A GPU Framework for the Visualization and On-the-Fly Amplification of Real Terrains

Yacine Amara[1], Sylvain Meunier[2], and Xavier Marsault[3]

[1] USTHB, Algeria
[2] SIC – Signal Image Communications, France
[3] MAP-ARIA, UMR CNRS 694, France

Abstract. This paper describes a GPU framework for the real-time visualization of natural textured terrains, as well as the steps that are needed to populate them on-the-fly with tens of thousands of plant and/or mineral objects. Our main contribution is a robust modular architecture developed for the G80 and later GPUs, that performs texture/seed selection and rendering. It does not deal with algorithms that procedurally model or render either terrain or specific natural objects, but uses them for demonstration purposes. It can be used to calculate and display realistic landscapes and ecosystems with minimal pre-stored data, CPU load and popping artefacts. It works in tandem with a pre-classification of available aerial images, and makes it possible to fine-tune the properties of objects added to the scene.

1 Introduction

In real-time terrain visualization, altimetric and photographic database resolution quickly becomes insufficient when the viewpoint approaches ground level. While limiting memory load and computing costs, one may add, as soon as necessary, geometrical and textural details for the ground, resulting in the emergence of sets of plant and mineral objects adapted to the viewpoint requirements. Obtaining such an enriched reality from restricted data sets is known as "amplification". The term was first mentioned in [23]. This feature is important in many applications dealing with terrain rendering (e.g. flight simulation, video games).

Earth navigators such as Google Earth are designed to stream and display digital models of real textured terrains without amplification. On the other hand, Microsoft "Flight Simulator", without providing access to real aerial textures, implements a kind of amplification by mapping pre-equipped generic patterns onto the ground. A first attempt to reconcile the two approaches was made with Eingana [10] by the French company EMG in 2001. Based on limited-resolution databases, and using various analytical tools, Eingana generated a fairly realistic 3D planet by fractal amplification. But all such applications suffer from inaesthetic visual popping artefacts. Level-of-detail management on the GPU is therefore an important challenge in this context.

Seed models for terrain amplification such as [26] have already been presented, but not as full GPU real-time systems. We attain this goal on the G80 GPU by the use of robust techniques (render-to-texture, asynchronous transfer, multi-render-target, stream reduction, instancing). This is our main contribution in the present paper.

G. Bebis et al. (Eds.): ISVC 2007, Part I, LNCS 4841, pp. 586–597, 2007.

After a short overview of previous work (section 2), we give an outline of our technique (section 3), then exhaustive details of our implementation (section 4). For our experimentation, we use databases from the Haute-Savoie region in the French Alps (a large area of 4,388 km^2) provided by [22]: a 16 m/vertex "digital elevation model" (DEM) and a set of 50 cm/pixel aerial textures. The results are presented and discussed in section 5, followed by the conclusion.

2 Previous Work

2.1 Data Structures for Large-Area Texture Management and Visualization

Terrain navigators currently handle tens of GB of geo-referenced textures, whereas the GPU memory barely reaches one GB. Long before graphics cards became programmable, efficient main-memory resident data structures for visibility and hierarchical optimization [28], and corresponding CPU algorithms, were designed to provide the textures strictly necessary to display the current viewpoint. This always imposed constraints on the ground mesh, except for clipmaps [25]. Little work has been done on the handling of such structures on the GPU. But in 2006, A.E. Lefohn's [16] describes a complex C++ library that would handle them in this way. In 2005, S. Lefebvre shows in [15] how to work in a "virtual texture space" on the GPU, for arbitrary meshes. He also builds a streaming architecture for progressive and dynamic texture loading. This framework is used to traverse virtual-texture tile structures by carrying out all the complex and expensive computations on the GPU.

2.2 Seed Models

Seed models are extremely useful in the procedural specification and instantiation of landscapes. They avoid the need to precompute and store millions of plant and mineral objects just to enrich a few square kilometers. Exploiting the high redundancy that is present in nature, they make it possible to populate terrain on the fly without storing all the objects and their properties, while preserving the variety of the results.

