EUROGRAPHICS 2007 / D. Cohen-Or and P. Slavík
(Guest Editors)

Volume 26 (2007), Number 3

Omni-directional Relief Impostors

C. Andújar, J. Boo, P. Brunet, M. Fairén, I. Navazo, P. Vázquez, À. Vinacua

Group MOVING†, Software Department, Universitat Politècnica de Catalunya, Barcelona, Spain

Abstract

Relief impostors have been proposed as a compact and high-quality representation for high-frequency detail in 3D models. In this paper we propose an algorithm to represent a complex object through the combination of a reduced set of relief maps. These relief maps can be rendered with very few artifacts and no apparent deformation from any view direction. We present an efficient algorithm to optimize the set of viewing planes supporting the relief maps, and an image-space metric to select a sufficient subset of relief maps for each view direction. Selected maps (typically three) are rendered based on the well-known ray-height-field intersection algorithm implemented on the GPU. We discuss several strategies to merge overlapping relief maps while minimizing sampling artifacts and to reduce extra texture requirements. We show that our representation can maintain the geometry and the silhouette of a large class of complex shapes with no limit in the viewing direction. Since the rendering cost is output sensitive, our representation can be used to build a hierarchical model of a 3D scene.

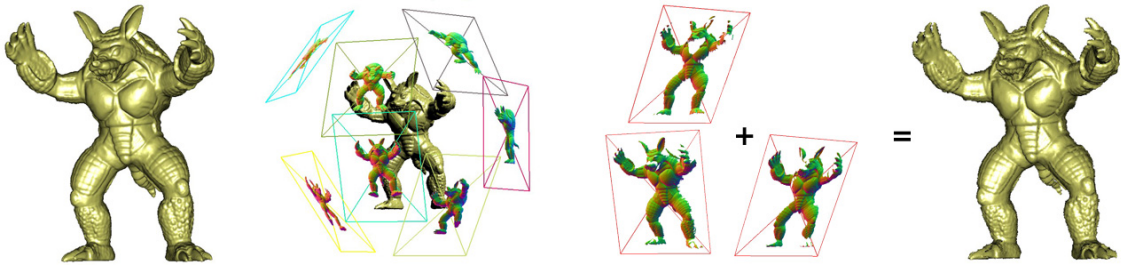


Figure 1: Several relief maps are combined to provide an impostor set that can be rendered from arbitrary directions.

1. Introduction

Real-time 3D exploration of complex geometric models is an important problem in many practical applications. The amount of geometric information is often too high, and simplified representations must be used for distant objects. A huge variety of such representations has been proposed in the last few years, including simplified triangle meshes, textured meshes, impostor clouds, and relief textures.

These representations present, however, a number of limitations. Simplified triangle meshes and textured meshes cannot handle parallax effects and do not preserve high frequency detail on the silhouettes. Impostors can be a good option for distant objects, but they are only valid for a small cone of viewing directions. Deformation artifacts and holes can occasionally appear. The cone of acceptable directions also applies to relief impostors, although in this case, the cone is larger. But artifacts can be produced because of a lack of information in zones with multi-valued projection. Relief impostors can be mapped onto a simplified mesh, but a large number of impostors is required in this case. Con-

† This work partially supported by grant TIN2004-08065-C01 of the Spanish Government

versely, small impostor sets are not valid for arbitrary viewing directions. On the other hand, algorithms based on the warping of six orthogonal textures (or relief textures) cannot avoid artifacts in general concave objects.

The omni-directional relief impostors (ORI) proposed in this paper overcome these problems by combining a number of relief impostors into a representation that preserves high frequency details and supports parallax effects and advanced per fragment lighting effects. Compared to previous approaches, our ORIs include a sufficient number of properly oriented relief impostors that can be rendered for any view direction with few artifacts. An image-space threshold measure, which is computed in the preprocessing step, can be used to ensure high quality renderings during interactive navigations.

Contributions: This paper gives new algorithms for the construction, selection and rendering of a compact view-dependent representation for the interactive exploration of complex models. Our main contributions include,

- An efficient algorithm to compute a sufficient number of properly oriented planes supporting the relief maps.
- An algorithm for the view-dependent selection of a minimal set of relief maps maximizing the visual quality of the final image, and a GPU-based rendering algorithm (see [vid]) that merges overlapping relief maps while minimizing sampling artifacts and reducing memory requirements.
- The use of a *Projected Length Threshold* that is computed during preprocessing and that measures the quality of the omni-directional relief impostors. This metric is used to discuss and compare different options regarding the construction and selection of the impostors.
- Being also able to represent sub-regions of complex models, the omni-directional relief impostors can be used in a hierarchical way, using a space-subdivision scheme, for the interactive inspection of complex scenes and very complex shapes.

