Volume 26 (2007), Number 3

Ray-Casted BlockMaps for Large Urban Models Streaming and Visualization

paper1195

Abstract

We introduce a network- and GPU-friendly technique that efficiently exploits the highly structured nature of urban environments to ensure rendering quality and interactive performance of city exploration tasks. Central to our approach is a novel discrete representation, called BlockMap, for the efficient encoding, transmission and rendering of a small set of textured buildings far from the viewer. A BlockMap compactly represents a set of textured vertical prisms with a bounded on-screen footprint. BlockMaps are stored into small fixed size texture chunks and efficiently rendered through GPU raycasting. Blockmaps can be seamlessly integrated into hierarchical data structures for interactive rendering of large textured urban models. We illustrate an efficient output-sensitive framework in which a visibility-aware traversal of the hierarchy renders components close to the viewer with textured polygons and employs BlockMaps for far away geometry. Our approach provides a bounded size far distance representation of cities, naturally scales with the improving shader technology, and outperforms current state of the art approaches. Its efficiency and generality is demonstrated with the interactive exploration of a large textured model of the city of Paris on a commodity graphics platform.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Three-Dimensional Graphics and Realism]: Color, shading, shadowing, and texture; I.3.7 [Three-Dimensional Graphics and Realism]: Raycasting; I.3.7 [Picture/Image Generation]: Viewing Algorithms;

1. Introduction

The enormous success of easy-to-use web-based 3D map browsers, such as Google Earth, NASA WorldWind, or Microsoft Virtual Earth, is renewing the interest in technology for distributing and rendering massive high quality 3D representations of the Earth. While, historically, frameworks for distributed visualization of multi-resolution representations of the planet have concentrated on photo-textured digital terrains, the interest is now shifting to urban environments, if only because of all the related business opportunities. As a consequence, entire cities are being digitalized and a large amount of new data will be available in the near future, especially because automatic 3D reconstruction technologies are starting to be developed and successfully applied to urban complexes [TAB*03, FJZ05, CCV06].

Network streaming and real-time visualization of virtual cities is a very challenging problem because of the great complexity of urban scenery, which combines large extents with very rich small scale visual details. Models are made of many small connected components (single or small group of buildings), each one usually represented with a relatively small number of polygons but a large amount of color information, typically one photograph for each façade of the building. Moreover, geometry is highly discontinuous and different views of the model have widely different depth complexity, ranging from full visibility of flyovers to nearly full occlusion at ground level.

This kind of 3D model does not fit well within classical multiresolution/CLOD rendering and incremental streaming schemes. While multiresolution texturing and geometric levels of details may provide acceptable solution for objects near to the viewer and moderate degrees of simplification, major problems arise with distant objects. Because of limited image resolution and perspective projection, typically several triangles and a large amount of input texels project to a single pixel, making joint filtering of geometry and texturing an important issue. Relying on surface meshes to drive such simplification is very complicated, as the triangles mapping on a given pixel may stem from disconnected surfaces, or totally different objects. For such reasons, image based ap-

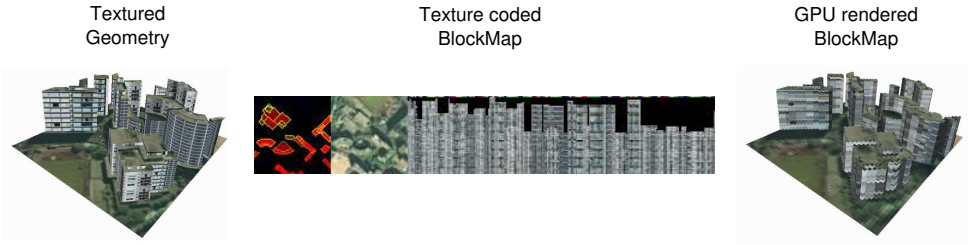

Figure 1: *The BlockMaps idea: replace far away groups of buildings with a concise texture-based representation, called* ***BlockMap****, which is efficiently rendered by a GPU ray caster.*

proximations (impostors), have been proposed as a mean to replace to the original data with good visual accuracy. However, away from their supported set of views, impostors introduce rendering artifacts, such as parallax errors, disocclusions, or rubber sheet effect, which can only be reduced by introducing severe constraints on viewpoint motion or by radically increasing storage and transmission costs. The limitations of current techniques for simplification and incremental streaming of distant geometry thus impose important scalability limits on current networked viewers.

Contribution. In this paper, we introduce a network- and GPU-friendly simplified representation that efficiently exploits the highly structured nature of urban environments, and illustrate how it can be used to construct an efficient output-sensitive system. Central to our approach is a novel discrete representation for textured sets of buildings, that we call *BlockMap*. The BlockMap encodes both the geometry and the sampled appearance (color and normals) of a small group of simple buildings, represented by a collection of textured vertical prisms (see Figure 1). BlockMaps are stored into small fixed size texture chunks (e.g. 32x256 RGB textures), and can be efficiently rendered through GPU raycasting. Since BlockMaps are simplified representations of the original textured geometry, they provide full support to visibility queries, and, when built into a hierarchy, offer multiresolution adaptability. In this paper, we show their proficient use by incorporating them in a state-of-the-art framework based on multiresolution texture atlases, in which a visibility-aware traversal of the hierarchy renders components close to the viewer with textured polygons and employs BlockMaps for far away geometry. By working in a network setting, we also test the effectiveness of the BlockMap as a scalable output-sensitive LOD technique that is able to provide a bounded size representation for distant portions of a large urban scene.