We consider three levels of description for seeds. A single seed manages the placement of each object. It is described by a property vector: species, position on the ground, size, orientation, color, level of detail (lod). A seed cluster contains some closely-spaced objects which are processed as a single seed. Metaseeds are seeds of seeds, and store properties such as object density and activation radius. Visible objects in aerial textures (trees, shrubs, rocks) are described on the first two levels, while smaller objects (plants, flowers, piles of stones, etc.) that are invisible and numerous justify a "metaseed" description. At the seed level, it is easy to take into account geographical (altitude), climatic (moisture), seasonal (snow in winter), temporal (growth, ageing) rules, using a parameterized seed model. At the overall level, botanical and biological rules may be added to synthesize spatial distributions and specify metaseeds. Seed patterns are used to store such distributions and compose virtual patchy landscapes. Well described in [8], [9], [17], [15], they usually consist of squared generic sets of seeds or seed clusters. They can store one or more species, which makes them useful as a way of simulating varied and adaptable ecosystems. Random or controlled variations can be introduced on this level to avoid strict repetition of generic

fittings. At the pattern-instancing level on the ground, distributions usually take account of local species density. The use of patterns has several advantages: genericity, storage saving, reduced number of primitives sent to the graphics card. A first GPU implementation of this technique to generate random forests can be found in [15].

2.3 Real-Time Plant Rendering

Some of the tools – e.g. [1], [18] – which are used to generate realistic 3D plant models, are unsuited to real-time rendering of dense scenes. Alternative representations (volumetric textures, image-based or point-based approximations) which are more efficient, but also maintain good visual quality, have been designed. Volumetric textures were introduced by A. Meyer and F. Neyret [19] and improved by P. Decaudin [6], making it possible to render several thousand trees in real time. The method presented in [27] for real-time grass rendering with patches uses a combination of geometry-based and volume-based approaches. In both cases, it is difficult to reach the individual level of objects, which rules these methods out for seed rendering in the context of amplification. The point-rendering method proposed by G. Gilet [12] borrows some good ideas from [5] and provides continuous detail management. It reduces the number of points when the "tree to viewer" distance increases, and retains the original object geometry for close-up viewpoints. But the algorithm remains ill-adapted to high frame rates. Simple billboarding has been widely used for plants in real-time applications, originally to replace a set of polygons by a rectangle facing the user, onto which a semi-transparent texture is projected, representing the plant for a given viewpoint. Unfortunately, the objects lack relief and parallax, even if two crossed polygons or view-dependent textures are used. Billboard clouds, by creating volume, improve realism, but not thickness. X. Decoret has proposed in [7] a simplification method for generating sets of static billboards that approximate plant geometry, offer a good parallax and give an impression of depth. And A. Fuhrmann has improved his algorithm to cope better with trees [11]. He can generate realistic models of a few dozen billboards, with several levels of detail.

3 Outline of Our Technique

Our GPU architecture is dedicated to the realtime visualization and amplification of natural textured terrains, based on 4 modules. We suppose our ground to be covered by a virtual regular grid (RG) of tiles (Fig. 2). Each module is associated with a GPU rendering pass at a different level of abstraction: "tile selection", "ground texture rendering", "seed selection" and "seed rendering". This framework is generic in that it can be used with different algorithms for terrain mesh and seed-instance rendering, and the corresponding passes deal with arbitrary ground meshes, since they work at the GPU vertex level. For this purpose, we adapted the texture-streaming architecture [15] to large terrains, and improved it in many aspects (section 4).

Following [15], a module called TLM (Tile Load Map) computes the useful tiles lists for the viewpoint in one render-to-texture pass. It also deals with their visibility in the frustum, occlusion and level of detail. This information is sent in a 2D texture to a second module which stores, analyzes and compares the lists for frame

coherence, and decides which tiles to send to the GPU. Another module loads the required textures asynchronously into protected "caches" on the GPU memory. During the second pass, the ground geometry is sent to the GPU to be fully textured.

The on-the-fly terrain amplification concept does not allow for the pre-storage of seeds. We therefore developed an adaptive seed model - inspired by previous publications on virtual ecosystem simulation – in which seed placement is constrained by the aerial images, whose visible detail must be finely restored. We also ensure a color relationship between aerial images and the generated objects. Upon the launch of the application, a set of P generic square patterns containing random virtual seeds is produced (only their 2D positions are computed), and stored in the GPU memory. We design patterns whose number of seeds approximate the plant or mineral density for a typical zone. Plant and mineral objects are also cached in the GPU memory.

