EUROGRAPHICS '99 / P. Brunet and R. Scopigno
(Guest Editors)

Volume 18 (*1999*), *Number 3*

Multi-layered impostors for accelerated rendering

Xavier Decoret[†] Gernot Schaufler[‡] François Sillion[†] Julie Dorsey[‡]

[†] iMAGIS – GRAVIR/IMAG-INRIA
Grenoble, France

and

[‡] MIT Laboratory for Computer Science
Cambridge, MA, USA

Abstract

This paper describes the successful combination of pre-generated and dynamically updated image-based representations to accelerate the visualization of complex virtual environments. We introduce a new type of impostor, which has the desirable property of limiting de-occlusion errors to a user-specified amount. This impostor, composed of multiple layers of textured meshes, replaces the distant geometry and is much faster to draw. It captures the relevant depth complexity in the model without resorting to a complete sampling of the scene. We show that layers can be dynamically updated during visualization. This guarantees bounded scene complexity in each frame and also exploits temporal coherence to improve image quality when possible. We demonstrate the strengths of this approach in the context of city walkthroughs.

1. Introduction

The interactive visualization of extremely complex geometric datasets is becoming an increasingly important application of computer graphics. Although the performance of graphics hardware has improved dramatically in recent years, the demand for performance continues to grow, as environments containing many millions of polygons become commonplace. In order to visualize such scenes at interactive rates, it is necessary to limit the number of geometric primitives rendered in each frame.

Recently, image-based rendering (IBR) techniques have emerged as an attractive alternative to geometry-based systems for interactive scene display. With IBR methods, the three dimensional scene is replaced by a set of images, and traversing the scene is therefore independent of object space complexity.

To date, two distinct approaches have been explored in the construction of image impostors or caches [1, 2, 3, 4] for the interactive navigation of complex scenes. The first approach involves the pre-generation of an image-based representation for a collection of viewpoints. The advantage of this scheme is that it is possible to deal with excessive and potentially unbounded geometric complexity in a preprocessing step. The disadvantage is that the representation is fixed, so it must be used whenever the point of view is within the associated region of space for which this representation has been generated. Hence, artifacts that are introduced cannot be corrected, even if there is sufficient time and additional information about the viewer's position and viewing direction available during runtime.

In the second approach, impostors are updated dynamically. With this strategy, if the user moves slowly or comes to a complete halt, the image can be progressively refined to the correct image — that is, the one that would have been obtained by rendering the original geometry. A disadvantage of dynamically generated impostors arises from from the potentially unbounded complexity of the geometry that needs to be converted into the image-based representation. As the updates are done on the fly, they must fit into the frame-time budget allocated for generating the final images.

This paper introduces a novel impostor [3] representation that combines pre-generated and dynamically updated impostors into a single framework. We call this new representation the *multi-mesh impostor* (MMI). An advantage of the MMI is that does not enforce a single choice of how to incorporate its contents into the final image. Hence, it allows for a pre-calculated representation where necessary, but also supports a gradual transition to dynamic updates for better quality when feasible within the given frame-time constraints. In addition, the MMI improves on previous image-based scene representations by allowing the user to control the number of occlusion artifacts. By choosing a single quality parame-

ter, the occurrence of typical image-based rendering artifacts such as “rubber sheets” or cracks due to de-occlusion events can be restricted to a given size.

The remainder of the paper is organized as follows. The next section surveys previous work and discusses the strengths and weaknesses of pre-generated and dynamically updated impostors. Section 3 presents a taxonomy of the most common artifacts that occur with existing imposter techniques. Section 4 introduces the multi-meshed impostor (MMI). Section 5 presents an application of the MMI — combined with dynamically updated impostors — to the interactive visualization of large models. Section 6 reports results of the combination of MMIs and pre-generated impostors relative to previous approaches. Section 7 concludes with a discussion of the tradeoffs that have been made in the implementation.

2. Previous work

In this section, we review previous work on pre-generated and dynamically updated image-based representations

2.1. Pre-generated image-based representations

Pre-generated image-based representations make use of off-line resources to deal with the potentially unbounded geometric complexity of a given scene. Researchers have generated planar or omni-directional images to represent distant geometry. Such images must be available for sufficiently close viewpoints so that switching from one image to the next during a walkthrough is seamless. Using image warping usually reduces popping artifacts, although many warping methods introduce problems of their own, such as cracks or rubber-sheet triangles.

In order to cope with these artifacts, the number of viewpoints from which a representation is valid can be increased, but this adds to the considerable storage overhead of image-based methods (consider the memory requirements of one full-screen sized image in true-color of almost three megabytes). These representations must be paged from secondary storage during runtime, as this image data will generally not fit into main memory.

In order of ascending complexity of the sample organization, Grossman et al.[24] and Dally et al.[5] use individual points or small groups of points as samples in the image-based representation. Maciel et al.[3] and Aliaga[6] use planar projections to sample large portions of a model and texture-map them onto polygons for the final image. Chen[7], McMillan et al.[8] and Xiong[28] use environment maps to represent the distant portion of the scene that is surrounding the user. Sillion et al.[2], Darsa et al.[9] and Pulli et al.[10] use meshes to approximate a 3D image warp. Finally, Laveau et al.[11], Max et al.[12, 13] and Rafferty et al.[26] use 3D image warping of the samples.

An alternative approach extends the notion of an image to layered depth images[14] (LDIs) where each pixel contains all the intersections of the ray with the scene. Consequently, any portion of the object’s surface is available for exposure, at the added cost of storage and processing of samples that will not become visible under the set of relevant viewing conditions. This representation must be generated during preprocessing, as current graphics hardware does not support the maintainence of multiple samples along the viewing ray per pixel.

2.2. Dynamically updated image-based representations

With dynamically updated image-based representations, there is little time to convert an image into a more elaborate representation. Instead, the data produced in the frame buffer by the graphics hardware must be reused largely in its current state. Special hardware [15] has been proposed to do this, and on some computers, a unified memory architecture [25] helps in reusing image data for rendering.

When flat image data is not augmented by depth information, one cannot deviate much from the point of view for which the image was generated without noticing the substitution. Warping a depth-image can help increase the lifetime of the image-based representation. However, as long as only one image is warped, occlusion artifacts will appear as surface sections hidden in the reference image become exposed.

In the work of Kajiya et al.[15] and Lengyel et al.[16], the final image is composited from individually updated image layers. The layers each contain a set of non-penetrating objects and are composited back to front by novel video hardware that removes the traditional frame buffer and decouples video refresh from object image refreshes. Shade et al.[1] and Schaufler et al.[4] use polygons with textures depicting portions of the scene. If the viewpoint moves too much, the textures are updated. Regan and Pose [17] use shells of environment maps around the user as the projection of distant objects changes slower under motion of the viewpoint. Each environment map is updated at a rate corresponding to the distance from the map to the user.