The projected length threshold can be tested for every camera position to check if the screen size of the object allows its rendering with the ORI model. In case that its screen projection is larger than the threshold, we can still render the ORI with some visible artifacts, switch to a higher quality representation for close-ups, or use a spatial subdivision and render the ORI model of the smaller children (discussed in Section 5).

2. Related Work

Surface simplification literature has focused mainly on simplification algorithms tuned for highly tessellated, topologically simple surfaces, which are frequently modelled as triangular meshes (see [LRC*03] for a comparative survey). Cignoni et al. [CMR*99] propose a general approach for preserving detail on simplified meshes through color and bump maps. The algorithm receives as input both the original and the simplified mesh and computes a set of triangular textures which are then packed into a single map. In [TCS03] a new method for the creation of normal maps for covering the detail on simplified meshes is proposed, and a set of objective techniques to measure the quality of different recovering techniques. These simplification algorithms generate water-tight approximate models; unlike our present proposal, they need a relatively long pre-process construction, cannot handle parallax effects and do not preserve high frequency detail on the silhouettes.

Impostors that replace distant geometry can be used for extreme appearance-preserving simplification. A complete overview of the state-of-the-art can be seen in [JWP05]. They classify different techniques, of which we highlight here those using only one impostor [SDB97], layered depth images (LDI) [OB99, Wil02] and billboard clouds [DDSD03, ABC*04]. The first two techniques are only valid for restricted viewing directions. Billboard clouds are view-independent but need a computation time that is too long for online simplification and when few planes are used the appearance of the object is not faithfully captured and crack artifacts occur.

Several works based on relief texture mapping (texture that includes a depth map) try to represent fine detail geometry or complex objects by using textures [OBM00]. Policarpo et al. [POC05] 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 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. For each ray, the intersection is found by a linear search followed by a binary search along the ray. This algorithm has been extended in [PO06] to render non-height-field surface details in real time. This new technique stores multiple depth and normal values per texel and generalizes their previous proposal for rendering height-field surfaces. Moreover, it allows to add details to the underlying object's silhouettes. These approaches can miss the first intersection point when the ray intersects several times with the height-field.

Baboud and Décoret [BD06] obtain more accurate intersections than the linear search of [POC05] by storing in a texture a safety radius, this way larger steps can be made, although it is slower. Moreover, the height fields are built using inverse perspective frustra, which help improve the sampling on certain regions.

All these proposals can be used to render realistic impostors of 3D objects with fast preprocess computation. [PO06] shows that by using a single impostor they can obtain a relatively wide angular range, even though some deformation can be perceived in the final image when the angle is wide

enough. [BD06] uses 6 impostors mapped on the bounding box. All of them still present some problems of artifacts for general complex objects and only permit a limited number of viewpoint directions.

Relief impostors representing complex objects have been explored in [VK06]. Their construction algorithm ensures that all poligons are captured by at least one impostor. The resulting representation is less redundant than ours, but artifacts at junctions are much more noticeable and there is no upper bound to the number of planes required to capture all the polygons.

In this paper we do not focus on representing mesostructure details in objects —although it is supported— but on representing the geometry of general complex objects. Our algorithm is based on [POC05] for the ray-heightfield intersection and we combine a sufficient number of properly oriented relief impostors (those having maximal contribution to the visual quality) allowing a hierarchical use of them to produce an almost free of artifacts representation at interactive rates.

3. Omni-directional Relief Impostors

In order to overcome the restrictions in the viewing direction of relief maps, it seems natural to attempt to combine several such maps with different orientations. In order to do so, one must be able to answer several questions: for a given object, which are the best relief impostors to compute? For a given set of relief impostors and a viewing direction, which of the impostors should be used? And for each fragment in the overlap region of two chosen impostors, how can we decide which one is more appropriate? Since the impostors are computed at a certain resolution, it is evident that they will present artifacts and visible defects if we zoom-in enough. Another question we need an answer for is then: what is the threshold beyond which we cannot use the set of impostors? The following subsections discuss our approach to each of these questions.