Advantages. BlockMaps provide a compact, complete, bounded error, replacement for the original geometry in the classical LOD meaning, and can be seamlessly integrated into hierarchical data structures for incremental streaming and interactive rendering of large textured urban models. They are simple to code, GPU friendly, and rendering performance naturally scales with the improving shader technology. They are especially useful in a networked remote visualization framework, where they provide a size bounded representation of urban scene that can be progressively downloaded and displayed.

Limitations. As for all current large scale urban model streaming and rendering approaches, our method has also some limitations: it has been designed for static models, editing is not supported; being tuned for far-field representations, BlockMaps need to be combined with other techniques to provide an all-scales view; we have currently not implemented out-of-core compression and speculative prefetching – these are orthogonal to our method, but important to reduce perceived refinement latency.

Despite these limitations, the current method and prototype system is of immediate practical use and provides unprecedented performance in streaming and rendering very large complex models. As highlighted in Section 2, while certain other methods share some of *BlockMap*'s advantages, they typically do not meet its capability in all of the areas. The BlockMap data structure, and the algorithms required to construct and render them are presented in Section 3. We then briefly illustrate the structure of the overall multiresolution systems implemented to test the BlockMap approach (Section 4), and discuss the results obtained with complex textured city models (Section 5). We finally summarize our findings in Section 6).

2. Related work

Real-time rendering large urban (and more general) environments is an active research area. In the following, we will discuss the approaches most closely related to our work. Readers may refer to recent surveys (e.g., [LRC*02, KMS*06]) for further details.

Representing and rendering far geometry. The idea of using a different representation for distant geometry, that has a small, slowly changing on screen projection is central to many approaches for massive model rendering. Using a simpler geometric representations for far geometry is the core idea of *LOD* techniques, while switching to a radically different image based representation characterizes *impostor* approaches. The term impostor was introduced in [MS95],

where simple textured quads were suggested for replacing the rendering of more complex geometry. This approach has been later extended with the introduction of *Textured Depth Meshes* (very simple triangle meshes with depth values taken from rendering) [SDB97, DSV97] in order to alleviate parallax errors. In [DSSD99] the disocclusion error generated by associating points belonging to different surfaces to the same impostor is estimated, and objects are grouped in a way to minimize the error. In [JWS02, JW02], a collection of slices at increasing distance is used as impostor to form a *layered environment-map impostor*. Impostors based on *Layered Depth Images*, that for each pixel store all the intersections of the view ray with the scene, have been introduced in [SGwHS98] and later extended in [WWS01], where the regular sampling of LDIs is replaced with a more general adaptive point sampling of the geometry. Similarly to impostors, BlockMaps are stored in texture memory. However, the BlockMap representation is more similar to LOD than to impostor approaches, as a BlockMap provides a view-independent, simplified representation of the original textured geometry, provides full support to visibility queries, and, when built into a hierarchy, offers multiresolution adaptability. Encoding shape and appearance into a texture is also the goal of geometry images [GGH02], which enables the powerful GPU rasterization architecture to process geometry in addition to images. Geometry images focus on reparametrizations of meshes onto regular grids, while we focus on a specialized representation of urban structures.

Managing a large amount of textures. Massively textured urban models, where the facade of each building is represented by a different image, require the management of very large amounts of textures. The data management issue has been historically addressed in the context of terrain datasets with very simple parameterization (e.g., the Clipmap approach [TMJ98], that required specialized hardware, or the software based approach of [CE98]). These solutions, however, consider a single smooth, uniform texture parameterization, and are not directly applicable to detailed urban datasets with many vertical walls. Just a few solution were proposed for the more general situation of complex meshes with large texture information. The GoLD approach [BGB*05] is one of the most recent representatives in this area. It consists in a CLOD hierarchical data structure that partitions the geometry and the texture over it into patches. This approach, like most of the CLOD multiresolution schemes, is more oriented to the management of large unstructured meshes with simple topology than to scene composed by tens of thousands of single textured components. In [BD05] the authors presents a hierarchical structure called texture-atlas tree that is tailored, like our solution, to the management of large textured urban models. Their work, however, considers multi-resolution textures but single resolution geometry, and does not take into account network streaming issues. In our approach, we enrich the structure with BlockMaps for all inner nodes, providing adaptability both in terms of textures and geometry. Moreover, as demonstrated in the results section, single geometry and texture processing are tightly coupled, less information is needed for a BlockMap representation than for a subsampled atlas with geometry at the same quality.

