Point-Based Impostors for Real-Time Visualization

Michael Wimmer, Peter Wonka, François Sillion
iMAGIS - GRAVIR/IMAG-INRIA, Vienna University of Technology

Abstract. We present a new data structure for encoding the appearance of a geometric model as seen from a viewing region (view cell). This representation can be used in interactive or real-time visualization applications to replace a complex model by an impostor, maintaining high quality rendering while cutting down rendering time. Our approach relies on an object-space sampled representation similar to a point cloud or a layered depth image, but introduces two fundamental additions to previous techniques. First, the sampling rate is controlled to provide sufficient density across all possible viewing conditions from the specified view cell. Second, a correct, antialiased representation of the plenoptic function is computed using Monte Carlo integration. Our system therefore achieves high quality rendering using a simple representation with bounded complexity. We demonstrate the method for an application in urban visualization.

1 Introduction

Fig. 1. A scene from an urban walkthrough. Geometry in the red region has been replaced by a point-based impostor, providing for fast, antialiased rendering of the far field.

A general and recently quite popular approach to handle the interactive display of complex scenes is to break down the view space into cells (*view cells*) and compute optimizations separately for each view cell. A typical example is region visibility, which calculates the potentially visible set (PVS) of objects from a specific view cell.

In this paper we address the problem of simplifying distant geometry (the far field) for view cells (see Fig. 1). This is an important problem because rendering distant geometry often leads to aliasing artifacts and, even after visibility calculations, the amount of geometry remaining in the PVS is often overwhelming for current graphics accelerators.

We begin by defining our goals in this process. First, we set out to build a representation that provides a high-quality rendered image for all views from a specific view cell. This means that we try to avoid artifacts such as holes, aliasing or missing data, and that we require high fidelity in the rendering of view-dependent appearance changes. Second, we insist that our representation be as compact as possible, while being amenable to hardware acceleration on consumer-level graphics hardware.

Distant geometry has some peculiarities resulting from its complexity that make simplification difficult:

- One aspect is that because of limited image resolution and perspective projection, typically several triangles project to a single pixel. This makes antialiasing and filtering an important issue.
- Another aspect is that we can not rely on surfaces to drive the simplification process: the triangles that project to one pixel can stem from disconnected surfaces and even from different objects. This means, for example, that there is no well-defined normal for such a pixel.

Generally, simplification can be based either on geometric criteria as in most level-of-detail approaches, or on image-based representations. Geometric simplification is useful for certain models, but has its limitations. Scenes with alpha textures, regular structures as found in buildings, chaotic disconnected structures as found in trees, or new shading paradigms such as pixel and vertex shading are not easy to deal with even in recent appearance-based simplification methods. We will therefore concentrate on image-based simplification approaches.

We observe, however, that none of the simplified representations proposed to date meets all the goals defined above:

- Layered depth images (LDIs) and similar representations introduce bias in the sampling patterns, and cannot accommodate situations in which many primitives project onto each pixel of a view. In the presence of such *microgeometry*, we also have to be aware that appearance can change with the viewing direction. This requires a representation with directionally dependent information, such as a light field.
- Light field approaches are powerful enough to cope with complex and directionally dependent appearance. However, geometric information needs to be incorporated into the light field in order to obtain acceptable quality within a reasonable amount of storage space [3]. A possible starting point to build an impostor would therefore be a surface lightfield [28], but as the name of this primitive already indicates, this requires a surface (and its parameterization).

In this paper we introduce a new representation for a complex geometric model that is especially useful to build impostor objects to replace the far field for a given view cell. The proposed representation effectively decouples the geometry, represented as a set of 3D points, and the appearance, represented as a simplified light field computed from the original model. In our algorithm, the points are selected from a number of viewpoints, chosen so as to approach a minimal sampling criterion in image space, across the entire view cell. Therefore our sampling mechanism prevents the appearance of holes. In the context of very complex models composed of a great many independent primitives, 3D points can not be considered as representatives of a well-defined surface, as in the surfel or QSplat techniques [20, 21]. In contrast, we define a proper characterization of the radiance contribution to the image from these points, which can be computed using Monte Carlo integration, and is encoded as a texture map to model

view-dependent effects. The resulting representation is compact, renders quickly using existing graphics hardware, produces a high quality image with little aliasing, has no missing or disappearing objects as the viewpoint moves, and correctly represents view-dependent changes within the view cell (such as occlusion/disocclusion effects or appearance changes).