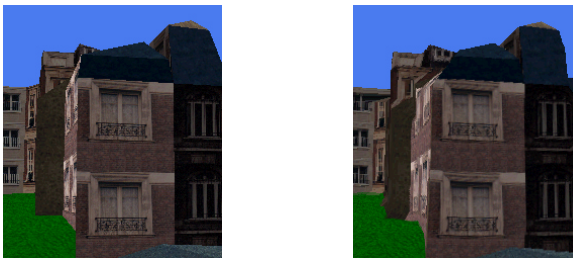
Schaufler[27] has presented a warping technique that is compatible with current graphics hardware architectures. Mark et al.[18] maintain the user’s current view along with a depth map of this image as a representation of the complete scene. By warping this view, they can generate fast intermediate frames at low cost between full renderings of the scene’s geometry. They also demonstrate this approach in a remote server-client setting. A related strategy was developed by Mann et al. [19] in the context of remote navigation of virtual environments.

Taken individually, the above approaches offer advantages, however, as explained in the introduction, a combination of these approaches would be desirable.

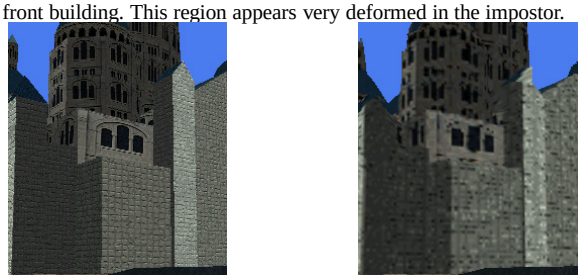
3. Limitations of existing impostor techniques

All of the image-based impostor techniques reviewed above perform some simplification of the underlying 3D model, replacing it with a meshed or point sampled representation. This simplification typically results in a number of potential image artifacts in the final application. The following table presents a taxonomy of the most common of such artifacts. In each case, the left-hand image is created from the geometry and the right-hand image from an impostor for the same viewpoint. The first two problems (**A** and **B**) are due to a poor representation of the model, while the last three (**C**-**E**) are linked to the dynamic behavior of the representation as the viewer moves.

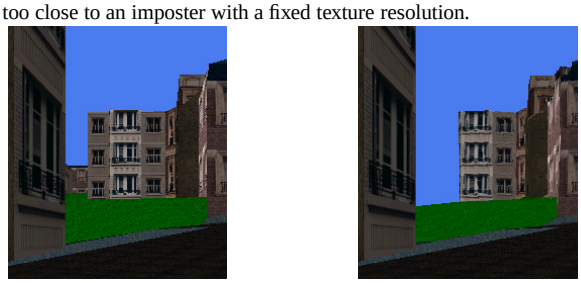
- *(A) Deformation by the impostor representation*: When the geometry of the underlying model is not properly sampled, the resulting imposter representation often appears deformed. To alleviate this problem, it is necessary to perform some analysis of the images to establish a precise correlation between the features of the geometry and the resulting mesh on the imposter [2].
- *(B) Resolution mismatch*:
  Since impostors are created from a sampled representation, they have a "built-in" resolution, which does not necessarily correspond to the image resolution at viewing time. This fixed resolution is usually determined by a predefined viewing distance and is associated with some region of the geometric model. If the viewer deters from these predefined viewing conditions, aliasing artifacts will occur. Appropriate filtering can of course reduce the resolution mismatch, but also creates unwanted blurring of image features and model textures.
- *(C) Incomplete representation*: Building impostors from images limits the amount of information about the geometry that is visible in the reference images. Hence, innapropriate selection of the reference images (or of the set of potentially visible objects) results in objects not appearing when they should be uncovered.
- *(D) Rubber sheet effect*:
  Meshed impostors implicitly force the representation of the model to be continuous in space, as if a sheet of stretchable material is wrapped onto the sampled mesh points. When the viewer moves in such a way that he looks in-between disjoint objects of the model, the view is blocked by this "rubber sheet" connecting foreground and background.
- *(E) Image cracks*:
  Point sampled impostors, on the other hand, suffer from the "cracking" problem, which occurs when no information is present for certain directions from a new viewpoint. This effect can be reduced by splatting or by using multiple depth samples [14].



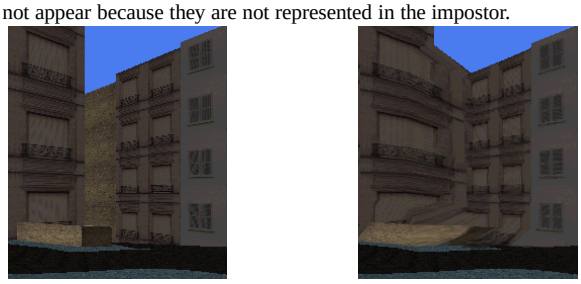
A: Deformation by the impostor representation. A meshed impostor is created without including in the mesh the top-left corner of the front building. This region appears very deformed in the impostor.



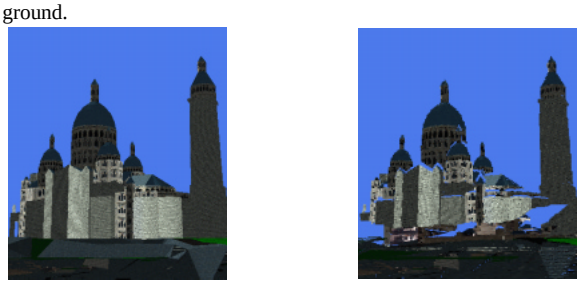
B: Resolution mismatch. Aliasing artifacts occur as the user moves too close to an imposter with a fixed texture resolution.



C: Incomplete representation. As we move past the building on the left, new objects should be uncovered. However, some objects do not appear because they are not represented in the impostor.



D: Rubber sheet effect. In this example, the view to the building in the center is blocked due to a "rubber sheet" that sits in the foreground.



E: Image cracks. "Cracks" appear in this view, especially around the church and tower.

In the next section, we describe a new type of pre-generated meshed impostor that addresses the artifacts caused by the movement of the viewer (items **C**-**E** above). In Sections 5 and 6, we show how this impostor can be combined with dynamic updates to refine the image quality on the fly and address items **A** and **B**.

4. Reducing visibility artifacts using multiple meshes

In order to improve meshed impostors [2, 9] by using multiple meshes, we assume that the environment is divided into a set of *viewing cells*. Given such a cell and three-dimensional geometry representing the model portion visible from that cell, we compute a suitable impostor that will (a) be fast to draw, and (b) correctly represent the geometry for all viewpoints within the viewing cell.

A correct visual representation requires that at least all visibility changes up to a certain magnitude are captured by our impostors. We measure the size of *visibility events* by the change of the angle under which two points on two different objects can be seen from within the viewing cell. This angular variation increases as the points become more separated in depth. Putting them into the same impostor will limit their relative positions in the final image substantially; putting them into different impostors will capture the parallax effects between them.

