Relief Impostor Selection for Large Scale Urban Rendering

Carlos Andujar[*] José Díaz[†] Pere Brunet[‡]

Modeling, Visualization, Interaction and Virtual Reality Group
Universitat Politècnica de Catalunya

Abstract

Image-based rendering techniques are often the preferred choice to accelerate the exploration of massive outdoor models and complex human-made structures. In the last few years, relief mapping has been shown to be extremely useful as a compact representation of highly-detailed 3D models. In this paper we describe a rendering system for interactive, high-quality visualization of large scale urban models through a hierarchical collection of properly-oriented relief-mapped polygons. At the heart of our approach is a visibility-aware algorithm for the selection of the set of viewing planes supporting the relief maps. Our selection algorithm optimizes both the sampling density and the coverage of the relief maps and its running time is mostly independent on the underlying geometry. We show that our approach is suitable for navigating through large scale urban models at interactive rates while preserving both geometric and appearance details.

1 Introduction

Real-time visualization of large urban models is a challenging problem requiring a careful handling of their intrinsic multiple levels of scale. Image-based rendering techniques offer a convenient solution to accelerate the rendering of massive outdoor models. These techniques typically outperform textured-mapped simplified meshes for replacing distant geometry, as traditional simplification techniques typically perform poorly on dense collections of small connected components.

Relief mapping techniques make use of per-texel depth values to provide parallax effects and faithfully reproduce details on silhouettes. Unfortunately, most relief mapping approaches suffer from undersampling problems in regions with multi-valued projection. This problem is particularly noticeable in urban model visualization due to the large number of self-occlusions between human-made structures. Omni-directional relief impostors (ORIs) [1] overcome these limitations by representing detailed 3D objects through a small set of properly-oriented relief mapped polygons. Unlike other approaches, each relief map provides a global view of the whole model from a particular direction (see Figure 1-a). A small subset of the selected maps (typically three) are rendered based on a ray-height-field intersection algorithm. A major issue with ORIs is the selection of the set of viewing planes supporting the relief maps. The goal here is to maximize the sampling quality and coverage while minimizing the number of directions.

The construction algorithm proposed in [1] starts with 20 regularly spaced directions and optimize this initial set of views by searching for the neighboring view that maximizes the entropy of relative projected areas of the visible polygons. In this case, it was considered that *n*=20 relief maps was a good compromise between memory requirements and sampling quality for representing single objects.

[*]e-mail: andujar@lsi.upc.edu
[†]e-mail:jose.diaz.iriberri@gmail.com
[‡]e-mail:pere@lsi.upc.edu

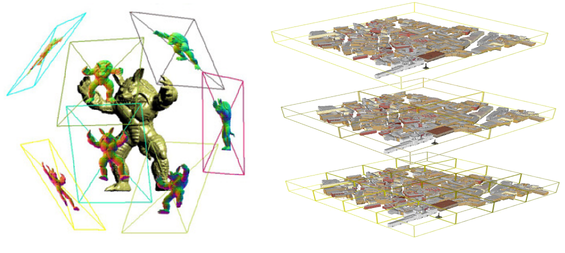

(a) (b)

Figure 1: A set of relief maps defining an omni-directional relief impostor *(a)*. Three levels of the quadtree hierarchy *(b)*

In this paper we propose a radically different approach for selecting an optimized set of view directions particularly designed for large urban models. Our algorithm optimizes both the number of directions and the directions themselves while achieving a good trade-off between sampling density and sampling coverage. Our optimization algorithm only requires as input a depth map taken from an orthogonal camera aligned with the vertical direction. Unlike [1], our optimization phase is based solely on simple 2D image processing operations and does not require to render repeatedly the original model to evaluate the sampling quality, thus enabling searching on a larger space with fast preprocessing times.