But there is no existing real-time method for computing "land cover classification" (LCC) for each pixel. Inspired by the works of S. Premoze [21], we use a pre-classification step for aerial images, which is the current trend. It draws on recent cooperation with INRIA\MISTIS [4] and IGN\MATIS [24] using statistical modelling applied to image analysis. We demonstrate in section 4.4 how to use LCC types (in a density-map like approach) to extract seeds on the fly and produce a plausible scene. Only the "single seed" model is fully implemented in our work.

4 Implementation Details

Ground textures are packed into a quadtree made up of 512 x 512 pixels tiles, stored on disk. Each one has its own mipmap pyramid, compressed into a 1:8 lossy format DXT1 (to minimize texture-volume transfer and GPU decompression time). The ground geometry of each frame (computed by the "terrain algorithm" – see Fig. 2) is cached in a VBO during an initial "dummy rendering pass", while the depthmap is also stored in the GPU memory (for use in sections 4.2 and 4.4 with a texture access). The terrain heightmap texture is loaded onto the GPU for an instant access in pass 3.

4.1 Encoding and Packing LCC Types in a Quadtree

We must store them in a quickly accessible data structure. Using the terrain vertex structure (16 m spacing) would involve an excessive loss of precision in the seed placement. It is preferable to store LCC types in textures, on disk, for access in a quadtree, since we have to retrieve them at different levels of detail (trees being visible up to 3,000 m, their LCC types are spread over multiple levels in the quadtree).

The coding scheme (Fig. 1) is not commonplace, because ground textures mapped to the terrain are not necessarily of the highest resolution at which we have to locate the LCC types. All these constraints impose a redundant coding that should be consistent between all the relevant levels, if popping objects are to be avoided. Moreover, we cannot pack ground colors and LCC types together in the same 32-bit RGBA tiles, because the latter must not be compressed.

In sum, a specific "LCC quadtree" is needed to pack LCC types into luminance tiles. We use 3 levels for trees. To ensure consistency between these different levels, we propose the following coding method, which allows a 1 m positioning precision

on the ground (which is sufficient for mountains). Level 1 uses 8 bits to store each LCC type resulting from the classification. We duplicate these types on 4 neighbors at level 0. Lastly, level 2 uses 8 bits to encode 4 neighboring texels from level 1, meaning that 3 different species can be stored on 2 bits (0 is kept for bare ground). We can increase the coding precision and the number of species supported by giving more bits to the LCC texture format. With 16 (resp. 32) bits, 15 (resp. 255) species can be addressed, although this means using more GPU memory and bus bandwidth.

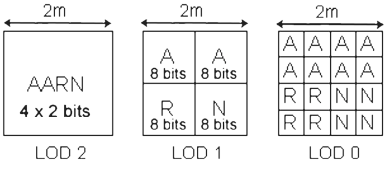

Fig. 1. LCC type quadtree-packing diagram (A, R and N are examples of LCC types)

4.2 The "Tile Load Map" (Pass 1)

This section deals with the core component of the tile-streaming architecture: the "Tile Load Map (TLM)" algorithm (Fig. 2). This module is dedicated to the selection of all the geo-referenced tiles needed to display the current viewpoint: ground tiles, associated LCC types and an aperiodic set of seed pattern indices. Since all are designed to share the same size, the computation is performed in a unique render-to-texture pass, using the subtle vertex (VS1) and fragment (FS1) programs, as described below. It produces the TLM: a 4-component texture of the same size as the RG is read back to the CPU (this is obligatory). After an analysis step, the TLM is expressed in 3 maps: the "Ground_texture Load Map (GLM)", the "Classification Load Map (CLM)" and the "seed_Pattern Load Map" (PLM), which can transmit the appropriate tiles and pattern indices to the GPU, with minimal OpenGL calls.

First, as described in [15], the ground geometry is drawn in the virtual texture space of the TLM corresponding to the entire terrain. Global texture coordinates (u_g, v_g) are computed in VS1 on the basis of 2D world vertex coordinates, which are sent to FS1 in the POSITION semantic. The corresponding vertex screen coordinates (SC) are computed with the ViewProject matrix for the current viewpoint. We also calculate the distance D between the viewpoint and the current vertex. SC and D are sent in TEXCOORDi semantics. The TLM computes 4 features for each tile: a visibility index (v_index) and 3 lod parameters (tlod_min, tlod_max, radial_lod).