The paper is organized as follows: after reviewing previous work in section 2, we present an overview of the proposed representation, its construction and its usage in section 3. We then provide in section 4 an in-depth analysis of the point-sampled rendering primitive, its meaning in terms of appearance modeling and theoretically founded means of representing image radiance information at these points. Section 5 presents the details of the current implementation, which computes impostor representations for the far field. Results are presented and discussed in section 6.

2 Previous Work

Our work can be seen in the context of image-based rendering. The idea of image-based rendering is to synthesize new views based on given images. Ideally, we would like to replace distant geometry by one image, an alpha texture map, because it is very fast to render [14]. Schaufler et al. [22] and Shade et al. [24] used this idea to build a hierarchical image cache for an online system, with relaxed constraints for image quality. However, to properly capture parallax effects of the replaced geometry, it would be necessary to precalculate many alpha texture maps for several viewpoints in the view cell and to blend between them dependent on the viewing position. This is basically equivalent to storing the five-dimensional plenoptic function (the function that encodes all possible environment maps from any point in the scene [17]) and requires too much memory if high image quality is desired. A more efficient solution is a 4D parameterization of the plenoptic function, the light field [11], which can be seen as a collection of images taken from a regular grid on a single plane. At runtime it is possible to synthesize images for new viewpoints not only on this plane, but also for all viewpoints within a certain view cell behind the plane. Gortler et al. [9] independently developed a similar parameterization. They additionally use depth information for better reconstruction. Depth information can also be added as a triangle mesh to create surface light fields [19, 28]. Shum et al. [3] study the relationship between depth and (spatial) spectral support of the light field in more detail and add depth information to the light field in layers. Although light fields can be rendered interactively, memory consumption and calculation times make it hard to use them for most real-time rendering applications.

Starting from the texture map idea, depth information can be added to make an image usable for a larger number of viewpoints, for example by using layers [18, 23]. These layers can be rendered quickly on existing hardware, but they contain a strong directional bias which can lead to image artifacts, especially in complex scenes. Several authors have added depth information to images using triangles [1, 7, 8, 15, 26]. While a (layered) depth mesh can be calculated and simplified for one viewpoint, the representation is undersampled for other viewpoints, leading to disocclusion artifacts or blurring effects. The calculation of an equally sampled high quality triangle mesh remains an open problem. Finally, depth can be added per point sample. In particular, layered depth images (LDI) [16, 25] provide greater flexibility by allowing several depth values per image sample. However, warping the information seen from a view cell into the image of a single viewpoint again leads to a sampling bias. To overcome this problem, several LDIs have to be used [4, 13].

As an alternative, point-based rendering algorithms were investigated by Levoy et

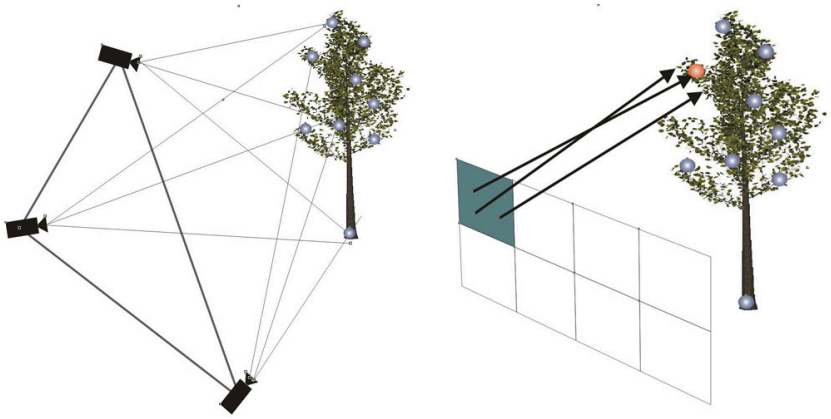

Fig. 2. *Left:* Geometry is sampled using three perspective LDI cameras from a view cell. *Right:* Rays are cast from the view cell to calculate a Monte Carlo integral. A texture map on the sampling plane records contributions for different view cell locations.

al. [12], Grossman et al. [10] and Pfister et al. [20]. These algorithms are currently not implemented in hardware and especially hole filling is a challenging problem. For faster rendering, warping can be replaced by hardware transformation together with a splatting operation [21]. The main argument for the use of points is their simplicity. However, in the context of complex geometry, points can no longer be seen as point samples on a surface [20, 21]. Therefore, our work stands in contrast to previous point rendering techniques and we will derive a characterization of points that differs fundamentally from those approaches.