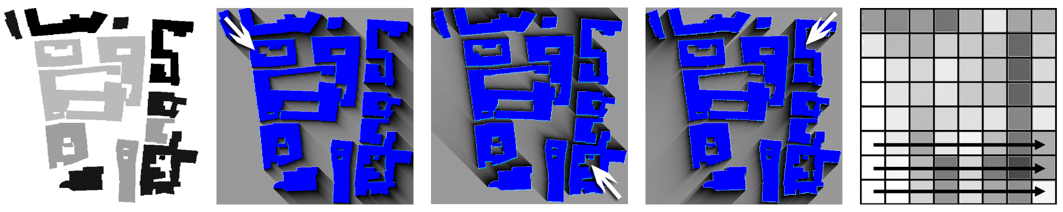
The main contributions of the paper are:

- A visibility-aware algorithm for selecting a small subset of relief maps which optimizes both the sampling density and sampling coverage on urban models.
- A GPU-based rendering strategy to seamlessly render overlapping relief maps which guarantees that all texels in a relief map can contribute to the final output image.
- A rendering system for interactive visualization of large urban models through a quadtree hierarchy of relief impostors.

2 Related work

In this section we review only previous work most closely related to our approach. Interested readers may refer to recent surveys on massive model visualization [5] for further information.

Impostors that replace distant geometry can be used for extreme appearance-preserving simplification [6]. A complete overview of the state-of-the-art is given in [4]. Several works based on relief texture mapping try to represent fine detail geometry or complex objects by using depth textures. Policarpo *et al.* [8] exploit the programmability of modern GPUs to implement a pixel-driven solution to relief texture mapping. All the necessary information for adding surface details to polygonal surfaces is stored in RGBA textures. The RGB channels encode a normal or color map, while its alpha channel stores quantized depth information. The technique uses an inverse formulation based on a ray-heightfield intersection algorithm implemented on the GPU.



(a) (b) (c) (d) (e)

Figure 2: Relief map selection: *(a)* input depth map, *(b-d)* θ_m-maps corresponding to the three best directions found by the selection algorithm, *(e)* scanlines used for computing the θ_m-maps for $\psi = 0$.

A major problem in the above image-based techniques is how to guarantee sufficient sampling so as to minimize disocclusion artifacts. For example, a relief map constructed from a vertical view direction might fully capture the geometry of a urban model (assuming it appears as 2.5D geometry when viewed at a distance) but would completely miss facade details. Some attempts have been proposed to overcome this problem [7, 2]. The original relief mapping approach is extended in [7] to render non-height-field surface details in real time. This new technique stores multiple depth and color values per texel. Unfortunately, faithfully representing building blocks with high depth complexity would require storing a prohibitive number of (z,color) pairs. Baboud and Décoret [2] build the height fields using inverse perspective frusta, which help improve the sampling on certain regions but fail to capture concave parts.

The BlockMap representation [3] encodes both the geometry and the sampled appearance of a small group of simple buildings, represented by a collection of textured vertical prisms. Although the BlockMap representation is less redundant than ours, our approach does not suffer from artifacts when reproducing non-orthogonal surfaces like tilt roofs and provides a more general solution for a larger class of building shapes.

3 Overview

Omni-directional relief impostors can be used in a hierarchical way, using a space-subdivision scheme, for the interactive inspection of complex scenes (see Figure 1-b). Although we have adopted a simple quadtree-based representation inspired by [1], relief impostors can be seamlessly integrated into streaming-friendly, multi-threaded architectures for interactive rendering of complex urban models [3].

Unlike [1], we build the relief maps for each submodel in a bottom-up fashion, using the relief maps of the child nodes to obtain the relief maps for the parent node. This strategy dramatically accelerates preprocessing times and provides a fully scalable construction since the whole city model is not required to fit in memory at any time. Relief maps for the leaves of the quadtree hierarchy are generated using the view selection algorithm described in Section 5. Once an optimal view set has been computed, the model is rendered using orthogonal cameras aligned with the chosen view directions, and we grab the depth and color maps. We use six clipping planes that bound the corresponding box to ensure that only interior geometry is captured by the impostors.