Considering a point A on an object $\mathcal{O}_1$ and another point B on an object $\mathcal{O}_2$ we need an upper bound for how A and B can move relative to each other in the image (see Figure 1).

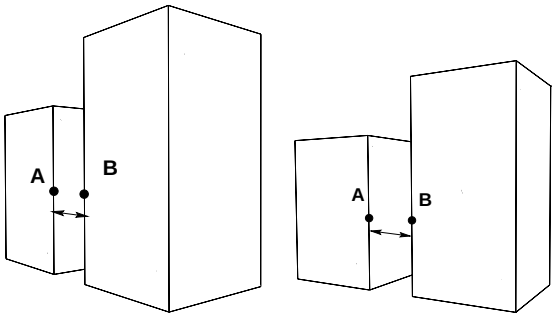


Figure 1: *Two cubes seen from two different points of view. Due to parallax, points A and B have different relative locations.*

4.1. Quantifying the importance of visibility events

Visibility events between two objects can only occur if the two objects overlap in image space as seen from somewhere within the viewing cell. Otherwise, the mesh of the impostor will sufficiently approximate the object's surfaces in 3D.

We define the *critical zone* S of an object as the convex hull of the object and the viewing cell (see Figure 2). If $\mathcal{O}_1$ does not intersect the critical zone of $\mathcal{O}_2$, there can be no overlap in image space. Hence, the importance $w_{i,j}$ of visibility events between $\mathcal{O}_1$ and $\mathcal{O}_2$ is zero. Otherwise, $\mathcal{O}_1$ may cover and uncover parts of $\mathcal{O}_2$, and $w_{i,j}$ must be set to quantify the amount of overlap.

In the general 3D case and with viewing cells of arbitrary shape, this is a difficult problem. We propose a solution for the restricted case where:

- objects have the shape of buildings that are extruded from their footprint on the ground plane,
- viewing cells are line segments (e.g. as streets, paths, etc.),
- the user moves at "ground level," that is mostly horizontally or on a designated terrain.

Under such assumptions, we can compute $w_{i,j}$ for two objects from their orthogonal projections onto a ground plane.

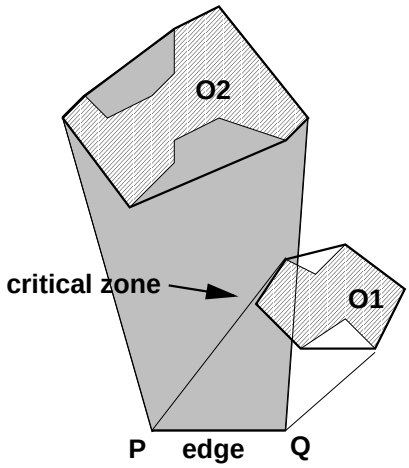


Figure 2: *Critical zone for the object $\mathcal{O}_1$ and the viewing cell $[PQ]$*

To obtain $w_{i,j}$, we derive a formula for the maximum and minimum angles under which a point $M \in [PQ]$ can see the two points A and B on $\mathcal{O}_1$ and $\mathcal{O}_2$. We set $w_{i,j}$ to the difference between these two extremal angles.

In the coordinate system indicated in Figure 3, (x_A, y_A) is defined to be at $(0,1)$.

There are two configurations:

- $y_B = 1$, that is $(AB) \parallel (PQ)$ as on the left of Figure 3.
  Let H be the intersection of $[AB]$'s *median* with (PQ). $\widehat{AMB}$ is increasing for $x_M \in (-\infty, x_H]$ and decreasing for $x_M \in [x_H, +\infty)$. Therefore, the maximum angle occurs for P, Q or H if $H \in [PQ]$, and the minimum is in P or Q.
- $y_B \neq 1$ as on the right of Figure 3.
  In this case, we have:

$$cos(\widehat{AMB}) = \frac{\overrightarrow{AM} \cdot \overrightarrow{BM}}{|AM| \cdot |BM|} \quad (1)$$

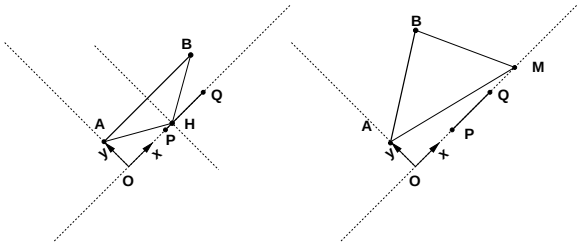


Figure 3: *The two cases for calculating the extremal viewing angles $\widehat{AMB}$.*

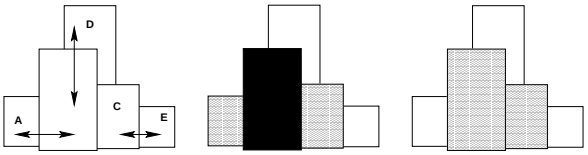


Figure 4: *The decomposition into layers is not unique: (a) represents 5 blocks that have relations as indicated by arrows. We do not consider any weight here. (b) and (c) show two ways of grouping blocks respecting the relations constraints, with 2 and 3 layers, respectively.*

$$= \frac{x(x-x_b)+y_b}{\sqrt{(x^2+1)((x-x_b)^2+y_B^2)}} \quad (2)$$

Since the angle of interest lies in $[0,\pi]$, it is extremal when its cosine is extremal. We found these extrema to be:

$$x_{M_1} = \frac{x_B}{1-y_B} \quad (3)$$

$$x_{M_2} = \frac{x_B+\sqrt{y_B*((y_B-1)^2+x_B^2)}}{1-y_B} \quad (4)$$

$$x_{M_3} = \frac{x_B-\sqrt{y_B*((y_B-1)^2+x_B^2)}}{1-y_B} \quad (5)$$

If those points lie on $[PQ]$ then one of them maximizes and one of them minimizes the angle $\widehat{AMB}$. Otherwise, we know that these extrema occur either at P or at Q.

4.2. A grouping criterion based on visibility events

To partition the objects into layers we construct a graph $\mathcal{G}$ of weighted relations between the objects in the following manner:

- $\mathcal{G}$ has a node per object,
- an edge is created between each pair of nodes with an associated weight $w_{i,j}$,
- the weight $w_{i,j}$ of the edge connecting nodes i and j is set to reflect the importance of visibility events between the two objects as described above.

$$w_{i,j} = \widehat{AMB}_{max} - \widehat{AMB}_{min}$$

- edge weights can be artificially increased to enforce relative importance of some objects such as landmarks or objects of particular interest.

We then partition $\mathcal{G}$ into subgraphs $\{\mathcal{G}_1 \ldots \mathcal{G}_l\}$ such that:

$\forall B_i, B_j \in \mathcal{G}_k \quad w_{i,j} \leq T,$ where T is a given threshold.