Visibility_index. In FS1, up to 6 clipping planes are used to cull the ground geomentry polygons in texture space. SC coordinates are used in an occlusion-culling test to access the depthmap and discard hidden tiles. All culled tiles return a 0 value for the v_index. Strictly positive values, corresponding to visible tiles, are then used to store up to 255 possible seed pattern indices, which are computed on the fly, for each frame. In fact, this is an intuitive, and indeed better, solution than precomputing them on the CPU, for two reasons : it avoids storing memory-consuming arrays in the CPU or GPU for large terrains, and it allows some geographical, seasonal and temporal rules to be taken into account for optimal placement in VS1. Up to now, we

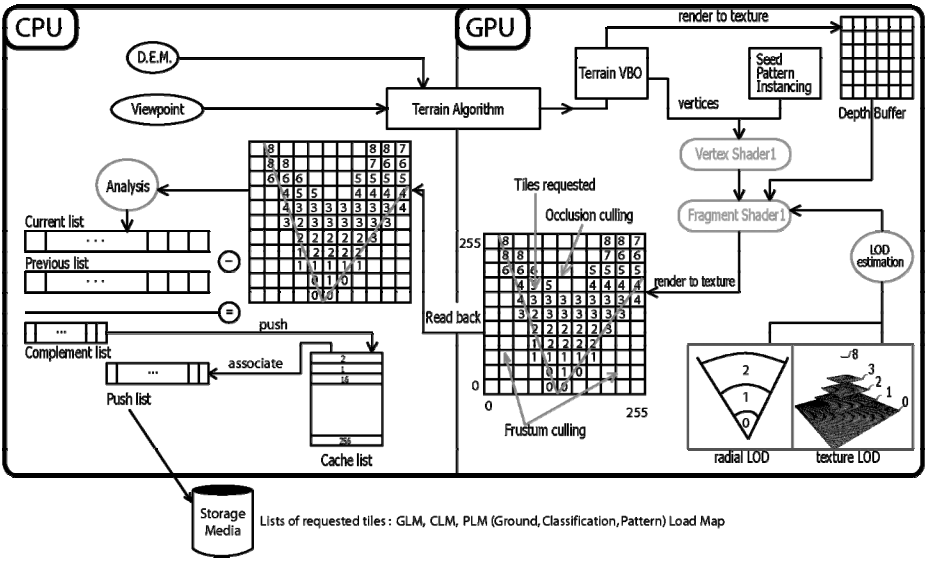

Fig. 2. The "Tile Load Map" process for the current viewpoint

have used a GPU-translation of the Park and Miller space-time-coherent, aperiodic, pseudo-random generator described in [20]. Each time the application is launched, the same seeds appear at the same place, depending only on a germ choice.

LOD estimation. Two different mechanisms are used in FS1 to estimate the required tile lods. For the ground tiles, we use the direct mipmapping estimation provided by the GPU to minimize the amount of textures needed to display the current viewpoint. For this purpose, a dedicated "LOD-texture" is created and cached on the GPU. Its mipmap pyramid is filled with integer values representing rising levels, starting from 0. In FS1, a simple access to this texture with SC coordinates gives the current lod value for each fragment. Here, we have to take account of a corrective parameter related to screen resolution in order to produce good mipmap estimations in texture space.

As regards LCC tiles and nested seed patterns, we use a different technique, given that the mipmapping mechanism is not suited to them: a user on the ground or at the height of a peak is likely to obtain horizontal polygons whose high lod values would result in erroneous LCC readings, and prevent the display of the trees. To eliminate this problem, the distance D is used to evaluate a radial lod: when a pattern is used at a distance of less than 3 000 m, a radial selection is computed for the first 3 lods of the LCC quadtree, with thresholds of 700 m, 1 400 m and 3 000 m. The fourth component of the structure is then used to store this radial level (0, 1 or 2), value 3 being reserved for non-active patterns.

TLM readback to CPU memory. The 4 TLM features are written in the output structure of FS1 with the "max blending" and polygon antialiasing options turned on (respectively for cumulative results and conservative rasterization per tile). To optimize transfers, we use the 8-bit/component BGRA texture format.