3.1. Construction

The first aspect of the problem that we address can be stated as follows: given a geometrical model of an object, and a desired number n of impostors, find the n directions to use in the construction of the depth maps. The value chosen for n affects the storage and the pre-processing of the model, but not its rendering, since impostors will be used only as needed. We've found 20 to be a good compromise in our tests.

Lacking a formal setting in which the relative benefits of one choice against another can be assessed, we experimented with different alternatives, and compared the results. The strategies that we evaluated were:

a **Regularly spaced:** the n directions are sampled in a regular fashion around the object.
This approach has the advantage of simplicity. But clearly it might construct impostors of little relevance. Most objects have some dull sides that are not very interesting compared to others (for example the base of a sculpture), and these will be sampled as carefully as any other in this approach. Since we are using $n = 20$, we have tested this strategy using as viewing directions the position vectors of the centroids of the faces of a fixed icosahedron centered at the origin. This yields normals spaced in such a way that the maximum deviation from an arbitrary direction to the closest one available in the sample is approximately 37 degrees.

b **Guaranteed minimal sampling:** the n directions are chosen by measuring the information content of the image of the object seen from that direction, and using the directions supplying more information. This can be done in several ways. In our case, we tested the information theory-based strategy by Vázquez *et al.* which has been used for Image-Based Modeling [VFSH03].
This approach attempts to address the shortcomings just discussed for the regular sampling. The problem is that since we are operating with a fixed n, for intricate objects some directions may turn out to be not sampled at all, undermining the objective of omnidirectional applicability of the impostor set.

c **Information-based perturbation:** finds the directions near those used in strategy *a)* above that maximize the information content and the stability of the image (meaning that small perturbations in the viewing direction tend not to introduce new faces of the object into the image, a desirable property because faces perpendicular to the impostor are not handled by the relief mapping). For each initial view, we search in a neighborhood for the viewpoint that yields the highest amount of information content, using the *viewpoint entropy* measure described in [VFSH03] due to its simplicity. Our search is performed for a set of views which have maximum distance of four degrees with the initial one.
This strategy compromises between the first two, since it uses the information content to adjust the viewing direction, but does not drift too far from the regular sampling in order to ensure the existence of impostors for all possible viewpoints.

In Figure 2, two views of the bunny are rendered, where each triangle has been colored according to the number of impostors that sample it. Triangles that are not sampled are black, and fragments that are seen by only one or two impostors are red. Lighter colors represent increasing number of impostors sampling that particular point, up to green fragments that correspond to 10 or more samples. The top row corresponds to strategy a) above (regularly spaced directions), whereas the bottom row shows the result of using strategy c). The figure clearly shows how this perturba-

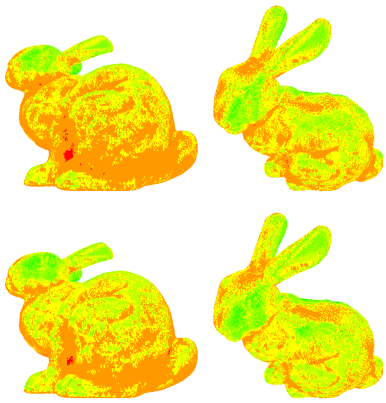


Figure 2: *Compared covering of the bunny from 2 different viewpoints, using regularly spaced impostors (top row) versus information-based impostors (bottom row). Lighter colors mean larger coverage.*

tion, while not expensive to compute, produces higher and more uniform coverages of the model. We therefore adopt this strategy in our implementation.

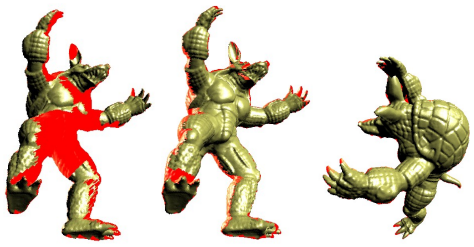
To support highly skewed views, the polygons of each relief impostor are extended to provide additional fragments and avoid cracks. These extents, however, need no texture, since the fragment shader skips all texture accesses until the ray intersects the real relief map.