In other words, we place those objects into the same layer among which only negligible visibility events can occur. As shown in Figure 4 this decomposition is not unique.

We can express the graph partitioning problem as an n-color problem. If we represent each layer as a color, we want to attribute a color to each node of a graph such that connected nodes have different colors. This problem always has a solution: by allowing as many colors as the number of nodes, each node can be assigned a different color. However, finding the smallest number of colors required to color a non-planar graph is known to be NP-complete [20]. We will apply a heuristic to solve for our particular objective in tractable time.

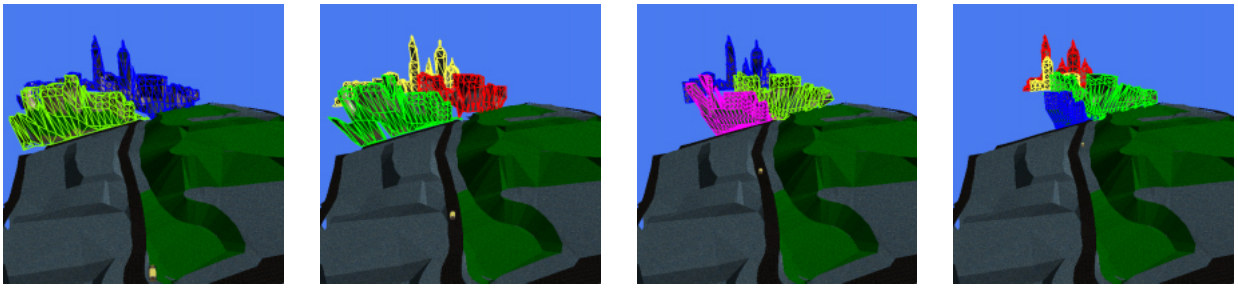
Our incremental algorithm considers one object at a time. It tries to place the object into an existing layer, ensuring that the weight of edges connecting it to objects already in this layer does not exceed a given threshold. If no such layer exists, we create a new layer for this object. When several layers could accept the object, we choose one such that the bounding box of objects in this layer (including the one we want to add) has minimal area. The rationale for this decision is that the closer objects are in the image, the smaller the texture size can be for a desired sampling rate.

4.3. Results of the layer organization

We implemented the described criterion in a city visualization software described further in this paper. For the moment, we just illustrate how our criterion behaves. Figure 5 (see also the color section) shows bird's eye views of the MMIs computed for four different viewing cells. Each viewing cell is a street segment (indicated by the small car in the street, positioned at the segment's starting point). The objects we consider are city blocks. The application of the selection criterion produces variable groupings of the blocks and different numbers of layers. The same 3D geometry set is processed for all four viewing cells. Each layer is used to build a meshed impostor, shown on the images. As the viewing cell gets closer to the sets of objects, the number of requested layers increases, which is the behavior we would expect.

5. Application to the interactive visualization of large models

Making the best use of the capabilities of rendering hardware in the visualization of large models enables a number



2 layers 3 layers 3 layers 4 layers

Figure 5: *MMIs with different numbers of layers are computed for different viewing cells*

of possibilities for how to combine off-line computation, the results of which need to be stored and loaded at runtime; online computation and its distribution among the host CPU; and the graphics subsystem.

5.1. Estimating storage requirements

On the storage-intensive end of the spectrum, MMIs are precalculated for every viewing cell, and several textures, several meshes, and a list of objects in the near geometry need to be stored with every viewing cell. During runtime, the images and meshes of viewing cells adjacent to the current viewing cell are fetched into main memory while the user is moving in the vicinity. Once the user enters the viewing cell, the local geometry together with the image-based representation for the distant geometry is rendered.

Let us illustrate the orders of magnitude of storage required for a sample application. We chose the example of an urban visualization application.

Our city model consists of 572 street segments. In order to represent the distant geometry for each segment with MMIs, we need to store a mesh and a texture for each layer. Expecting an average number of two to three layers per MMI, in the worst case this will result in 572×18 images to store, if we represent the 360^o around the viewing cell with six MMIs. Using images of 256 by 256 resolution, each image requires about 197kB of memory, resulting in a total of image storage of 2.02GB.

The resolution of the meshes of each MMI layer is much less of an issue. As texture coordinates need not be kept for projective texture mapping, a 30 by 30 regular grid of triangle pairs only requires 10.8kB of storage (assuming float coordinates with four bytes per coordinate). This is a very conservative estimate, as many of these triangles will lie outside the silhouettes of the geometric objects, and thus will not make it into the representation. Hence, for 572×18 MMIs, the storage for the mesh vertices would not exceed 111.2MB. With a total of 2.1GB of uncompressed storage, we can apply JPEG compression to the images and geometry compression to the meshes, which should realistically result in a 1:10 compression ratio.

Consequently, our model would fit into less than one third of the space available on a common CDROM, leaving ample space for other data and software.

5.2. Steering online computation from stored data

As the image data obviously requires the vast majority of the storage, we are investigating the tradeoff in terms of storage requirements vs. online generation of the images. Whenever an image is required of a certain model portion, it is only in the case of excessive complexity of this portion that we cannot afford to generate the image at runtime. However, we might want to generate the image off-line, analyze it, and build a very efficient mesh for it — an approach that is clearly too expensive at runtime. Therefore, the mesh and the camera settings are determined and stored for fast retrieval at runtime, but the image is generated online using the camera parameters to be textured onto the mesh.

Even greater flexibility is feasible when the combined rendering of the local geometry and the representation of the distant geometry does not completely consume the frame-time budget. The local geometry's complexity could be chosen to be low enough so that time remains to improve on imperfections that rendering pre-generated representations might exhibit. In particular, the pre-generation has no accurate information about from where the representation will be viewed. Only approximate information in the form of the shape and size of the viewing cell is available off-line.

In contrast, during the walkthrough phase, the precise position and viewing direction are known to the system and should be used to generate the most accurate image under the given time and resource constraints. The improvement will involve the accurate geometric positioning of all image elements, but also any view-dependent effects such as specular reflection effects, lighting modifications (if the virtual viewer carries a light source), or view-dependent selection of better textures. Image-based techniques allow to amortize the cost of more accurate rendering over several frames,