GPU techniques The GPU technique used for rendering a BlockMap is strictly related with the issue of height field ray tracing and displacement mapping techniques [Coo84]. Recently, a number of specialized techniques have been presented to exploit the GPU for that purpose. The Relief Mapping approach was introduced in [OBM00], where a warping approach is used to find the final position of an orthogonally displaced texel over a given flat texture. An approach to the problem of rendering generalized displacement mapped surfaces by GPU raycasting was proposed in [WWT*03] and [WTL*04]. In these methods, the results of all possible ray intersection queries within a three-dimensional volume are precomputed and stored in a compressed five-dimensional map. While general, this approach implies a substantial storage overhead. Instead, we render BlockMaps by exploiting a small number of precomputed maps which compactly encode the heightfield geometry. Similarly to [Don05], distance maps are used for space leaping.

3. BlockMaps: many buildings in few pixels

The goal of the BlockMap representation is to compactly and faithfully represent a city portion as seen from a distance. For obvious scalability reasons, this means that the representation should be able to rapidly render many buildings projecting to few pixels, with a memory and time complexity ideally depending only on the region's screen footprint.

An important characteristic of urban models is that a set of textured vertical prisms typically provides a good approximation of the geometry and appearance of a region, well preserving the large scale features, i.e., the silhouette and color of the buildings. This assumption holds not only because the vast majority of buildings have a prism-like shape, but also because automated methods for ground-based acquisition of large-scale 3D city models impose geometric constraints on reconstruction to make the acquisition tasks more tractable.

Starting from this assumption, with BlockMaps we introduce a simple, height-field-like, efficient representation for a set of discrete vertical textured prisms. We exploit the regularity of this discrete description to obtain a very compact encoding and a fast GPU-friendly rendering algorithm, and use the discretization stepsize to control reconstruction quality.

3.1. The BlockMap data structure.

The BlockMap representation stores in a rectangular portion of texture memory all the data required to represent a group of textured vertical prisms aligned on a square grid. The data representation is tuned for being efficiently accessed at rendering time by a GPU-based raycasting algorithm.

The prisms representing buildings are coded by texel values, with each texel coding height and roof/ground texture

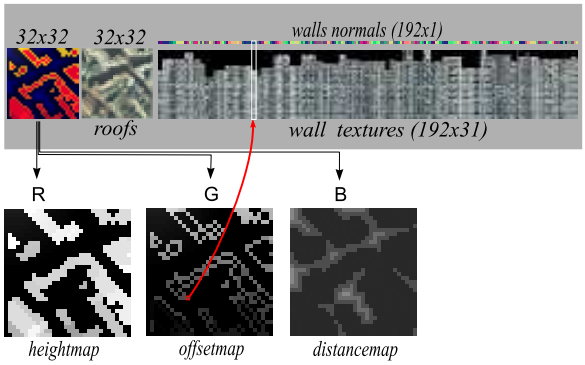

Figure 2: *In the upper part a the* 256×32 *texture coding a 32:192 BlockMap is shown. The bottom part shows the three channels of the first quad, named* geometry*, of the BlockMap. The red arrow illustrate the texture* 2 *fetches executed if a vertical wall is hit.*

information. For the texels corresponding to visible building walls, we also store a column of texture data as well as the wall normal.

Figure 2 illustrates our data structure with an example of BlockMap encoded in a 256×32 24 bit RGB texture.

The three channels of the leftmost 32×32 square are shown at the bottom of the figure, and they code the *height map*, the *offset map* and the *distance map*. Each texel of the *height map* stores the height value of the corresponding texel. The *offset map* is used to address the *wall texture/normal*, as detailed below, and the *distance map* stores a discrete distance field defined in the void inter-building space used to speedup raycasting by space leaping [Don05].

The second 32×32 square simply contains the roof/ground color information, and the rightmost 192×32 portion of the BlockMap encodes the shading information for the vertical walls. Each column in this last portion is associated with a single texel of the height map, and the u coordinate of the column is stored in the *offset map* (second channel of the first square). The first 31 texels of the column contains color data for the vertical wall, while the 32^{nd} texel contains its rgb-coded normal. In this rightmost 192×32 portion we store a texture column only for visible *wall* texels, i.e., texels having at least one of the 4 neighbors with a smaller height, that is, where at least one of their sides is partially visible. Note that in the offset map a non-null value is present only for the texels at the boundary of the buildings footprint, that we call *boundary texels*. Since we store only one column per boundary texel, and a boundary texel can have more than one side exposed, we average texture and normal values of each side, as explained in more detail when describing the construction process. Hereafter we will indicate a 256×32 texture thus partitioned as a 32:192 BlockMap.

Even though the example uses a a 256×32 24 bit RGB texture, other choices for the the texture size are obviously possible, and the best granularity for a BlockMap representation can be selected depending on the particular working conditions of the application. As discussed in Section 3.3, the size of the BlockMap clearly affects the accuracy of the representation as well as the loading/transmitting and rendering time.