Our relief map selection algorithm always ensures that one of the relief maps (called *zenithal map*) is built using an orthogonal projection onto a horizontal plane (see Figure 2-a). This relief map fully captures the geometry of the building blocks, assuming rough 2.5D geometry. The rendering algorithm detailed in next section guarantees that the geometric error (in the Hausdorff distance sense) is bounded by half of the length of the diagonal of a zenithal map texel in object space, and the geometric error in screen space is the projection of such length. The view-dependent navigation of the quadtree is based on these projected error values. In a pre-order traversal, a given node is rendered (by rendering a subset of their relief maps) if its screen-projected error is below a user-defined tolerance. If the projected error is above the tolerance value, we recursively descend the quadtree repeating the above test.

4 Rendering the quadtree nodes

Each quadtree node stores a collection of up to eight properly-aligned relief maps (see next section) plus a zenithal map. Fortunately, we do not have to render all eight precomputed relief maps, but only those contributing to the current view. In a typical scenario only the selected subset will reside in GPU memory whereas the others will be requested from a remote server as needed. Traditional pre-fetching and streaming techniques [5] can be adopted for smooth animation.

Therefore the first step is to select a subset of the precomputed relief maps to be rendered. We force this subset to always include the zenithal relief map, as it fully captures the geometry of the buildings and provides the best sampling for horizontal surfaces like roofs. The rest of the selected impostors would basically contribute to add detail to building facades and other non-horizontal surfaces poorly captured by the zenithal map.

The view-dependent selection proposed in [1] is based on discretizing the Gauss sphere representing all possible view directions and finding the best subset for each view v using a brute-force approach which compares the render of the original model from v with that of the subset being tested. For comparing the two renderings, the authors measure the coverage error defined as the number of pixels covered by the original model that are missing in the render of the selected relief maps.

In our urban rendering system we have adopted a completely different strategy. Since the zenithal relief map is always rendered, the coverage error is approximately zero. Therefore we use a much simpler and faster selection strategy: we select the two relief maps whose direction is better aligned with the current camera direction, in addition to the zenithal map. Once the relief maps to render have been chosen, the support polygon is in turn sent to the graphics pipeline, where the actual relief mapping of the impostors will be computed by the fragment shader. The shader requires computing the intersection of the fragment's viewing ray with the height field encoded by the depth map. For this particular problem we have adopted the simple algorithm described in [9].

Since each relief map provides a global view of the underlying geometry, we will have pixels covered by more than one relief map, and we need to be able to decide which one will contribute its color to the pixel. We compute the correct depth value in the fragment shader, so that correct visibility and interpenetration with other scene objects are guaranteed. In case of approximate tie, which is very likely to happen in overlapping parts, we prioritize the relief map whose direction is better aligned with the view direction, giving to the zenithal map the lowest priority. This priority is implemented as a simple offset of the depth value computed in the fragment shader.

A key issue is how to compute the rectangular portion of the plane to be drawn. This is important because this determines which fragments will be processed and hence which viewing rays will be checked for intersection with the model. We project the vertices of the bounding box of the model (stored in each quadtree node) into the plane (in the direction of the viewpoint). Using textured coordinates outside the range $[0,1]$ enables the ray intersection code to quickly jump to the portion of the plane where texture information is available. This strategy guarantees that every frontfacing relief map will be able to contribute to the final image, even those at grazing angles, and ensures seamless reconstructions between adjacent quadtree cells.

5 Relief map selection

We now describe the algorithm to select the set of directions that will be used to build the relief maps. The problem can be stated as follows: given the part of the urban model overlapping a quadtree cell, decide the number n of required relief maps and compute the n directions to use in the construction of them.

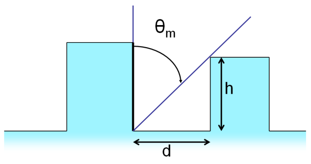


Figure 3: Basic components for computing the sampling quality of a facade along a given view direction.