3.2. Selection

The cost of rendering each relief impostor is not negligible, so we should not send them all to the rendering pipeline if fewer would suffice. It is obvious that the impostors can be culled by their directions, but it seems reasonable that fewer than half of the impostors will suffice in many cases. As in the construction of the impostors, we approach this issue through the experimentation of reasonable choices, and their evaluation on test cases. The selection mechanisms that we have evaluated are:

i m **closest:** Here the $m < n$ impostors for which the normal to the supporting plane is closest to the viewing direction are used. Although these provide the best samples for that direction, it is not always the case that the one with the second closest normal is the one that best complements the information of the impostor with the closest normal

ii $i \leq m$ **best:** The closest impostor is not necessarily the best. So in general taking the m closest uses impostors that spend rendering time and yield very few if any useful fragments. Instead, we precompute for each of 320 uniformly spaced directions (obtained by subdividing an icosahedron), the set of (up to m) best impostors to use. To do so, we run over all 320 directions, and for each experiment different combinations of impostors, comparing them with the result of rendering the original mesh in that direction. We set the goal of finding a combination with fewer than 0.2% wrong pixels (see section 3.4). We first try each of the m closest impostors to see if one suffices. If we do not reach the set goal, we try pairs of impostors from this set of m closest, and so on until finding a combination of impostors that meets the criterion. We store that selection and proceed to the next viewpoint. At rendering time, we use the selection of the closest direction sampled.



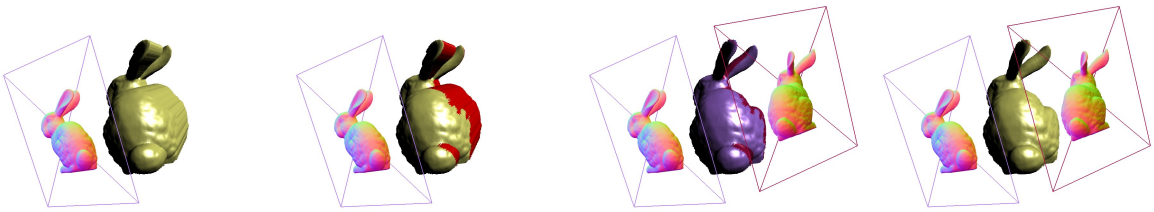
(a) (b) (c)

Figure 3: *Impact of selection criteria. For each of the two criteria used, the **worst** view of the model is rendered for one model. Red pixels are pixels not represented in the impostor chosen.*

In Figure 3 we show the advantages of option ii. The first two images represent the worst view using only one impostor, when the closest (Figure 3(a)), and when the *second*-closest impostor is chosen (Figure 3(b)). Notice that the worst case for the choice of the second closest is better than the worst case for the closest impostor (both impostors have directions about 30 degrees apart from the viewing direction). In contrast, when using the *best* impostor for each viewing direction, the worst possible view is that seen in Figure 3(c), which clearly misses much less. These choices are compared numerically in Section 4.

3.3. Rendering

Once the impostors to use have been chosen, the support polygon of each is in turn sent to the graphics pipeline, where the actual relief mapping of the impostors will be computed by the fragment shader. However, now we will have pixels covered by more than one impostor, and we need to be able to determine which is the correct one. Notice that simple z-ordering of fragments does not suffice to do the filtering, because relief maps generate solid artifacts *behind* the faces that comprise the image at the impostor. To illustrate this, Figure 4 shows a skew view of a relief map impostor for the bunny (where the map is represented by the rectangle, shifted for clarity). In Figure 4(a) the bunny's body and ears cast solid shadows from the direction of construction of the relief map. These are highlighted in red in Figure 4(b). [PO06] have suggested storing multilevel relief



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

Figure 4: *Artifacts due to skew viewing direction (a) are highlighted in (b). Notice that although some (the shadow of the body) will disappear when storing two depths in the relief-map, others (like the ears) will persist. Adding a relief map from a different direction helps correct the problem: (c) shows the contribution of each of the two impostors to the fragments of the rendered image (d)*

maps to address this. Storing two depths, in this case, would overcome the shadow of the body, but not that of the ears. Multilevel relief maps are further limited by the fact that these techniques can only render on the projection onto the viewing plane of the supporting polygon, hence even in the presence of multiple depths, the model will be squashed as it turns. Our method uses instead a different projection direction to construct a second relief map (Figures 4(c) and (d)) to provide both the missing information and the additional coverage of the viewing plane, therefore rendering a more accurate model without deformation under rotation.

In order to detect the added volumes shown in Figure 4(a), and in a manner similar to [OBM00], the shader detects sharp discontinuities in depth, when computing a ray's intersection with the model. Figure 5 shows how this is done.