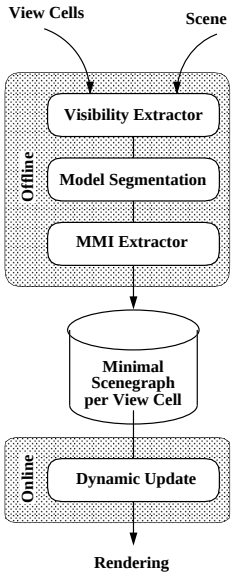


Figure 6: *The pipeline of geometry extractors.*

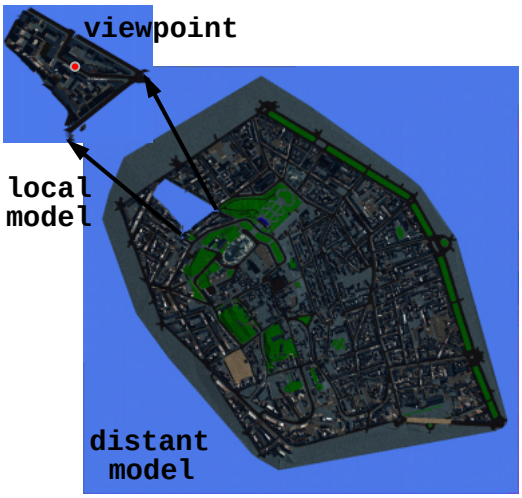


Figure 8: *Schematic overview of the scene partitioning enforced by our system. A zone of near geometry rendered as polygons surrounds the user. More distant sections are replaced by image-based representations.*

thereby exploiting the coherence in the frames to be generated. Off-line computations produce a single representation for a given viewing cell and only online techniques can improve this representation with the knowledge of the current viewing conditions.

In order to facilitate easy experimentation with such trade offs, our software architecture models the transformation of geometry from the full scene to the efficient representation to be generated per viewing cell as a processing pipeline. The geometry travels down this pipeline to be transformed into (approximatively) equivalent, more efficient representations. We refer to each stage in this pipeline as a geometry extractor. Our current pipeline of extractors is depicted in Figure 6.

First, we determine the potentially visible portion of the geometry by conservative visibility culling; next, we partition the model into near and far geometry. Finally, we extract an image-based representation for the distant part of the geometry. This arrangement also allows flexibility as to where to divide this pipeline into preprocessing and online computation. Currently, we support one pipeline stage of online computation, namely dynamic updates of image-based representations.

5.3. Visibility culling

A number of efficient techniques have been presented recently that facilitate the quick and accurate determination of a tight superset of the visible geometry as seen from a certain region of space (its PVS). We can pre-process our scene to find the PVS for all the viewing cells of our model.

In our current implementation, we use a sampling algorithm to estimate the PVS for viewing cells. It operates by taking full 360^o views of the environment from closely spaced points inside the viewing cell and recording the visible objects into an item buffer. This can be implemented efficiently in graphics hardware by assigning unique identifiers to objects coded into the RGB color channels and reading back the images. Any identifier found in the images denotes an element of the PVS. An example of the model portion identified by this approach is given in Figure 7. Note that in order to provide a frame of reference, the terrain geometry is not culled in these images.

We could easily plug in other algorithms by implementing them as geometry extractors. Interesting algorithms include those of Cohen-Or. et al.[21] and Coorg and Teller [22].

5.4. Model segmentation

Our approach is to partition the scene into two sections, the near geometry (the portion of the scene close to the viewpoint — this part will be rendered as geometry) and the far geometry (everything else — this part will be rendered with image-based representations) [2]. This partitioning, in particular the allowable complexity of the near geometry, depends on the rendering capabilities of the target system. In any case, the near geometry should be chosen in such a way that it can be rendered together with a suitable representation of the far geometry in less than the time available for a single frame. Figure 8 shows an example of this segmentation.

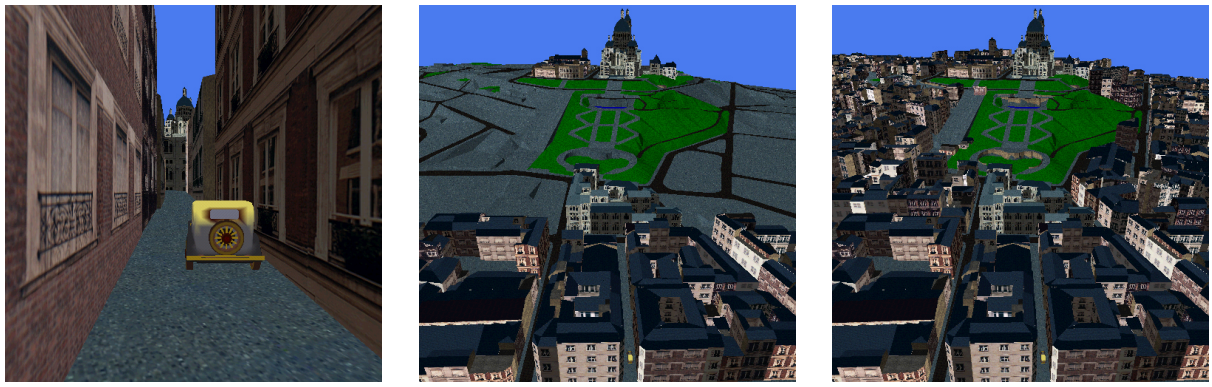


Figure 7: *From left to right: a view from the street, the set of potentially visible city blocks for the current viewing cells, and the entire model. Terrain geometry is not culled in order to maintain a frame of reference.*

5.5. Selective dynamic updates of impostors

An analysis of the primitive count in the near geometry together with its associated MMIs allows us to predict an accurate expected frame time for each viewing cell. We do not choose the near geometry's and the MMIs' combined complexity to deplete the whole frame time, but instead we strive to reduce both complexities as much as possible through visibility calculations and calculating efficient meshes for the MMIs in order to free frame time for dynamic updates. From the desired and the predicted frame times we can determine how much time we have to accomplish such updates.

Whenever free frame time remains in a viewing cell, we start to select the front-most meshes from the viewing cell's associated MMIs and convert them to dynamically updatable impostors. We store the textures of our meshes with a limited depth information of eight bits. For regular-grid impostors, this depth information is sub-sampled and converted into an efficient mesh of triangle strips. The resulting meshed impostor therefore makes no attempt to capture important depth disparities [2], under the assumption that it will be short-lived. Additional images are then rendered when needed to update the new representation in response to user movement within the viewing cell.

We borrow a criterion from the layered impostor approach to do the updates in such an order, that the gain in image fidelity is maximized. Schaufler [27] proposes to calculate an error angle as a measure of screen space deviation from the accurate image. MMI layers are considered for dynamic update in the order of decreasing error angle. Due to the inherent image warping of regular grid impostors, the images generated will be valid for some time, so more meshes can be switched to dynamic update as the images just generated can be reused.

This strategy continues as long as the coherence in the image sequence is sufficiently high and will eventually replace all the meshes of the current MMIs with a new texture. Sudden fast motion of the viewpoint will force us to switch back to MMIs only.

Figure 9: *View of a landmark in our model ("Sacre Cœur" church).*

6. Results

We have implemented the described walkthrough system in C++ using the OpenGL graphics API. The city model shown in the figures throughout this paper represents a section of the city of Paris in France known as Montmartre (see Figure 9). It consists of 143,500 polygons which have been textured using 34 texture maps. There are 146 blocks of houses and a street graph of 572 street segments connected by 427 junctions in the model.