Recall that we always build one of the relief maps by projecting the model onto a horizontal plane (the zenithal map). The selection algorithm will have to compute additional view directions which will be used to build the *complementary relief maps*. This step is critical to ensure proper sampling of the appearance attributes of model parts like building facades. Note that the depth channel of the zenithal map provides accurate sampling of the geometry whereas its color channels provide excellent sampling of the color of roughly horizontal surfaces. Therefore complementary maps will basically add color detail to those parts poorly sampled by the zenithal map.

Our algorithm choses n view directions from a hemisphere around the model. Each direction is represented by a pair of angles (θ,ψ) where $\theta \in (0,90)$ represents the elevation measured from the vertical axis (i.e. $\theta = 0$ corresponds to the zenithal map) and $\psi \in [0,360)$ represents the direction measured from the X axis.

The goal here is to maximize the color sampling quality and coverage while minimizing the number of directions. By sampling coverage we refer to the presence of a particular model feature in the resulting relief map, regardless of the number of texels used for sampling it. Sampling coverage from a particular view direction is severely affected by self-occlusions, which is particularly important in urban models. Therefore sampling coverage optimization algorithm must be visibility-aware. A related concept is color sampling density, which refers to the ratio between the area of a surface part and the corresponding area in texture space. Besides texture resolution, sampling density depends on how well the relief map direction is aligned with the surface normal being captured.

Note that in a typical urban environment with multiple occlusions, maximizing sampling coverage and sampling density are often opposite goals. Consider several possible elevation values for building a relief map, ranging from a nearly zenithal direction ($\theta \approx 0$) to a horizontal direction ($\theta \approx 90$) perfectly aligned with the main facades of a building. A nearly zenithal direction would provide greater coverage as bird's eye views are less subject to occlusion effects, but the sampling density will decrease proportionally with the sinus of the elevation angle θ.

The direction achieving a good compromise between sampling density and coverage depends basically on the depth complexity of the model in hand (how densely occluded are the buildings). Examples of two extreme cases are rural areas with low-height buildings distributed sparsely and downtown areas populated with skyscrapers. The first kind of scene is mostly unoccluded, so nearly horizontal views would provide both excellent sampling density and coverage. Analogously, the second kind of scene is densely occluded and hence nearly horizontal views would provide very poor coverage of second-line buildings and hence it would suffer from disocclusion events.

Suppose we fix a given direction angle ψ and want to find an elevation angle θ achieving a good trade-off between sampling quality and sampling coverage. For this particular direction and for a given facade roughly aligned with ψ, we can compute the maximum angle θ_m for which the whole facade is captured by a relief map oriented along (θ,ψ) (see Figure 3). As we move the elevation angle θ from 0 to θ_m, the sampling density increases whereas the coverage is preserved, as no disocclusion events occur. Therefore for this particular value of ψ taking values $\theta < \theta_m$ is clearly suboptimal.

When moving θ from θ_m to 90, the sampling density still increases but the sampling coverage decreases due to the partial occlusion of the facade. Therefore we must select a value for θ achieving a good balance between density and coverage. Regarding the selection of ψ values, we should also optimize the sampling quality by avoiding unstable views so as to minimize artifacts due to disocclusion events [1].

The input of our selection algorithm is the depth map corresponding to the zenithal direction (see Figure 2-a). This depth map contains all the geometric information and thus is enough to evaluate the sampling density and coverage of any view direction. In the following we consider inverted $h = 1 - z$ values so that they correspond to heights above the horizontal plane. Our algorithm only produces a set of directions where all (θ,ψ) pairs have distinct values on ψ (there is at most one elevation for each selected direction). Multiple θ values for the same direction ψ makes sense on arbitrarily shaped objects but not on urban models with rough 2.5D structure.

Our selection algorithm proceeds through two main steps. We first build a θ_m-map for a fixed set of probe directions which define the search space for the optimization of ψ values (in the examples we used 24 directions, leading to 15 degree steps). The θ_m-map contains the value of θ_m for each point not belonging to a building (i.e. for image points with h below some threshold). Figure 2 shows the θ_m-maps for three directions (ψ =315, 135 and 225 deg).