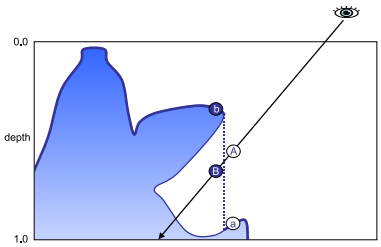


Figure 5: *Detection of an artifact volume in the shader*

When the shader traces the ray, it eventually finds two neighboring samples (labeled "A" and "B") that are above and below the surface of the object, respectively. If the difference between their stored depths (labeled "a" and "b") is larger than a threshold, the shader knows it has hit a vertical wall, and discards the fragment.

By using several relief maps at once, some pixels in the image will be covered by the support polygon of more than one map. Therefore we must have a mechanism to choose which will determine the color of that pixel. A number of possibilities suggest themselves; the results of the most relevant are depicted in Figure 6, where the first column displays the original bunny model (Figure 6(a)), and — schematically— the three impostors involved in the test (Figure 6(f)). In the remaining columns, the different strategies discussed below are tested. The first row shows the result, and the second row shows which impostor contributed to each fragment in the result. We have highlighted in purple ovals some of the artifacts originated by some of these strategies. The options we have evaluated are:

1. Use the z-value of the impostor plane. This is arbitrary and is bound to make the wrong decision often.
2. Use the fragment's real value of z, obtained by the shader in its computation of the ray-surface intersection. The drawback is that the impostor providing the color may not be the one that has best sampled that portion of the model. Furthermore, in the absence of occlusions, all fragments sampling that point have approximately the same depth, and therefore the outcome is arbitrary. This strategy is shown in Figures 6(b) and 6(g).
3. Prioritize the impostor that is closest to the viewing direction. This can be done efficiently by setting `glDepthRange(`z_i, z_i`)`, for the i-th impostor; for instance one can pick a number $0 < \mathit{offset} \leq 1$ and take $z_i = i\frac{\mathit{offset}}{20}$, for impostors ordered from closest to farthest. This method has the advantage of likely choosing the best sample available. This strategy is shown in Figures 6(c) and 6(h).
4. Use the impostor whose normal is closest to that of the model's surface at the computed intersection point.
   Since the actual value of z is not used, a pixel may take its color from a much farther (but better aligned) sample. This strategy is shown in Figures 6(d) and 6(i).
5. Use the correct value of z taken from the relief map, modified to prioritize the best samples. Given the value of z encoded in the relief map, which is a value in $[0,1]$ with a resolution of $1/256$, we compute $\mathit{bias} = 1/32$, and decrease z by $\mathit{bias} \cdot \cos(\mathit{impostorNormal}, \mathit{surfaceNormal})$.

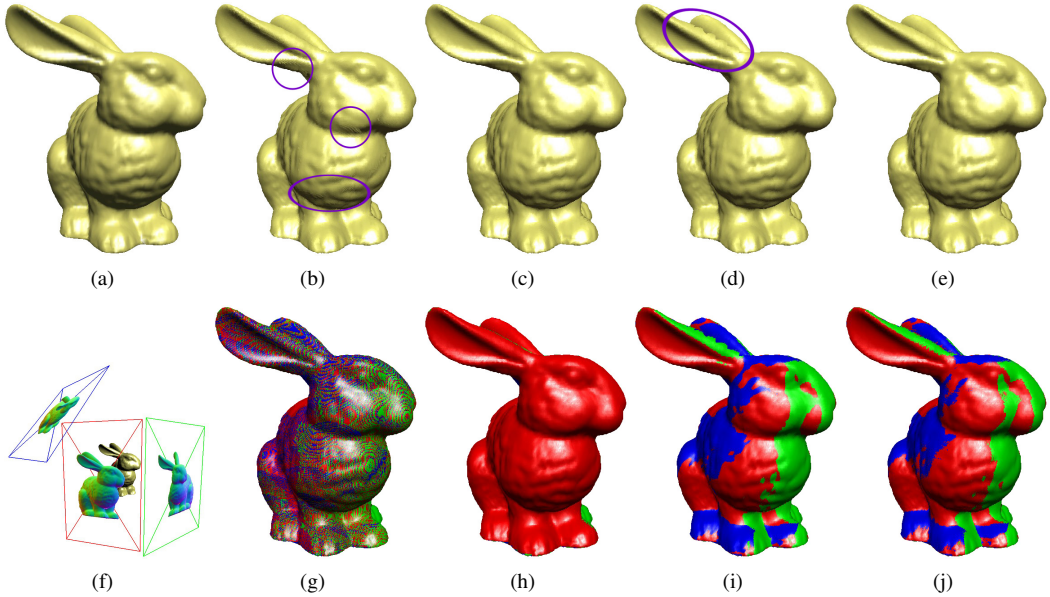


Figure 6: *A comparison of the diferent strategies to choose among redundant fragments*