TLM analysis on the CPU. This stage simultaneously manages lists of tiles likely to be downloaded to the GPU (in one or several 'Texture-cache') with conservation of temporal coherence. It optimally ensures the nesting between useful tiles extracted from the GLM/CLM and the corresponding tiles in their respective quadtrees. Here, the tlod_min and tlod_max that estimate the lod range of a tile are used in conjunction with the v_index to select grouped tiles in the ground quadtree and build the "currentList". We maintain Texture-cache coherence by using a stack containing free positions in the "cacheList" (Fig. 2). Comparisons between the currentList and the previousList result in a third list called "pushList", which stores the indices of textures to be loaded up from the hard disk. Unused locations in the Texture-cache are pushed into "cacheList" using a "complement list". Given that on the G80, 8192 x 8192 pixels are available for Texture-cache, no overflow management for "cacheList" is necessary, and this saves computing time on the CPU. At the end of this pass, the required seed-patterns are grouped by type, so as to optimize the number of VBO calls in pass 3.

4.3 Ground-Tile Rendering (Pass 2)

The required ground tiles are first downloaded into the corresponding Texture-cache, which manages a tile pool of up to 256 locations on the G80. A rectangular luminance texture coding the quadtree indices is sent to the FS2. For texturing the current fragment, (u_g, v_g) are processed in a loop between tlod_min and tlod_max with a series of scaling and translating transformations, so as to obtain the associated tile location in Texture-cache, and the corresponding relative coordinates. The fragment color can then be read. Our solution to solve the discontinuity problem that appears in mipmapping in this cache involves a simple process using precomputed borders for the internal tiles.

4.4 Visible Seed Selection (Pass 3)

To retrieve visible seeds with minimal computing time, only visible patterns (in a radius corresponding to the visibility range of a tree) are extracted and stored in the PLM (section 4.2). In the same way as for the tiles, the graphic pipeline is then programmed for a "one-pass rendering" in a 2D texture called SLM (Seed Load Map). The size of this texture is at most 512 x 512 pixels, which can store enough seeds to populate large scenes. All the seeds are processed by the graphics card, and sent from the PLM by OpenGL calls to the patterns, which are 1D arrays initially cached on the GPU (Fig. 3) in vertex buffer objects (VBO). Each seed stores 8 GPU-computable properties: 3D position on the ground, LCC type (species), surrounding sphere (center, radius), level of detail, size, orientation, ground color, and an "index" within the VBO. The surrounding "sphere" qualifies the seed instance, whose center is placed at mid-height from it. The "ground color" is designed for a colorimetric follow-up of the terrain (sections 4.4 and 4.5). Pseudo-random functions are introduced in order to disturb size, orientation and color, and to obtain non-redundant variations, as in nature. Except for the "index" (which is used to compute the output position in the SLM), scalar heights are transmitted to the output structure of the fragment program FS3, for use in pass 4. But, a texture can store up to 4 floating components. So the

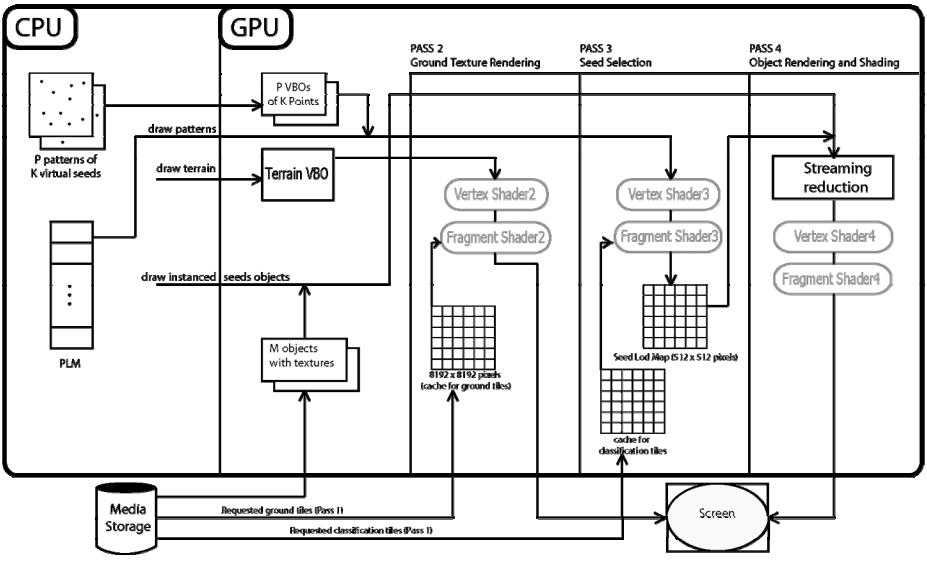

Fig. 3. Details of the seed and ground rendering passes

ARB_draw_buffers extension is enabled in order to activate the Multi Render Target (MRT). FS3 returns (+∞, 0,…, 0) if the corresponding seed is invalid, hidden, or outside the viewing frustum. The five following steps describe in details how seeds can be selected and stored in the SLM.