The θ_m-maps can be computed very efficiently using a simple algorithm. The algorithm visits the texels of the input depth map using a collection of scanlines aligned according to ψ (see Figure 2-e). As we proceed through a scanline, we keep an initially empty stack of (h,x) pairs for each building of height h at distance x encountered during the scan (the stored distance x is measured from the scanline origin). Every time we encounter a pixel corresponding to a building at (h,x), we pop from the stack all the buildings with height below h (these wont contribute anymore to the current scanline) and we push (h,x) into the stack. By construction the stack is always sorted by h and x (with the smallest value of h and greatest value of x in the top of the stack).

Pixels not belonging to a building ($h < \varepsilon$) will be referred to *street pixels*. Every time we encounter a street pixel we compute its θ_m value using the stack. For each building in the stack we compute a θ_m value as $\arctan(d/h)$ where d is the distance from the current texel to the building (see Figure 3). The minimum θ_m is stored in the θ_m-map, and we pop buildings from the stack until the building providing the minimum θ_m occupies the top. The computation of the 24 θ_m-maps for a 128x128 depth map takes less than 100ms.

The second step of the algorithm uses the 24 θ_m-maps to select n

view directions. We use a greedy algorithm combined with a voting scheme. A *facade pixel* is a pixel belonging to a building but immediately following a street pixel along the scanline. Each facade pixel votes for all the (θ, ψ) directions from which it is visible. The vote is weighted by the height of the building corresponding to the facade pixel. The set of θ values are discretized into 10-degree bins, resulting in 24x10 bins. Each vote is divided by the total facade area so that votes accumulated on a bin (θ, ψ) can be seen as the fraction of facade area visible from a relief map oriented along (θ, ψ). The best view is selected by choosing the pair maximizing the benefit function $sin(\theta)V(\theta, \psi)$ where $V(\theta, \psi)$ is the fraction of facade surfaces visible from (θ, ψ). Experimentally we have found this function to give a good trade-off between sampling density and sampling coverage. Once the best view is selected, we repeat the voting process but keeping facade pixels captured by previously computed views from voting. The process is repeated until we reach a user-defined coverage (e.g. 80% of the facade surfaces) or the number of computed views reaches a user-defined limit (we use up to 8 views per quadtree node). Figure 2 shows the three best directions found by the selection algorithm. Note that our algorithm naturally produces stable, well distributed views.

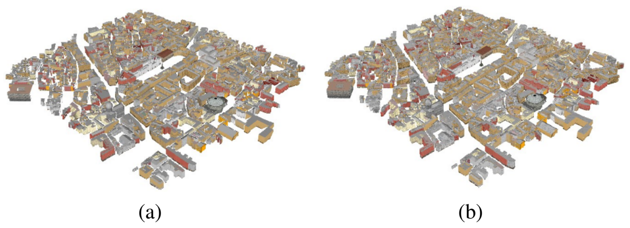


Figure 4: Results using the closest impostor (*a*) and the two closest impostors (*b*).

6 Results

We have developed a prototype implementation of the described system and tested it with a 2GB textured model of Rome covering around 36 km^2. Reported results have been measured on a dual-core CPU @2.13 GHz and a GeForce 7950GT graphics card equipped with 512 MB.

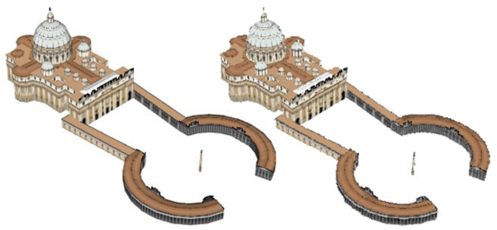


Figure 5: Original building and the result using two relief maps.

