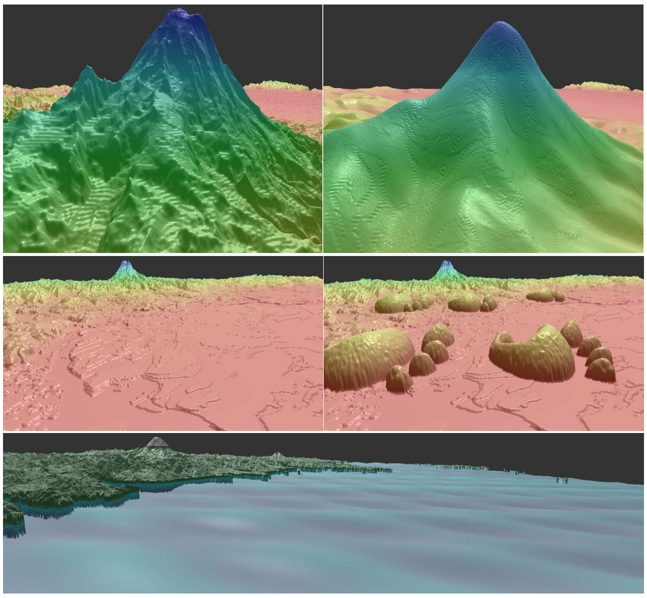


Figure 10: On top and middle, Puget Sound data set (4096 × 4096 px) before (left) and after (right) interactive erosion (≈ 35 - 45 Hz) and editing. On bottom, Puget Sound data set with wave simulation process (≈ 35 - 45 Hz).

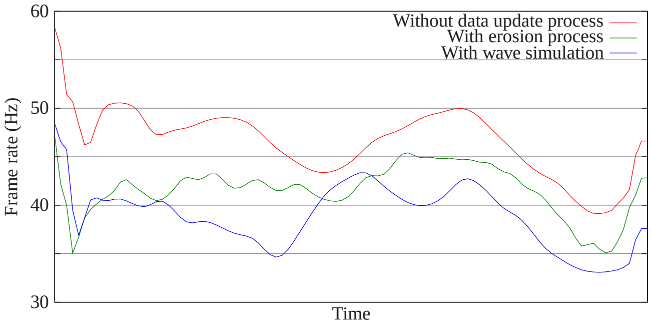


Figure 11: Speed rendering comparison between our method, our method with an erosion process and our method during a wave simulation on the Puget Sound terrain. For both erosion and wave simulations, height data are updated every 100 ms.

As examples, we have implemented a real-time erosion mechanism [17], an interactive terrain stamping and a simple ocean wave simulation. For the erosion mechanism and wave simulation, heightfields are entirely updated. For the terrain stamping, only the modified area of the heightfield is updated. Results are shown on Figure 10.

As shown on Figure 11, we are able to maintain real-time performances while the erosion process or the wave simulation are run: only the cell minimum and maximum altitudes are retained over the regular erosion process. For the interactive terrain stamping, only the minimum and maximum altitudes of the modified cells need to be updated.

6 Conclusions and Future Works

In this paper, we have introduced a new hybrid method designed to render heightfield data sets with a good balance between speed, quality and flexibility, the latter allowing for terrains to be dynamic.

We combined two different rendering methods: a mesh rendering and a ray-casting algorithm using different levels-of-precision. To combine these methods, we proposed a heuristic that modulates the rendering precision according to the distance, viewing angle and relief features in order to maintain high rendering performance without visual quality loss. This heuristic also takes into account a user-given quality factor to balance between triangle projections where no error is committed and ray-based intersections.

Our rendering process requires neither complex pre-processing steps nor sophisticated accelerating data structures. It directly uses raw heightfield data. This allows us to edit terrains and to apply a mechanism to update the whole data like an erosion simulation process or a simple wave simulation, both in real-time. Compared to other terrain rendering techniques, our method is simple to use and easily allow dynamic heightfield data sets.

The rendering quality we obtain is good compared to standard terrain rendering methods, such as geometry clipmaps, since we use mesh and high precision ray-casting for the most salient relief features. Crest lines and small features of the relief at the horizon are also well preserved. However, as previous research has already shown, ray-casting remains globally slower than adaptive mesh based rendering, which hinders one to obtain high frame rates. This can be considered as the main counterpart of improved flexibility and rendering quality. But flexibility and quality of the rendering is important for some kinds of applications like scientific data visualization or geomatic applications. Finally, our method provides better results than standard methods to render relief details with high frequencies maintaining a real-time frame rate. The cell division of the terrain and the independence between the cells provide a high degree of flexibility of our method. As a consequence, our method allows one to easily select parts of the terrains which are rendered or not. This might prove useful for applications which should render different parts of the terrain using different algorithms (to handle non-heightfield structures of a terrain, for example).

For the moment, no streaming mechanism is implemented to handle very large data sets. But since we use a cell-subdivision of the height data, the rendering of large data sets seems possible, provided we define an adequate loading and caching strategy, similarly to Dick et al. [5].