Seed positioning on the ground. Generic patterns only store seed locations (x, y), and so seeds are first positioned in 2D, using translations performed in VS3. This is followed by an orthogonal projection onto the ground mesh to compute the z value. Since the ground mesh is processed by a multiresolution algorithm, values for z may be view-dependent, except when the viewpoint is close to the ground. To avoid visual artefacts, all values of z must be the same, whatever the level of detail of the ground. We decided to anchor objects to the ground on their smallest z component. Each seed in the ground heightmap being surrounded by at most 4 basic vertices, its anchor point is obtained by a bilinear interpolation on 4 z values. Lastly, we ensure that the lower part of the trunk goes down a little way into the ground. This technique is particularly suited to amplified terrain meshes with detailed heightmaps whose amplitude is known in advance.

Seed visibility. This takes place in two steps, like the TLM process. VS3 performs a culling test: a seed whose surrounding sphere is totally exterior to the frustum is removed. Let R be the transformed radius of the sphere (taking account of on-the-fly random "size" computations), and ε a tolerance value. We project the center of the sphere into the "Normalized Device Coordinate" (NDC) space, then check if it belongs to [-1-R-ε, 1+R+ε] x [-1-R-ε, 1+R+ε] x [0, +∞[. The occlusion-culling test is performed in FS3, and discards a seed hidden by the ground (behind a mountain for

example) if its depth is greater than the value stored in the depthmap. For maximal precision, this value is computed at the "summit" position of the object projected onto the screen. For distant seeds, it is necessary to take account of the mesh simplification performed by the terrain multiresolution algorithm. An empirical tolerance distance of 50m is added so that visible objects will not be lost. Lastly, we enable the GL_CLAMP for the nearest mode of the depthmap texture, because an object whose summit does not appear on the screen may not have an associated depth.

Lod computation. This is performed by the VS3, which outputs a radial lod evaluation for each seed combined with its real size, so as to take into account the visual print during the lod transitions.

LCC value. The virtual seed species is instanced in FS3 for each seed : we access values in the LCC quadtree using the encoding scheme and the methods set out in sections 4.1 and 4.3. FS3 discards the current seed and returns (+∞, 0,…, 0) if no valid LCC type is found.

Ground color. This is accessed in FS3 using the same method given for ground-tile rendering in section 4.3. A random sampling of ground colors is carried out in the neighborhood of each seed. This allows fine modulations of plant or mineral colors to take place in pass 4.

4.5 Visible Object Rendering (Pass 4)

Although selection and rendering algorithms are independent, five parameters (stored in the SLM) ensure their nesting: species, size, orientation, lod and ground color. Our aim is to render selected entities with multiple hardware instancing calls, while avoiding a complete SLM readback in the CPU. Given M mineral and V plant objects, expressed in N_M and N_V lods, the number of instancing calls is $(M.N_M + V.N_V)$. To specify the "instancing stream" of each call, we just have to know the number of primitives to send, with instancing properties being accessed in VS4 with SLM readings. But the SLM contains not only "visible seeds" but also more numerous "holes", corresponding to discarded seeds. Since the SLM size is known, VS4 and FS4 could in fact directly process all vertices and fragments coming from seeds, and discard those coming from "holes", but this would be totally ineffective. A better way to tackle the problem is to use a GPU "stream reduction" algorithm, like the one described in [13]. This has several advantages: filtering capabilities that sort seeds by type and pack them without holes, and access to the number of packed seeds, which is transmitted to the CPU at very negligible cost using a readback of very few pixels.

The colorimetric ground follow-up naturally takes account of shadowed zones, and reduces unnecessary computations. The colorimetric transition is performed by a smooth blending with the ground textures at the rendering step of each instance. The foliage and trunks of the trees are processed independently, and the luminosity of the texels is modulated according to their relative positions within the object. Clouds (based on dynamic time-morphing textures) are also projected onto these objects and the ground textures, for more realism. All shading is performed in FS4.