6.1. Using MMIs

Figure 11 (see color section) shows the improvement in image quality obtained when pre-generating and rendering MMIs instead of SMIs. Note how the rubber skin triangles have disappeared from the houses toward the end of the sequence.

In order to document the performance of our approach we have recorded a walkthrough across the city model and calculated the average frame rate along this path. In our comparison, we have used the following rendering options:

- *Full Model:* In this rendering mode, the hardware renders the complete scene model without further optimization. This rendering mode gives an idea of the raw rendering performance of the platform.
- *Local Model:* Rendering just the local model sets an upper bound of the frame rate achievable. This geometry needs to be rendered as polygons since it is very close to the viewer and any simplification would be too obvious.
- *Visible Model:* The visible model is the output of our visibility culling extractor (described in Section 5.3). This is the model part, which is found to be visible from the current viewing cell. Its frame rate needs to be achieved by any alternative rendering method in order to be competitive with brute-force hardware rendering.
- *Single Mesh Impostors:* We include the performance of single mesh impostors [2] for comparison.
- *Multi Mesh Impostors:* Multiple meshes are used to capture visibility events amongst objects. Our error criterion was set to a maximum error of 10 pixels in the final frames of a resolution of 512 by 512 pixels, although visually the error is much less due to the image warping done by the meshes.
- *Multi Mesh Impostors with Dynamic Updates:* In order to overcome any artifacts introduced by the pre-generated representations, textures are updated online. Currently, our implementation is not fully optimized for the two rendering platforms used. (They offer different optimized approaches to improve speed by rendering directly into textures on the O2 platform or copying images directly from frame buffer to texture memory on the IR.)

The following table summarizes the frame rates obtained on a SGI O2 workstation, with one R10k processor running at 200 Mhz, and a SGI InfiniteReality workstation with one R10k processor running at 250 Mhz. We have used a single-buffer window configuration to eliminate the effects of synchronizing buffer switching with vertical retraces.

Frame Rate (Hz)	O2	IR
Full Model	1.0	3.3
Local Model	14.5	118.4
Visible Model	4.3	18.4
Single Mesh Impostors	15.9	103.3
Multi Mesh Impostors	10.9	85.0
MMI with Dynamic Updates	6.5	33.1

Table 1: *Frame rates achieved with the discussed rendering methods on two workstations with widely varying graphics performance.*

6.2. Dynamic impostor updates

Figure 10 shows examples of how image quality is improved when applying dynamic updates in addition to pre-generation of MMIs. Each row gives the MMI on the left, an image with dynamic updates in the middle, and an image rendered using geometry on the right.

In row one, dynamic updates are used to improve the polygonal silhouette of the meshes. Note that on the left, the silhouette of the towers in the background is of a triangular shape because of the mesh structure.

For the second row of images, we have selected an excessively large viewing cell to show how pre-calculated meshes cannot maintain a good look over all possible viewing directions of arbitrary geometry. In our model, the cathedral is a single object, and therefore, can only be assigned to a layer as a whole. Consequently, as the cathedral's tower in the front hides the rest of the cathedral, meshing artifacts occur under extreme viewing conditions. Again the middle image shows how this situation can be overcome with dynamic updates.

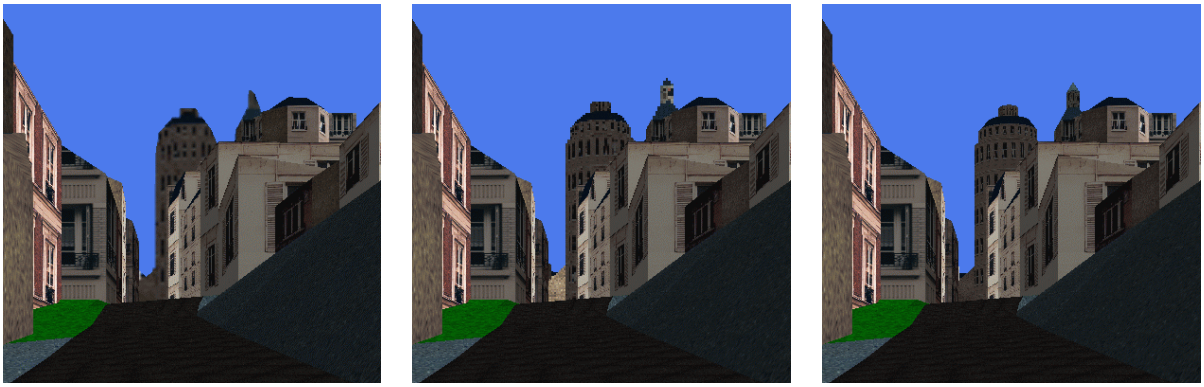
In the third row, an insufficient texture resolution stored in the MMI database is removed by a dynamic update.

7. Conclusions and Discussion

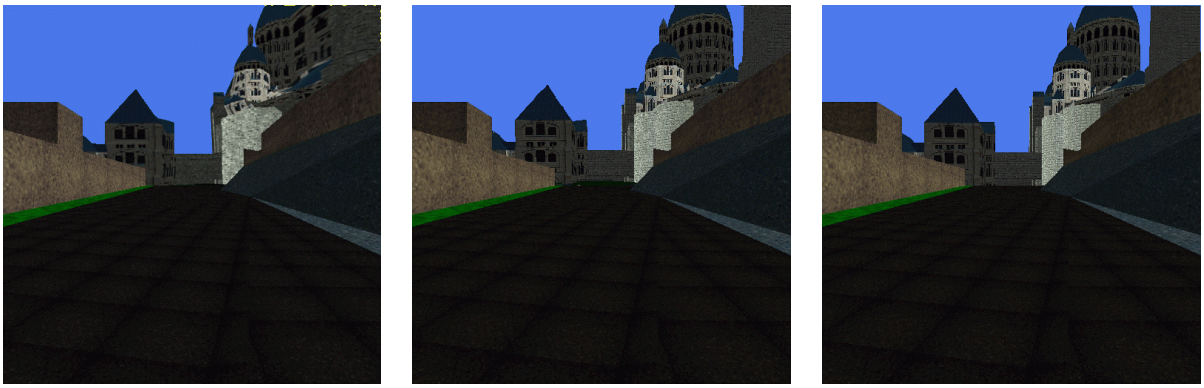
This paper has presented a new type of impostor, the multi-meshed impostor or MMI, which allows us to control the size of the visibility errors in the final image. We hope that this approach to image-based scene simplification will have a major impact on the general acceptance of image-based techniques in real world applications where uncontrolled artifacts are unacceptable.

The control over the visibility errors is obtained by generating MMIs for viewing cells of known size and analyzing the geometry to be replaced by the impostor. As a consequence of the depth differences between various parts of geometry, appropriate representation fidelity is selected. Geometry with too large an extent in depth is placed onto different impostor meshes in order to capture their perspective parallax effects.