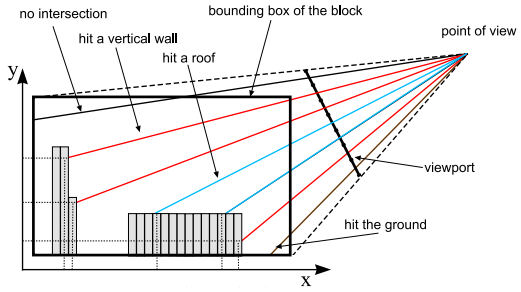

Figure 3: *A scheme of ray casting process. For each ray hitting a surface, the BlockMap provides access to normal and color to use for rendering.*

3.2. Raycasting BlockMaps.

The particular encoding of a BlockMap leads to an efficient GPU raycasting process. The fragments corresponding to rays entering the bounding box of the BlockMap are activated by the rasterization of the 3 visible faces of the bounding box. The fragment program computes, it if exists, the nearest intersection, and shades the fragment using color and normal information.

An example of how to raycast a BlockMap is presented in Figure 3, where for the sake of simplicity we use a section along the Z axis. The figure shows a number of different rays passing through a BlockMap's bounding box.

The intersection of a ray with the BlockMap is found by sampling the height map along the ray. Given a sampling point, its projection on the distance map is found (dropping the height coordinate), the corresponding value is fetch and used to place the next point along the ray.

The intersection procedure can exit with three possible cases:

- The ray exits the bounding box without hitting neither buildings nor the ground (see the black ray in Figure 3).
- The ray hits a roof/ground (blue and brown rays, respectively). The roofs and the ground are considered to have normal $n = (0,1,0)$ while the color is fetched on the *roofs texture* at the same *uv* coordinates as the hitting point.
- The ray hits a vertical wall. In this case the normal and color are stored in the *walls normals* and *walls textures*, respectively, at the same u coordinate, which is stored in the *offset map* at the same *uv* coordinates as the hitting point. The v coordinate to fetch the color is given by the height of the hitting point.

3.3. Building a BlockMap.

To compute the texture parameterization of the walls over the rightmost part of the BlockMap texture we simply unfold each building, by following its boundary in a linear way. With this simple approach, most of the times, adjacent boundary texels correspond to adjacent texture columns.

Figure 4 illustrates the process of computing texture and normal for a pixel-sized BlockMap column. Figure 4.(a) shows a height map where boundary texels are in cyan and non-boundary texels in blue/white. For each boundary texel, and for each boundary side, we compute a texture column that corresponds to the original textured geometry lying there as seen from that boundary side (shown as a red line-dot in the figure). In the case a boundary texel has more than a single side (like the one shown in Fig. 4.(b)) the computed texture columns are averaged together. Simple orthographic renderings of the original textured models are used to efficiently compute these columns.

If the final number of columns created is more than the size reserved for vertical texture (192 pixel in the case of a 32:192 BlockMap) the wall textures and normals regions are scaled along the u coordinate (and so are the values in the offset map).

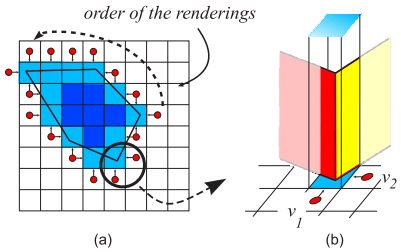

Figure 4: *Construction of a BlockMap: (a) Height map with boundary texels rendered in cyan. The red dots correspond to the locations from which the renderings are done. (b) Example of a boundary texel with two exposed sides, the two side renderings are averaged together.*

Accuracy. BlockMaps are a discrete representation of a height field made of $n \times n \times 256$ voxels, where n is the side in texels of the heightmap and 256 is the number of values to code the height. Consequently, the geometric Hausdorff error is bound by half of the length of the diagonal of a single voxel in object space, and the geometric error in screen space is the projection of such length. In order to bound the error in every point of the BlockMap, we consider the closest point of a bounding sphere of the BlockMap to the viewer as distance for the perspective division.

4. Multiresolution streaming and rendering with BlockMaps

As we previously discussed urban scenes exhibit a high topological complexity, that makes very difficult to find representations for drastically simplified levels of details that are

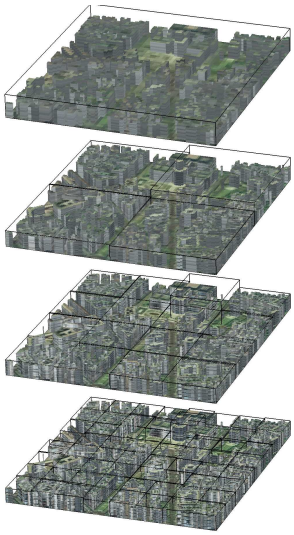

Figure 5: *The first four levels of the BlockMaps hierarchy. The same portion of the urban scene is covered with finer and finer BlockMaps arranged in a quadtree.*

compact in terms of rendering primitives as well as in storage and transmission requirements. BlockMaps, thanks to their discrete and geometry-independent nature, are able to compactly code the overall shape and appearance of a set of city buildings as a set of vertical prisms. Moreover BlockMaps have a easily predictable rendering and storage cost, that makes their use appealing in multiresolution and/or streaming frameworks.