This value of z converted to normalized device coordinates is stored in the fragment at the output of the shader. This strategy is shown in Figures 6(e) and 6(j).

These tests show that when viewing a single object, option 3 (Figures 6(c) and 6(h)) from the above is the most adequate. It is as fast as option 4, but has less artifacts. This is hence the strategy we have chosen for our implementation. When using our hierarchical approach introduced in Section 5 below, however, option 5 (Figures 6(e) and 6(j)) would be more advisable, because of its correct computation of the visibility (even between contributions from disjoint ORI's) coupled with the choice of good samples.

3.4. Computation of the Projected Length Threshold

The Projected Length Threshold (PLT) measures the maximum size of the projection of a certain object in display coordinates, such that its ORI rendering from any viewing direction can be considered perceptually similar to the rendered image of the initial object from the same viewing direction.

The PLT value of the ORI is based on the coverage error. The coverage error $d_i(s)$ for a viewport of $s \times s$ pixels and a viewing direction i is the error, in number of pixels, between the axonometric display of the ORI (using the algorithm in Section 3.3) and the axonometric display of the initial object. The coverage error is the number of pixels covered by the reference image and not covered by the ORI image. We do not consider the inverse covering (pixels covered by the ORI image and not covered by the reference image) because we have detected that it is not significant.

Let us first define the global coverage error $GC(s)$ that measures the maximum display error for arbitrary viewing directions, given the size s of the projection of the object in display coordinates. We approximate the projection size s is the edge size of the square bounding box of its screen projection in display coordinates. The global coverage error is now defined as

$$GC(s) = Max(d_i(s), i = 1 \ldots Nv)).$$

This maximum is computed over a predetermined set of Nv viewing directions.

In our experiments, we have used $Nv = 320$ viewing isotropic directions that we obtain from the subdivision of an icosahedron.

We use a first approximation for the projected length threshold, based on the value of the global coverage error. Once we have obtained the $GC(s_0)$ for a certain initial size s_0, we could iterate to find the maximum s_{max} such that $GC(s_{max})$ is lower than a prescribed tolerance. Anyway, this iteration is not necessary since it can be experimentally observed that $GC(s)$ is roughly proportional to the number $s \times s$ of pixels in the viewport. We seek a single-pixel threshold value such that in average there is only one covering error for each pixel row in the viewport. That is $GC(s_{max}) = s_{max}$, and hence $PLT = s_{max} = s_0^2/GC(s_0)$. The PLT value s_{max} is computed for every ORI in the preprocessing step. During the rendering, we can guarantee a good visual quality provided that the screen projection of the object is not larger than s_{max} pixels. A straightforward consequence is that the resolution of the impostor textures can be computed at this

step of the preprocess. A good choice is to use a texture size of the order of $2s_{max}$.

4. Results and Discussion

The ORI approach has been tested with different models on a P-IV3.2GHz, with an NVidia Gf-7950GT. Table 1 presents a comparison of results for the armadillo model, using the m closest impostors ($m = 1 \ldots 4$) and the $i < 6$-*best* criterion. The average coverage error decreases from 1.18% to 0.17% while the *PLT* value increases from 15 to 270 pixels. The $i < 6$-*best* criterion gives a frame rate of 84 fps on a viewport of 512×512 using an average of 2.9 impostors per frame.

	1 cl.	2 cl.	3 cl.	4 cl.	$i < 6$-*best*
Min	0.21%	0.10%	0.09%	0.07%	0.1%
Max	6.46%	2.28%	1.37%	0.45%	0.37%
Avg.	1.18%	0.44%	0.23%	0.18%	0.17%
PLT	15	44	80	222	270
#imp	1	2	3	4	2.9 avg.
FPS	230	118	80	60	84

Table 1: *Results for the armadillo. The shader uses 16 steps in the linear search, and 7 in the binary search. For the $i < m$ best, over 74% of the time three or fewer impostors suffice. FPS are given for a viewport of* 512×512.