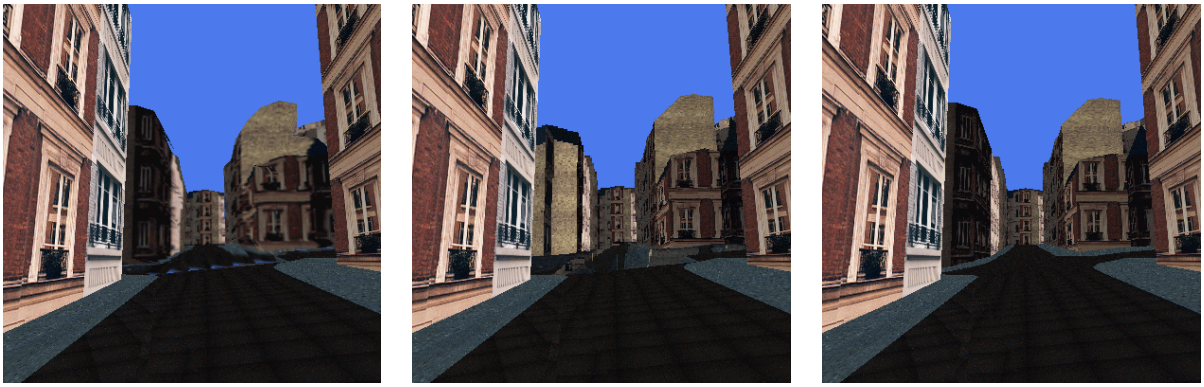
Since purely pre-generated image-based representations often do not allow us to obtain images identical to the ones obtained from geometry, we have combined MMIs with dynamic updates. Whenever frame time is available, our system strives to make the textures used in the image-based



Dynamic updates improve the silhouettes of the meshes.



Mesh distortions are removed from the final image.



Dynamic updates improve insufficient texture resolution.

Figure 10: *Improvement of image quality using dynamic updates. left: rendered using MMIs, center: rendered with dynamically updated mesh, right: rendered with geometry.*

representation converge towards the accurate image of the objects. Switching to dynamic updates or going back to pregenerated representations is especially easy in our system, as both representations build on the same image format.

The result of this combination is that the images generated by our system more closely resemble the images generated from geometry, and in the case of slow motion of the viewpoint, converge to the accurate image.

We still see potential to improve our approach in the way that we partition the model into different regions most suited for a particular type of image-based representation. In particular, we hope to decrease the size of the near model, where currently we are required to render the full geometric complexity. Level of detail approaches cannot help here either, as this part of the model dominates the user's field of view. Also, a more gradual transition from the near geometry to the distant representation would be desirable.

Acknowledgments

This work was supported in part by a joint collaborative research grant of NSF and INRIA (INT-9724005), an Alfred P. Sloan Foundation Research Fellowship (BR-3659), and by a grant from Intel Corporation. It also builds on preliminary work performed by Yann Argotti and Sami Shalabi. iMAGIS is a joint research project of CNRS, INPG, Université Joseph Fourier/Grenoble-I and INRIA.

References

1. Jonathan Shade, Dani Lischinski, David Salesin, Tony DeRose, and John Snyder. *Hierarchical image caching for accelerated walkthroughs of complex environments.* In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 75–82. ACM SIGGRAPH, Addison Wesley, August 1996.

2. François Sillion, George Drettakis, and Benoit Bodelet. *Efficient impostor manipulation for real-time visualization of urban scenery.* In D. Fellner and L. Szirmay-Kalos, editors, *Computer Graphics Forum (Proc. of Eurographics '97)*, volume 16, Budapest, Hungary, September 1997.

3. Paulo W. C. Maciel and Peter Shirley. *Visual navigation of large environments using textured clusters.* In Pat Hanrahan and Jim Winget, editors, *1995 Symposium on Interactive 3D Graphics*, pages 95–102. ACM SIGGRAPH, April 1995.

4. Gernot Schaufler and Wolfgang Sturzlinger. *A three-dimensional image cache for virtual reality. Computer Graphics Forum*, 15(3):C227–C235, C471–C472, September 1996.

5. William J. Dally, Leonard McMillan, Gary Bishop, and Henry Fuchs. *The delta tree: An object-centered approach to image-based rendering.* Technical Memo AIM-1604, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, May 1996.

6. Daniel G. Aliaga. *Visualization of complex models using dynamic texture-based simplification.* In *IEEE Visualization '96*. IEEE, October 1996. ISBN 0-89791-864-9.

7. Shenchang Eric Chen. *Quicktime VR - an image-based approach to virtual environment navigation.* In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 29–38. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995.

8. Leonard McMillan and Gary Bishop. *Plenoptic modeling: An image-based rendering system.* In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 39–46. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995.

9. L. Darsa, B. Costa, and Amitabh Varshney. *Walkthroughs of complex environments using image-based simplification. Computers & Graphics*, 22(1):55–69, February 1998. ISSN 0097-8493.

10. Kari Pulli, Michael Cohen, Tom Duchamp, Hugues Hoppe, Linda Shapiro, and Werner Stuetzle. *View-based rendering: Visualizing real objects from scanned range and color data.* In Julie Dorsey and Philipp Slusallek, editors, *Eurographics Rendering Workshop 1997*, pages 23–34, New York City, NY, June 1997. Eurographics, Springer Wien. ISBN 3-211-83001-4.

11. Stephane Laveau and Olivier Faugeras. *3-D scene representation as a collection of images and fundamental matrices.* Technical Report RR-2205, Inria, Institut National de Recherche en Informatique et en Automatique.

12. Nelson Max. *Hierarchical rendering of trees from precomputed multi-layer Z-buffers.* In Xavier Pueyo and Peter Schröder, editors, *Eurographics Rendering Workshop 1996*, pages 165–174, New York City, NY, June 1996. Eurographics, Springer Wien. ISBN 3-211-82883-4.

13. N. Max and K. Ohsaki. *Rendering trees from precomputed Z-buffer views.* In *Eurographics Rendering Workshop 1995*. Eurographics, June 1995.

14. Jonathan W. Shade, Steven J. Gortler, Li-wei He, and Richard Szeliski. *Layered depth images.* In Michael Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 231–242. ACM SIGGRAPH, Addison Wesley, July 1998. ISBN 0-89791-999-8.

15. Jay Torborg and Jim Kajiya. *Talisman: Commodity Real-time 3D graphics for the PC.* In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 353–364. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

16. Jed Lengyel and John Snyder. *Rendering with coherent layers.* In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 233–242. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7.

17. Matthew Regan and Ronald Pose. *Priority rendering with a virtual reality address recalculation pipeline.* In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 155–162. ACM SIGGRAPH, ACM Press, July 1994.