Since BlockMaps are able to provide faithful far field representations for all directions and do not suffer from the disocclusion, undersampling, and distorted parallax problems of image based impostor techniques, their integration is not limited to systems based on view-cells, but can be seamlessly employed for far field rendering in any type of interactive visualization system adopting a hybrid multiresolution representation. By simply associating BlockMaps to larger and larger areas, it is in particular possible to create a quadtree like hierarchy of levels of detail.

Embedding BlockMaps in a Multiresolution Hierarchy. To demonstrate the ease of integration of BlockMaps into existing multiresolution frameworks, we have embedded the BlockMap representations in an out-of-core multiresolution hierarchy based on the texture-atlas tree approach introduced in [BD05]. In this framework, the spatial domain of a urban scene is organized in a quad-tree built by recursive space partitioning guided by a texture/geometry complexity criterion. In each quadtree node, a reference to the geometry data in its region is stored together with all the texture data mapped on such geometry, arranged in a texture atlas. The texture atlas of a leaf quad is composed by assembling original facade textures, while the atlas of an inner quad is obtained

by composing the downsampled versions (by a factor of 2 on each direction) of the texture atlases of its four children. A shortcoming of this approach is that it does not scale up well for very large urban environments composed by tens of thousands of buildings, and it has major limitations in a networked setup. For instance, this approach requires the transmission of complex geometry even for very coarse detail levels. This kind of data, with high topological and texture coordinate complexity is hard to compress. On the other hand, BlockMaps are a valid substitute for the coarser levels of the hierarchy, with the major advantage that they are well bounded both in terms of transmitting and rendering cost. For this reason, we simply build, over the urban domain, a quadtree hierarchy of finer and finer BlockMap representations by reusing the same spatial partitioning of the existing hierarchy. BlockMaps are constructed for all nodes from the root until either when the BlockMap representation becomes too crude for the geometry contained into the node, or the cost of directly storing/rendering the original texture and geometry becomes competitive with the cost of the BlockMap representation. The example in Figure 5 shows the BlockMaps composing the first four levels of the hierarchy.

Once we have built the BlockMap hierarchy, at runtime we can use the BlockMaps whenever the projected size of the bounding box of a node is less than a given threshold and the rendering error is therefore safely bounded. In a networked framework obviously the BlockMap representation is also used whenever other, more costly to be transmitted, representations are not still available. In our current implementation, the rendering procedure proceeds by visiting the hierarchy coarse-to-fine and front-to-back, pruning those nodes whose geometry is occluded or lays outside the view frustum, and stopping refinement when a node representation, BlockMap or textured geometry, is considered accurate enough in terms of projected texels/pixel, or when child data is not yet available in-core. In the latter case, asynchronous I/O requests are issued by a separate thread. As in [BWPP04], occlusion culling is performed by interleaving hardware assisted occlusion queries with rendering of leaf nodes visible in the previous frame to mask query latency.

Compressing and Streaming BlockMaps. As outlined above, BlockMaps can be used in a multiresolution framework for efficiently transmitting coarse representation of even very large cities. With BlockMaps the coarsest levels of the tree have a predictable limited size, a major advantage when managing remotely stored datasets. When transmitting a BlockMap, straightforward compression techniques can be applied to further improve performance. By considering the case of 32:192 BlockMaps a lossless PNG-compressed representation occupies roughly 16KB (average over 512 BlockMaps representing the first levels of the benchmark dataset – see Section 5). By means of prefilter smoothing prior to compression on the rightmost part of the bitmap (texture images), the compression ratio can be increased, reaching $4/8KB$ for each BlockMap for near-lossless results. For larger BlockMaps the compressed size linearly scales with the number of the texels of the BlockMap. Figure 6 shows the cumulative cost of the transmission of the first levels of a BlockMap hierarchy, for the case of 32:256 BlockMaps.

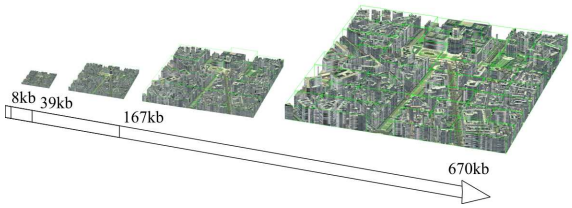

Figure 6: *Network streaming of BlockMaps: a urban scene is progressively transmitted. The numbers above the arrow reports the cumulative size of the transmitted BlockMaps.*

5. Results

An experimental software library and a rendering application supporting the technique have been implemented on Windows XP platform using C++, OpenGL, and GLSL shaders. We have extensively tested our system with a number of large urban models. The quantitative and qualitative results discussed here are for the Paris urban environment. The reported times were obtained on a dual core P4 @ 3GHz PC equipped with 2GB Ram, two HD 160GB SATA and a NVIDIA GeForce 8800GTX with 768MB. The geometry of the Paris dataset has been created from the cadastral maps, containing a vector representation composed by 80,414 building outlines, whose extrusion was tessellated to 3.7 millions triangles. The original dataset has no texture information, so, for the sake of testing, we have created and stored for each building a different 512^2 fake texture for its façades; overall, the texture information for the façades is composed by 19.63 Giga texel. In an uncompressed format this texture information would need almost 60 GB.