For the city model we use a five-level quadtree whose leaf nodes subdivide the model into 16x16 squares, resulting in 64 terminal nodes and 21 non-terminal nodes. Construction time is roughly 4 hours, including the time required to load a partial VRML model for each terminal quadtree node. Selection of view directions and construction of the relief maps accounts for a small fraction of this time, around 6 min. Table 1 shows the results with a part of the city model. In order to enable the comparison with the original model, we only show results with the largest city part fitting in GPU memory (covering 64 Ha of Rome) and the corresponding two-level subtree (see Figure 4). We include results using one, two or three relief maps per quadtree node. In all cases we use a 512x512 viewport. Note that rendering times are greatly decreased and allow for real-time exploration. Note also that the resulting images exhibit very few artifacts and the image quality is comparable to that of the original model.

Table 1: Comparison between rendering the original model and our approach.

	Original	Relief maps		
# texture maps	2741	4	8	12
Texture Memory (MB)	10.7	0.25	0.5	0.75
FPS	3	370	260	150

One of the benefits of our approach is that most texels of the complimentary relief maps contribute to the final image, and hence the texture resolution is a good measure of the sampling density. Note that approaches using parameterizations of the facade walls [3] build color maps whose sampling density depends on the perimeter of the buildings. As a consequence, quadtree cells covering large areas populated with many buildings either require huge textures or color information must be dramatically subsampled.

7 Conclusions

We have presented a urban rendering system for high-quality visualization of large scale urban models through a hierarchical collection of properly-oriented relief-mapped polygons. A new algorithm for the selection of the set of viewing planes supporting the relief maps has been presented. The algorithm achieves a good balance between the sampling density and sampling quality, and it is fast enough to be applied to each quadtree node. We have shown that our approach is suitable for navigating through large scale urban models at interactive rates.

Acknowledgements

This work has been partially funded by GeoVirtual and the Spanish Ministry of Science and Technology under grant TIN2004-08065-C02-01. The Rome city model has been provided by Tele Atlas.

References

[1] C. Andujar, J. Boo, P. Brunet, M. Fairen, I. Navazo, P. Vazquez, and A. Vinacua. Omni-directional relief impostors. *Computer Graphics Forum*, 26(3):553–560, 2007.

[2] L. Baboud and X. Décoret. Rendering geometry with relief textures. In *Proc. of Graphics Interface*, pages 195–201, Toronto, Canada, 2006.

[3] P. Cignoni, D. B. M, F. Ganovelli, E. Gobbetti, F. Marton, and R. Scopigno. Ray-casted blockmaps for large urban models visualization. *Computer Graphics Forum*, 26(3):405–413, 2007.

[4] S. Jeschke, M. Wimmer, and W. Purgathofer. Image-base representations for accelerated rendering of complex scenes. In *STAR reports, Eurographics 2005*, pages 1–20, 2005.

[5] D. Kasik, D. Manocha, A. Stephens, B. Bruderlin, P. Slusallek, E. Gobbetti, W. Correa, and I. Quilez. Real time interactive massive model visualization. In *Eurographics 2006 Tutorials*, 2006.

[6] P. W. C. Maciel and P. Shirley. Visual navigation of large environments using textured clusters. In *SI3D '95: Proceedings of the 1995 symposium on Interactive 3D graphics*, 1995.

[7] F. Policarpo and M. M. Oliveira. Relief mapping of non-height-field surface details. In *Proc. of ACM Symp. on Interactive 3D Graphics and Games*, pages 55–62. ACM Press, 2006.

[8] F. Policarpo, M. M. Oliveira, and J. Comba. Real-time relief mapping on arbitrary polygonal surfaces. In *Proc. of ACM Symposium on Interactive 3D Graphics and Games*, pages 155–162. ACM Press, 2005.

[9] F. Policarpo, M. M. Oliveira, and J. Comba. Real-time relief mapping on arbitrary polygonal surfaces. In *Proc. of ACM Symposium on Interactive 3D Graphics and Games*, pages 155–162. ACM Press, 2005.