18. William R. Mark, Leonard McMillan, and Gary Bishop. *Post-rendering 3D warping.* In Michael Cohen and David Zeltzer, editors, *1997 Symposium on Interactive 3D Graphics*, pages 7–16. ACM SIGGRAPH, April 1997. ISBN 0-89791-884-3.

19. Y. Mann and D. Cohen-Or. *Selective pixel transmissionfor navigating in remote virtual environments. Computer Graphics Forum*, 16(3):201–206, August 1997. Proceedings of Eurographics '97. ISSN 1067-7055.

20. M.R. Garey and D.S. Johnson. *Computer and Intractibility: a guide to the theory of NP-completeness*, page 191. W.H. Freeman, 1979.

21. Gadi Fibich Dan Halperin Cohen-Or, Daniel and Eyal Zadicerio. *Conservative visibility and strong occlusion for viewspace partitioning of densely occluded scenes. Computer Graphics Forum*, 17(3):243–255, August 1998. Proceedings of Eurographics '98. ISSN 0167-7055.

22. Satyan Coorg and Seth Teller. *Real-time occlusion culling for models with large occluders.* In Michael Cohen and David Zeltzer, editors, *1997 Symposium on Interactive 3D Graphics*, pages 83–90. ACM SIGGRAPH, April 1997. ISBN 0-89791-884-3.

23. Gortler, Steven J., Li-Wei He, and Michael F. Cohen, *"Rendering Layered Depth Images"*, Microsoft Research Technical Report MSTR-TR-97-09.

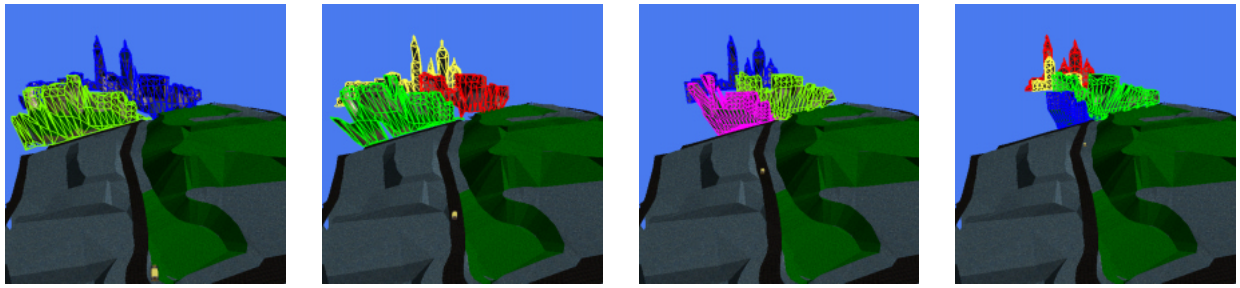
24. Grossman, J.P., and William J. Dally, *"Point Sample Rendering"*, 9th Eurographics Workshop on Rendering, August 1998, pp 181-192.

25. SGI O2, *"O2: Unified Memory Architecture"*, Silicon Graphics Datasheets and White Papers, April 1997, available at `http://www.sgi.com/o2/uma.html`.

26. Rafferty, Matthew M., Daniel G. Aliaga, Anselmo A. Lastra, *"3D Image Warping in Architectural Walkthroughs"*, VRAIS 98, pp. 228-233.

27. Schaufler, G., *"Per-Object Image Warping with Layered Impostors"*, in Proceedings of the 9th Eurographics Workshop on Rendering '98, Vienna, Austria, June 29-July 1, 1998, pp 145-156.

28. Xiong, Rebecca, *"CityScape - A Virtual Navigational System for Large Environments"*, MIT Masters Thesis, 1996.



2 layers 3 layers 3 layers 4 layers

Figure 5: *MMIs with different numbers of layers are computed for different viewing cells*

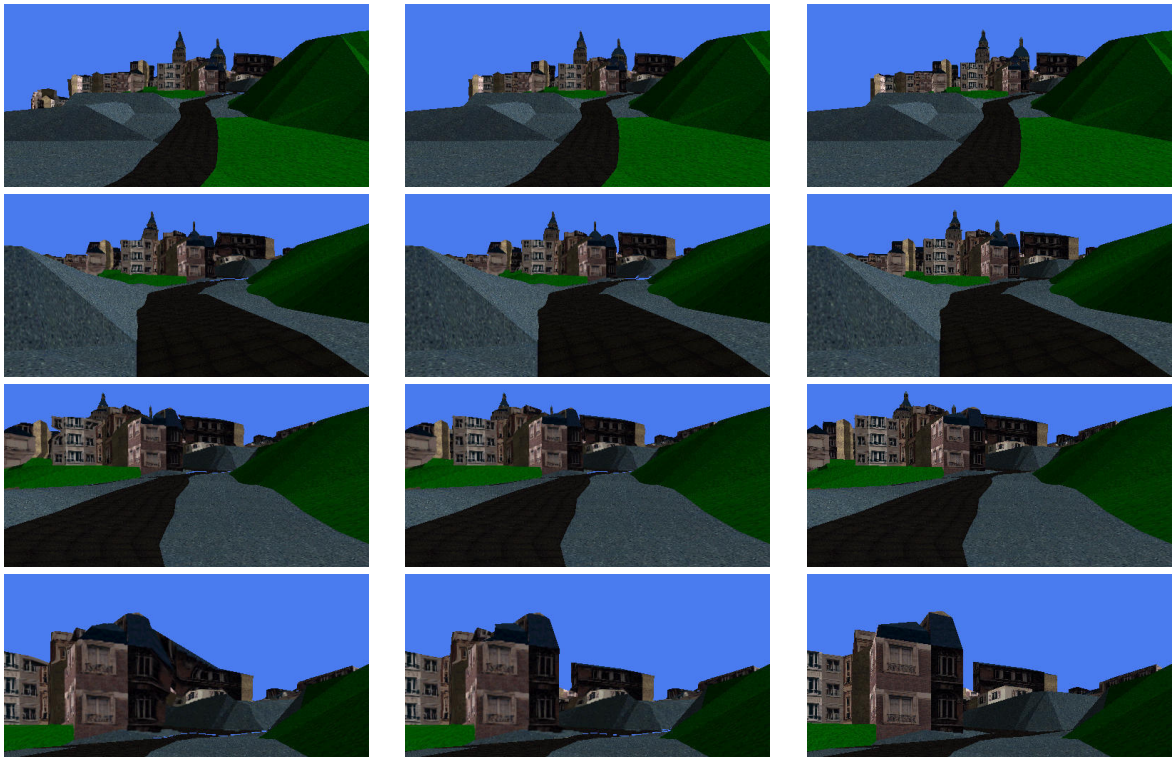


Figure 11: *Comparison of single-mesh impostors* [2] *and MMI impostors computed for a given distant model and the various viewing cells along a road. Left: Single-mesh impostors Middle: Multi-mesh impostor. Right: Actual geometry.*