Table 2 compares the interactive inspection results, using the $i < 6$-*best* criterion, for the armadillo, EG dragon, and Buddha models. The quality of the interactive inspections can be perceived from the videos in [vid]. The average coverage error is around 0.16%, and the projected length threshold varies between 169 and 196 pixels for a texture size of 256×256 for each impostor. The average of rendered impostors per frame is under 3. The number of frames per second is around 650 for a viewport size of 128×128, and it remains between 84 and 104 for a viewport size of 512×512.

	Armadillo	EG dragon	Buddha
# faces original	345k	480k	1.087k
Min Coverage Error	0.1%	0.04%	0.04%
Max Coverage Error	0.37%	0.26%	0.35%
Avg. Coverage Error	0.17%	0.14%	0.16%
PLT	169	196	171
Average # impostors	2.9	2.4	2.3
Construction time	5.7s	7.9s	15.2s
Selection time	28s	35s	79s
FPS 512x512	84	102	104
FPS 128x128	644	680	686

Table 2: *Summary performance on three models. Rendering times scale approximately linearly with screen resolution. Note that reported fps's include the per-frame application overhead.*

Textures can be encoded using a total of 2.5 bytes per texel: 1 byte for the depth information, 1 byte for the normal map (with 3Dc compression) and 0.5 bytes for the RGB color, using S3TC's DTX1 color compression. The memory requirements in the GPU considering that four impostors of the ORI are used in each frame is therefore $256 \times 256 \times 2.5 \times 4 = 650$ Kbytes.

Moreover, our approach requires a negligible computational effort in the CPU, and the preprocessing times are significantly small. As shown in Table 2, the ORI construction is between 5.7 seconds for the armadillo and 15.2 seconds for the Buddha, with a complementary cost of 30-80 seconds for the selection and *PLT* computation step (Section 3).

We have conducted experiments (see [vid]) that confirm that rendering multiple instances of an ORI in a scene has essentially no impact on performance, as long as the pixel coverage remains approximately constant, in sharp contrast with rendering the mesh geometry.

The ORI rendering algorithm will benefit from the new parallel computation facilities in the novel GPUs with configurable architectures: given that our algorithm is almost not using the vertex processors, most of the GPU processors can be devoted to the parallel execution of the fragment shader.

5. Hierarchical representation of large scenes

In the case of a hierarchical octree subdivision, we must be able to construct the relief impostor set for any part of the model inside an arbitrary cubical region of the space.

Given a geometric model M in a region R we can split R in two sub-regions R_1 and R_2 and obtain two clipped sub-models M_1 (M in R_1) and M_2 (M in R_2). We can construct the ORIs for M_1 and M_2. An interesting observation on ORIs is that the simultaneous rendering of these two ORIs gives the same visual result than the rendering of the initial ORI of the *model* M, with no artifacts at the junctions (provided that the *PLT* condition is satisfied).

We can construct a hierarchical ORI representation of a complex scene with a straightforward algorithm. We first compute a bounding box B of the scene. In a first step, we build the ORI of the whole scene, and store it in the root node of the hierarchical representation. This information (Section 3) obviously includes the corresponding *PLT* value. Then, we perform a subdivision of B, compute the ORI for each of the resulting regions, store them in the son nodes, and proceed recursively. For the construction of the ORIs of the sub-models, we simply use the algorithm presented in Section 3 and render the model with six clipping planes that bound the corresponding cubical region. In order to avoid artifacts, we inflate each cubical region with an offset value of 1% of the size of the father's edge. Any kind of subdivision (bintree, octree, kD-tree) can be used. The subdivision stops when the *PLT* is larger than a prescribed value, or when a maximum depth of the tree is reached.

Figure 7: *Two snapshots of an interactive exploration of the Armadillo model. Left: rendering with the top-level ORI. Right: rendering the ORIs of the one-level subdivision.*

The view-dependent navigation is based on the *PLT* values of the front nodes in the tree. In a pre-order traversal, a given node is rendered (by rendering its ORI) if its *PLT* is larger than the size of the screen projection of its box. If the PLT is smaller than its projected size, we know that we would have too many covering errors. In this case, the node is not rendered and its sons are inquired with the same test. When the tree leaves are reached, we can render either their ORI or switch to a better representation (triangle mesh) for close objects. Figure 7 presents two snapshots of an interactive inspection [vid] of the Armadillo. When the top-level ORI is used, we exceed the PLT, and we can see some artifacts (fingers at the lower left corner picture) that disappear if we move to deeper levels of the tree and render their ORIs (lower right).