Preprocessing. For the Paris dataset the creation of all the geometry from the cadastral profiles, the geometric partitioning and the quadtree construction took less than a couple of minutes and generated a tree of 25,405 nodes with 18,978 leaves, with maximum depth of 10. The recursive partitioning of the tree targeted less than 16 buildings for each leaf. The construction of the atlas-tree took approximatively four hours starting from the original 80,414 512^2 facade textures and generating 25,405 atlases. The generated atlases were S3TC compressed into 14.1 GB dataset. In the case of 64:256 BlockMaps the creation of a tree of 4,178 BlockMaps, roughly corresponding to the first 7 levels of the tree, took 35 mins.

Rendering. In order to evaluate the rendering costs of BlockMaps we have performed various tests on our urban

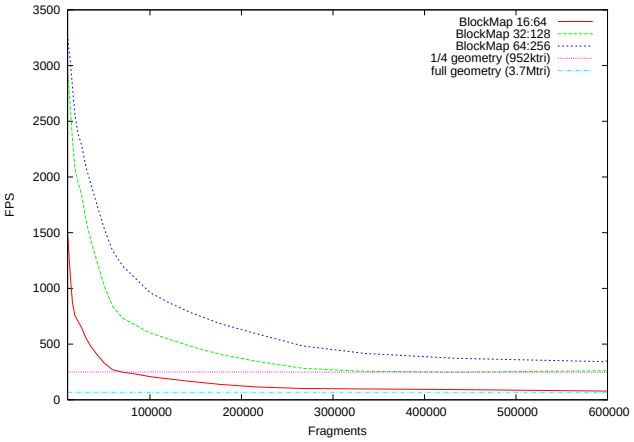

Figure 7: *Rendering rates of BlockMaps at different precision/granularity versus the original textured geometry.*

scene. To better estimate very short rendering times (a full screen single BlockMap can be rendered at more than 1kHz), we have chosen 50,000 random viewing directions at varying viewing distances. We have clustered viewpoints according to projected screen size of the BlockMap and we averaged together the times for one hundred of renderings with similar screen coverage. We repeated these tests both by simply drawing an entire level of the BlockMap hierarchy and by using the adaptive rendering strategy described in Section 4. In this latter case, the BlockMap hierarchy is recursively traversed, and BlockMap nodes are rendered if their screen projected geometric error is less than a pixel. The rendering of the textured geometric dataset in the test was done by using the textured-atlas approach described in Section 4.

The graphs in Figure 7 illustrate how BlockMap size and projected size affect rendering time. The rendering time of the BlockMaps of increasing size (from 16:64 to 64:256) were evaluated in comparison with the rendering times of the full, and of a quarter of, the original geometric Paris dataset.

The graphs highlight the strict relation between BlockMaps rendering time and the number of fragments covered by the projection of their bounding box, typical of all fragment shader limited rendering paths. Moreover the rendering of the original dataset (or a portion of it) is always more costly and independent of its impact on the screen. The size of the BlockMap obviously affects also the rendering, but it should be noted that smaller BlockMaps usually are *not* faster to render, because a deeper traversal of the tree is required to reach the same accuracy and more BlockMap atlases switches and shader uniforms updates are performed. Moreover, fine grained BlockMaps exhibit a higher overdraw ratio. On the other hand, it is also true that larger BlockMaps require more texture for covering the same area with the same texel densities, mostly because they represent walls with higher, better defined textures. Figure 8 illustrates the rendering cost of 64:256 BlockMaps for different accuracies and number of fragments covered. As expected, BlockMaps rendering time is strictly related to the number of fragments covered by the bounding box of the geometry that they are replacing, and coarser levels of the BlockMap tree are much faster to be rendered because of reduced overdrawing.

Integration in a multiresolution engine. The qualitative performance of our adaptive renderer combining BlockMaps and texture atlas is illustrated in an accompanying video, recorded live using a 1280x1024 rendering window resolution (see Fig. 9). The interactive sessions were designed to be representative of typical inspection tasks and to heavily stress the system. Note for example how in the first and last part of the flight, when the whole city is visualized, most of the geometry is substituted by BlockMaps, and the advantage over the traditional approach is very high, since we can sustain 100fps while guaranteeing a good visual quality. Similar gain can be also noted when the camera pan over the city at a close distance form the building roofs and the city extends up to the horizon covering almost the entire screen. Without using BlockMaps the rendering rate of the whole city would be limited to 60fps.