Finally, in our future works, we also would like to study aliasing problems alongside some form of simple and lightweight level-of-details mechanisms. Indeed, heightfields rendering algorithms generally suffer from aliasing, especially in faraway areas. We believe a ray-casting algorithm is inherently well suited to address accurately aliasing, for instance using adaptive over-sampling process.

Acknowledgements

We would like to thank the reviewers for their helpful comments.

Puget Sound and Grand Canyon data sets courtesy of USGS. Mont-Blanc, Como Lake and Dolomiti data sets courtesy of Aim@Shape. Como Lake texture manually stitched from data delivered by Bing maps (© Microsoft Corp., Harris Corp. Earthstar Geographics LLC)

References

[1] A. Asirvatham and H. Hoppe. Terrain rendering using gpu-based geometry clipmaps. In *GPU Gems 2*, chapter 2. Addison-Wesley Professional, 2005.

[2] L. Baboud and X. Décoret. Rendering geometry with relief textures. In *GI '06: Proceedings of Graphics Interface 2006*, pages 195–201. Canadian Information Processing Society, 2006.

[3] B. G. Becker and N. L. Max. Smooth transitions between bump rendering algorithms. In *SIGGRAPH '93*, pages 183–189. ACM, 1993.

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

[5] C. Dick, J. Krüger, and R. Westermann. GPU ray-casting for scalable terrain rendering. In *Proceedings of Eurographics 2009 - Areas Papers*, pages 43–50, 2009.

[6] C. Dick, J. Schneider, and R. Westermann. Efficient geometry compression for GPU-based decoding in realtime terrain rendering. *Computer Graphic Forum*, 28(1):67–83, 2009.

[7] M. Duchaineau, M. Wolinsky, D. E. Sigeti, M. C. Miller, C. Aldrich, and M. B. Mineev-Weinstein. Roaming terrain: Real-time optimally adapting meshes. In *Proceedings of Visualization '97*, pages 81–88. IEEE Computer Society Press, 1997.

[8] H. Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *Proceedings of Visualization '98*, pages 35–42. IEEE Computer Society Press, 1998.

[9] T. S. Jeong and J. Han. Per-pixel rendering of terrain data. In *Computational Science - ICCS 2006, 6th International Conference*, volume 3993 of *Lecture Notes in Computer Science*, pages 40–47. Springer, May 2006.

[10] R. Lario, R. Pajarola, and F. Tirado. Hyperblock-quadTIN : Hyperblock quadtree based triangulated irregular networks. In *Proceedings IASTED International Conference on Visualization, Imaging and Image Processing (VIIP)*, pages 733–738, 2003.

[11] F. Larue, L. Ammann, and J.-M. Dischler. A pipeline for the digitisation and the realistic rendering of paintings. In *8th International Symposium on Virtual Reality, Archaeology and Cultural Heritage*. Eurographics, november 2007.

[12] J. Levenberg. Fast view-dependent level-of-detail rendering using cached geometry. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 259–265. IEEE Computer Society, 2002.

[13] P. Lindstrom and V. Pascucci. Visualization of large terrains made easy. In *VIS '01: Proceedings of the conference on Visualization '01*, pages 363–371, Washington, DC, USA, 2001. IEEE Computer Society.

[14] F. Losasso and H. Hoppe. Geometry clipmaps: Terrain rendering using nested regular grids. volume 23, pages 769–776. ACM, 2004.

[15] S. Mantler and S. Jeschke. Interactive landscape visualization using GPU ray casting. In *GRAPHITE '06: Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*, pages 117–126. ACM, 2006.

[16] F. K. Musgrave. Grid tracing: Fast ray tracing for height fields. Research report no. rr-639, Yale University Dept. of Computer Science, 1988.

[17] J. Olsen. Realtime procedural terrain generation. October 2004.

[18] R. Pajarola and E. Gobbetti. Survey on semi-regular multiresolution models for interactive terrain rendering. 23(8):583–605, 2007.

[19] F. Policarpo and M. M. Oliveira. *GPU Gems 3*, chapter 18 Relaxed Cone Stepping for Relief Mapping. Addison-Wesley, 2007.

[20] F. Policarpo, M. M. Oliveira, and J. L. D. Comba. Real-time relief mapping on arbitrary polygonal surfaces. In *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 155–162. ACM, 2005.

[21] L. Szirmay-Kalos and T. Umenhoffer. Displacement mapping on the GPU - state of the art. *Computer Graphics Forum*, 27(6):1567–1592, 2008.

[22] N. Tatarchuk. Dynamic parallax occlusion mapping with approximate soft shadows. In *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 63–69. ACM, 2006.

[23] A. Tevs, I. Ihrke, and H.-P. Seidel. Maximum mipmaps for fast, accurate, and scalable dynamic height field rendering. In *SI3D '08: Proceedings of the 2008 symposium on Interactive 3D graphics and games*, pages 183–190. ACM, 2008.