6. Conclusions and future work

We have presented a number of algorithms for the construction, selection and rendering of Omni-directional Relief Impostors. ORIs support parallax effects and advanced per fragment lighting while preserving high frequency details for general view directions. An image-space metric for the selection of a sufficient subset for relief maps for each view direction and for measuring the quality of the rendered images has been proposed and discussed. The omni-directional relief impostors can be used in a hierarchical way for the interactive inspection of complex scenes.

Our approach requires a negligible computational effort in the CPU, and the preprocessing times are significantly small. Due to the large amount of information and detail that is stored in each ORI, the computational effort for the management of the ORIs in large scenes is also reasonably small.

Future work includes the use of indexed textures for reducing the redundancy, and the exploitation of ORI hierarchies for the navigation in huge scenes with a very large number of objects.

References

[ABC*04] ANDÚJAR C., BRUNET P., CHICA A., NAVAZO I., ROSSIGNAC J., VINACUA A.: Computing maximal tiles and application to impostor-based simplification. *Computer Graphics Forum 23*, 3 (2004).

[BD06] BABOUD L., DÉCORET X.: Rendering geometry with relief textures. In *Proc. of Graphics Interface* (Toronto, Ont., Canada, 2006), pp. 195–201.

[CMR*99] CIGNONI P., MONTANI C., ROCCHINI C., SCOPIGNO R., TARINI M.: Preserving attribute values on simplified meshes by resampling detail textures. *The Visual Computer 15* (1999).

[DDSD03] DECORET X., DURAN F., SILLION F., DORSEY J.: Billboard clouds for extreme model simplification. *ACM Transactions on Graphics 22*, 3 (2003), 689–696.

[JWP05] JESCHKE S., WIMMER M., PURGATHOFER W.: Image-base representations for accelerated rendering of complex scenes. In *STAR reports, Eurographics 2005* (2005), pp. 1–20.

[LRC*03] LUEBKE D., REDDY M., COHEN J. D., VARSHNEY A., WATSON B., HUEBNER R.: *Level of Detail for 3D Graphics*. Morgan Kaufmann, 2003.

[OB99] OLIVEIRA M. M., BISHOP G.: Image-based objects. In *Proc. of ACM Symposium on Interactive 3D Graphics* (1999), pp. 191–198.

[OBM00] OLIVEIRA M. M., BISHOP G., MCALLISTER D.: Relief texture mapping. In *Proc. of SIGGRAPH 2000* (2000), pp. 359–368.

[PO06] POLICARPO F., OLIVEIRA M. M.: Relief mapping of non-height-field surface details. In *Proc. of ACM Symp. on Interactive 3D Graphics and Games* (2006), ACM Press, pp. 55–62.

[POC05] POLICARPO F., OLIVEIRA M. M., COMBA J.: Real-time relief mapping on arbitrary polygonal surfaces. In *Proc. of ACM Symposium on Interactive 3D Graphics and Games* (2005), ACM Press, pp. 155–162.

[SDB97] SILLION F. X., DRETTAKIS G., BODELET B.: Efficient impostor manipulation for real-time visualization of urban scenery. *Computer Graphics Forum 16*, 3 (1997), 207–218.

[TCS03] TARINI M., CIGNONI P., SCOPIGNO R.: Visibility based methods and assessment for detail-recovery. In *Proc. of Visualization* (Seattle, USA, October 2003).

[VFSH03] VÁZQUEZ P.-P., FEIXAS M., SBERT M., HEIDRICH W.: Automatic view selection using viewpoint entropy and its application to image-based modeling. *Computer Graphics Forum 22*, 4 (Dec 2003), 689–700.

[vid] `http://www.lsi.upc.edu/~moving/ORIs.html`.

[VK06] VICHITVEJPAISAL P., KANONGCHAIYOS P.: Enhanced billboards for model simplification. In *WSCG* (2006).

[Wil02] WILSON A.: *Spatially Encoded Image-Space Simplifications for Interactive Walkthrough*. PhD thesis, UNC at Chapel Hill, Dept. of Computer Science, 2002.