Streaming BlockMaps. To test the behavior of BlockMaps in a networked environment, we stored BlockMap on a remote server with a controlled bandwidth of 160 KB/sec; in these conditions we were able to stream approximately 20 BlockMaps per second. Requests to the server are done in the order in which BlockMaps are needed for drawing and, when needed BlockMaps nodes are missing, coarser nodes are used. In the video we show a networked browsing session of the Paris dataset represented using only BlockMaps; BlockMap transmission starts when the user start the browsing session, i.e., with an empty local cache. The number of transmitted BlockMaps is shown in the upper left corner of the screen. As demonstrated by the video, the ability of BlockMaps to have a predictable size and to offer good approximation even at coarse levels is particularly useful in this limited bandwidth network setting. The visual quality of the inspection is always acceptable and incrementally improves at a rapid pace.

6. Conclusions and future work

This paper introduced the BlockMap, a network- and GPU-friendly simplified representation of a portion of urban scenery that provides a replacement for the original color and geometry in the classical LOD meaning, characterized by efficiency in rendering, compactness in space, completeness of the representation and bounded geometric error. BlockMaps can be arranged in a simple hierarchical structure that can be seamlessly and efficiently integrated as coarse representations into existing multiresolution structures. Compactness of the BlockMaps structure make them especially useful in a networked visualization framework, where they provide a size and error bounded representation of urban scene that can be progressively downloaded and displayed.

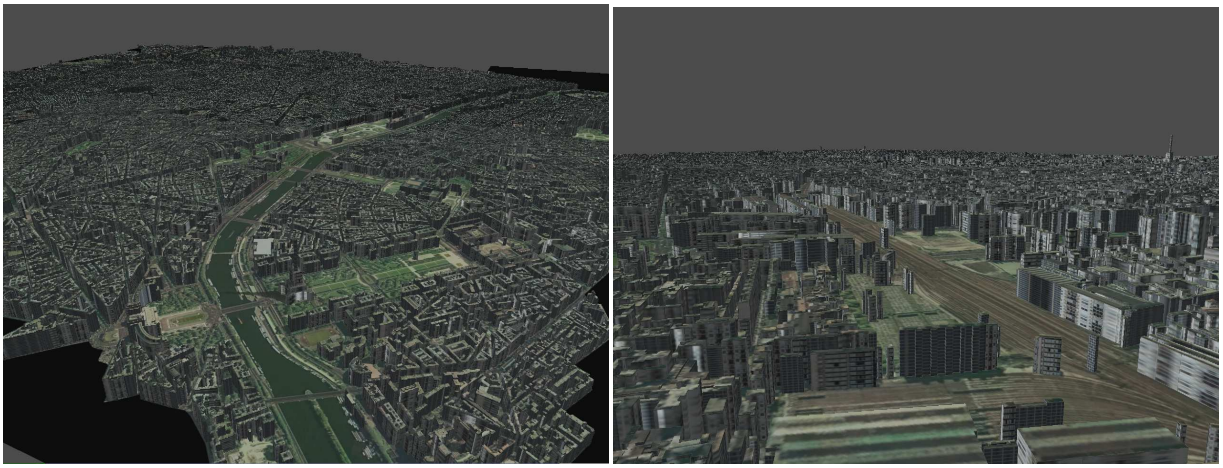

Figure 9: *Two snapshots from the flight over Paris video. The far away buildings are efficiently rendered using the BlockMaps approach.*

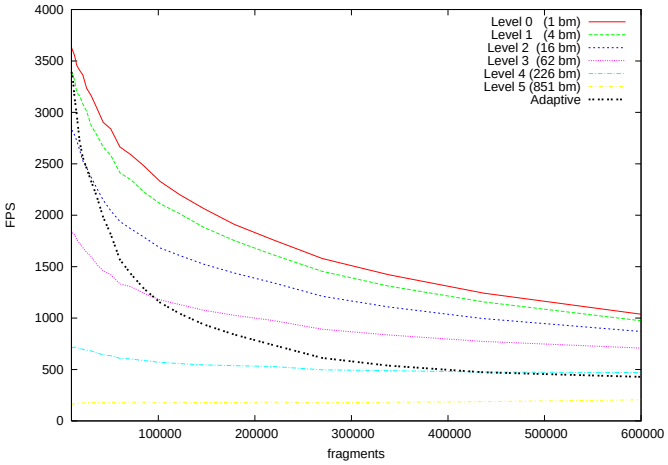

Figure 8: *Rendering rates of BlockMaps at different levels of accuracy*

References

[BD05] BUCHHOLZ H., DÖLLNER J.: View-dependent rendering of multiresolution texture-atlases. In *IEEE Visualization* (2005), p. 28.

[BGB*05] BORGEAT L., GODIN G., BLAIS F., MASSICOTTE P., LAHANIER C.: Gold: interactive display of huge colored and textured models. *ACM Trans. Graph. 24*, 3 (2005), 869–877.

[BWPP04] BITTNER J., WIMMER M., PIRINGER H., PURGATHOFER W.: Coherent hierarchical culling: Hardware occlusion queries made useful. *Comput. Graph. Forum 23*, 3 (2004), 615–624.