Fig. 4. Two viewpoints, at different distance, of the full textured and amplified terrain (with the permission of use from "Régie de Données des Pays de Savoie RGD73-74")

5 Experiments and Results

All the codes were developed in C++, OpenGL, Cg and GLSL. The following tests were performed at 1280 x 1024 (res 1) and 1024 x 768 (res 2) display resolutions, on a computer with a 2.93 GHz CPU, 2 GB RAM and a Geforce 8800GTX. Only trees were used for the experiments: the maximum forest density is 21,000 trees/km^2. Computing times for the 4 passes are shown in Table 1 and give an overall impression of the performance of our system. For demonstration purposes, RMK2 [3], based on an improved version of the SOAR algorithm [17], was the library chosen to carry out the ground-geometry rendering. An improved implementation of our terrain algorithm using geometry clipmaps [2] is in progress, and will provide higher frame rates.

Table. 1. Performance analysis of the 4 passes, for 3 typical viewpoints and a large number of checked seeds per frame. Computing time for pass 3 depends only on the number of checked seeds which is related to the overall plant density. The influence of resolution is obvious for pass 2 and pass 4 results, the latter being the most time-consuming part of our seed architecture. The ratio selection/generation time of models to rendering time may be improved later. Global comparisons with CPU techniques are difficult, since few other works have been carried out in this field. Nevertheless, a rapid study shows that the TLM and SLM algorithms on the GPU seem almost 10 times faster than their CPU implementation.

Viewpoint	1		2		3	
Resolution	1	2	1	2	1	2
Terrain mesh (ms)	3.84	3.5	4.55	4.37	1.2	1.1
Pass 1 (ms)	**1.42**	**1.4**	**1.56**	**1.49**	**1.19**	**1.16**
Pass 2 (ms)	**2.04**	**1.78**	**2.23**	**1.82**	**1.62**	**1.4**
Pass 3 (ms)	**2.87**	**2.78**	**2.99**	**2.83**	**2.5**	**2.4**
Pass 4 (ms)	**4.52**	**3.98**	**5.91**	**5.03**	**3.08**	**2.82**
Selected seeds	5 102 / frame		23711 / frame		3164 / frame	
Checked seeds	115 712 / frame		141 728 / frame		101 227 / frame	
FPS (Hz)	**68**	**74**	**58**	**64**	**104**	**112**

For trees, we use billboard clouds (our special thanks go to A.Fuhrmann) with 3 lods composed respectively of 17, 8 and 2 polygons textured with 16-bit luminance atlas images of at most 4096 x 2048 pixels.

We process and read back a 256 x 256 pixel TLM in pass 1: this does not really affect the performance of the application, provided that this size is not exceeded. Frustum and occlusion culling substantially reduce the computing costs, both in CPU and GPU, especially for pass 3. In pass 2, the transition between levels 0 and 1 of the ground quadtree occurs at an optimal 700 m distance from the ground (corresponding to 1 pixel = 1.14 texel). With this adjustment, two mipmap levels per tile only, and the 4x anisotropic filter activated, we obtain good display quality (Fig. 4), and at a low cost, without overloading the graphics bandwidth.

6 Conclusions and Prospects

We are putting forward a robust and complete GPU approach in 4 passes for the visualization and on-the-fly population of large-textured (and even ground-mesh-amplified) terrains in real time, with tens of thousands of natural elements. We use a pre-classification of ground textures, and powerful streaming and instancing algorithms. For each frame, the selection of textures and seeds is performed entirely on the GPU, driven by an optimal combination of two visibility algorithms. The choice of the rendering algorithms is thus independent of the previous step. Our animations contain no popping effects that could be due to aggressive LODs or rough simplifications. One of our future tasks will be to process “clusters” and “metaseeds” in order to obtain detailed ecosystem display. A challenge will be to apply smoothly-controlled on-the-fly amplification techniques to all textures, in conjunction with seasonal behavior. The TLM algorithm will also be extended to deal with planetary systems.