[CCV06] CORNELIS N., CORNELIS K., VAN GOOL L.: Fast compact city modeling for navigation pre-visualization. In *Proc. IEEE CVPR'06* (2006).

[CE98] CLINE D., EGBERT P. K.: Interactive display of very large textures. In *IEEE Visualization* (1998), pp. 343–350.

[Coo84] COOK R. L.: Shade trees. *Computer Graphics 18-3* (July 1984), 223–231.

[Don05] DONNELLY W.: *GPU Gems 2*. Addison-Wesley, 2005, ch. Per-Pixel Displacement Mapping with Distance Functions, pp. 123–136.

[DSSD99] DECORET X., SILLION F., SCHAUFLER G., DORSEY J.: Multi-layered impostors for accelerated rendering. *Computer Graphics Forum 18*, 3 (Sept. 1999), 61–73. ISSN 1067-7055.

[DSV97] DARSA L., SILVA B. C., VARSHNEY A.: Navigating static environments using image-space simplification and morphing. In *SI3D* (1997), pp. 25–34, 182.

[FJZ05] FRUEH C., JAIN S., ZAKHOR A.: Data processing algorithms for generating textured 3d building facade meshes from laser scans and camera images. *International Journal of Computer Vision 61*, 2 (feb 2005), 159–184.

[GGH02] GU X., GORTLER S. J., HOPPE H.: Geometry images. In *SIGGRAPH 2002 Conference Proceedings* (2002), Hughes J., (Ed.), Annual Conference Series, ACM Press/ACM SIGGRAPH, pp. 335–361.

[JW02] JESCHKE S., WIMMER M.: Textured depth meshes for realtime rendering of arbitrary scenes. In *Proceedings of the 13th Eurographics Workshop on Rendering (RENDERING TECHNIQUES-02)* (Aire-la-Ville, Switzerland, June 26–28 2002), Gibson S., Debevec P., (Eds.), Eurographics Association, pp. 181–190.

[JWS02] JESCHKE S., WIMMER M., SCHUMANN H.: Layered environment-map impostors for arbitrary scenes. In *Graphics Interface* (2002), pp. 1–8.

[KMS*06] KASIK D., MANOCHA D., STEPHENS A., BRUDERLIN B., SLUSALLEK P., GOBBETTI E., CORREA W., , QUILEZ I.: Real time interactive massive model visualization. In *Eurographics 2006: Tutorials* (2006).

[LRC*02] LUEBKE D., REDDY M., COHEN J., VARSHNEY A.,

WATSON B., HUEBNER R.: Advanced issues in level of detail. In *Course 14, SIGGRAPH 2002.* (July 21-26 2002).

[MS95] MACIEL P. W. C., SHIRLEY P.: Visual navigation of large environments using textured clusters. In *SI3D* (1995), pp. 95–102, 211.

[OBM00] OLIVEIRA M. M., BISHOP G., MCALLISTER D.: Relief texture mapping. In *Proceedings of the Computer Graphics Conference 2000 (SIGGRAPH-00)* (New York, July 23–28 2000), Hoffmeyer S., (Ed.), ACMPress, pp. 359–368.

[SDB97] SILLION F., DRETTAKIS G., BODELET B.: Efficient impostor manipulationfor real-time visualization of urban scenery. *Computer Graphics Forum 16*, 3 (Aug. 1997), 207–218. Proceedings of Eurographics '97. ISSN 1067-7055.

[SGwHS98] SHADE J., GORTLER S. J., WEI HE L., SZELISKI R.: Layered depth images. In *SIGGRAPH* (1998), pp. 231–242.

[ST99] SCHMALSTIEG D., TOBLER R. F.: Fast projected area computation for three-dimensional bounding boxes. *Journal of Graphics Tools: JGT 4*, 2 (1999), 37–43.

[TAB*03] TELLER S., ANTONE M., BODNAR Z., BOSSE M., COORG S., JETHWA M., , MASTER N.: Calibrated, registered images of an extended urban area. *International Journal of Computer Vision 53*, 1 (June 2003), 93–107.

[TMJ98] TANNER C. C., MIGDAL C. J., JONES M. T.: The clipmap: A virtual mipmap. In *SIGGRAPH* (1998), pp. 151–158.

[WTL*04] WANG X., TONG X., LIN S., HU S., GUO B., SHUM H.-Y.: Generalized displacement maps. In *Proceedings of the 2004 Eurographics Symposium on Rendering* (June 2004), Fellner D., Spencer S., (Eds.), Eurographics Association, pp. 227–234.

[WWS01] WIMMER M., WONKA P., SILLION F.: Point-based impostors for real-time visualization, May 29 2001.

[WWT*03] WANG L., WANG X., TONG X., LIN S., HU S.-M., GUO B., SHUM H.-Y.: View-dependent displacement mapping. *ACM Trans. Graph. 22*, 3 (2003), 334–339.