References

1. AMAP - botAnique et bioinforMatique de l’Architecture des Plantes, http://amap.cirad.fr/
2. Asirvatham, A., Hoppe, H.: Terrain Rendering Using GPU-based Geometry Clipmaps. In: GPUGems2, Addison-Wesley, Reading (2005)
3. Balogh, A.: Real-time Visualization of Detailed Terrain. Thesis of Automatic and Compuing University of Budapest (2003)
4. Blanchet, J., Forbes, F., Schmid, C.: Markov Random Fields for Recognizing Textures Modeled by Feature Vectors. In: International Conference on Applied Stochastic Models and Data Analysis, France (2005)
5. Dachsbacher, C., Vogelgsang, C., Stamminger, M.: Sequential Point Trees. ACM Trans. On Graphics (2003)
6. Decaudin, P., Neyret, F.: Rendering Forest Scenes in Real-Time. In: Eurographics Symposium on Rendering, Norrköping, Sweden (2004)
7. Decoret, X., Durand, F., Sillion, F.X., Dorsey, J.: Billboard Clouds for Extreme Model Simplification. In: Proceedings of the ACM Siggraph (2003)
8. Deussen, O., Colditz, C., Stamminger, M., Drettakis, G.: Interactive Visualization of Complex Plant Ecosystems. In: Proceedings of the IEEE Visualization Conference, IEEE Computer Society Press, Los Alamitos (2002)

9. Deussen, O., Hanrahan, P., Lintermann, B., Mech, R., Pharr, M., Prunsinkiewicz, P.: Realistic Modeling and Rendering of Plant Ecosystems. In: Computer Graphics, Siggraph (1998)
10. Eingana: Le premier atlas vivant en 3D et images satellite, Cdrom, EMG (2001)
11. Fuhrmann, A., Mantler, S., Umlauf, E.: Extreme Model Simplification for Forest Rendering. In: Eurographics Workshop on Natural Phenomena (2005)
12. Gilet, G., Meyer, A., Neyret, F.: Point-based Rendering of Trees. In: Eurographics Workshop on Natural Phenomena (2005)
13. Roger, D., Assarson, U., Holzschuch, N.: Whitted Ray-Tracing for Dynamic Scenes using a Ray-space Hierarchy on the GPU. In: Proc. of the Eurographics Symp. on Rendering (2007)
14. Lane, B., Prunsinkiewicz, P.: Generating Spatial Distributions for Multilevel Models of Plant Communities. In: Proceedings of Graphics Interface (2002)
15. Lefebvre, S.: Modèles d'Habillage de Surface pour la Synthèse d'Images. Thesis of Joseph Fourier University, Gravir/Imag/INRIA. Grenoble, France (2005)
16. Lefohn, A.E., Kniss, J., Strzodka, R., Sengupta, S., Owens, J.D.: Glift: Generic, Efficient, Random-access GPU Data Structures. ACM Transactions on Graphics (2006)
17. Lindstrom, P., Pascucci, V.: Terrain Simplification Simplified: a General Framework for View-dependant Out-of-core Visualization. IEEE Trans. on Vis. and Comp. Graphics (2002)
18. Lintermann, D., Deussen, O.: Xfrog, http://www.xfrogdownload.com
19. Meyer, A., Neyret, F.: Textures Volumiques Interactives. Journées Francophones d'Informatique Graphique, AFIG, 261-270 (1998)
20. Park, S.K., Miller, K.W.: Random Number Generators: Good Ones Are Hard To Find. Com. of the ACM 31, 1192–1201 (1998)
21. Premoze, S., Thompson, W.B., Shirley, P.: Geospecific Rendering of Alpine Terrain. Department of Computer Science, University of Utah (1999)
22. RGD73-74: Régie de Gestion des Données des Deux Savoies, http://www.rgd73-74.fr
23. Smith, A.R.: Plants, Fractal and Formal Languages. In: Proceedings of Siggraph (1984)
24. Trias-Sanz, R., Boldo, D.: A High-Reliability, High-Resolution Method for Land Cover Classification Into Forest and Non-forest. In: 14th Conf. on Image Analysis, Finland (2005)
25. Seoane, A., Taibo, J., Fernandez, L.: Hardware-independent Clipmapping. In: WSCG 2007. International Conference in Central Europe on Computer Graphics, Czech Republic (2007)
26. Wells, W.D.: Generating Enhanced Natural Environments and Terrain for Interactive Combat Simulation (GENETICS), PhD of the Naval Postgraduate School (2005)
27. Boulanger, K., Pattanaik, S., Bouatouch, K.: Rendering Grass in Real Time with Dynamic Light Source and Shadows. Irisa, internal publication n°1809 (2006)
28. Bittner, J., Wonka, P.: Visibility in Computer Graphics. Jour. of Env. and Planning (2003)