3 Overview of the algorithm

We propose a rendering primitive based on points to replace geometry as seen from a view cell. The system implements the following pipeline:

- Sampling geometry: we calculate sample points of the geometry using three perspective LDIs to obtain a dense sampling for a view cell (Fig. 2, left). We call the plane defined by the three camera locations the *sampling plane*.
- Sampling appearance: The antialiased appearance of each point is calculated for different view cell locations using Monte Carlo integration. We shoot rays from a rectangular region which is contained in the triangle formed by the three cameras (see Fig. 2, right). Each point is associated with a texture map which encodes the appearance contributions for different view cell locations.
- Real-time display: point-based impostors make full use of existing rendering hardware for transforming each point as well as shading it with its associated texture map depending on the current viewer location. No additional software calculations are necessary.

4 Point-Based Impostors

4.1 Complexity of appearance

The plenoptic function $P(s, \varphi)$ completely models what is visible from any point s in space in any given direction φ. Rendering is therefore the process of reconstructing parts of the plenoptic function from samples. Both geometry and appearance information of scene objects contribute to the plenoptic function. Current rendering algorithms usually emphasize one of those aspects:

- Light fields record rays regardless of geometric information, even if they all hit the same diffuse surface.
- Other, geometric approaches usually model appearance only to a degree allowed by their lighting model. They cannot account for *microgeometry*, i.e., geometry that projects to less than a pixel. Microgeometry can have different appearance when viewed from different angles (see Fig. 3, left). This usually results in aliasing artifacts or is dealt with by blurring, thereby discarding possibly important information.

Our point-based representation contains both geometric information (the 3D location) and directionally dependent appearance information. It therefore captures all of these aspects:

- Point sampling can easily adapt to unstructured geometry in the absence of surfaces or a structured model.
- A point on a diffuse, locally flat surface which projects to more than a pixel has the same appearance from all directions and can be encoded with a single color.
- Objects with view-dependent lighting effects (e.g. specular lighting) show different colors when viewed from different angles.
- Microgeometry is also view dependent and is encoded just like lighting effects.

4.2 Geometric sampling

The first step in the creation of a point-based rendering primitive is to decide on the geometric location of the sample points. The points encode the part of the plenoptic function defined by the view cell and the geometry to be sampled. To allow hardware reconstruction, there should be no holes when projecting the points into any possible view from the view cell.

A sufficient criterion that a point-sampled surface will have no holes in a projection is that the projected points can be triangulated so that the maximum projected edge-length is smaller than the side length of a pixel [10].

It is in general difficult to prove that a particular point-based representation fulfills this criterion. For unit magnification and orthographic viewing, three orthogonal LDIs provide such an adequate sampling [13]. Although this sampling strategy works with our method, it is not well suited for far-field objects because the sampling density is not adapted to the possible viewing directions from the view cell and typically results in a large number of point samples. Therefore, our method chooses perspective LDIs to better distribute the samples with respect to the view cell.

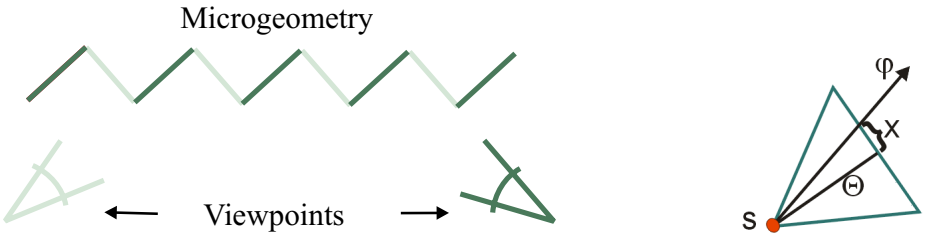

Fig. 3. *Left:* Microgeometry: if the structure in the figure projects only to a pixel for the view cell, directional appearance information is required to let it appear shaded light in the left camera and dark in the right camera. *Right:* The figure shows the relation of the plenoptic function and plenoptic image function parameters.

4.3 Appearance: the plenoptic image function

The geometric sampling step results in a set of n points $p_1, \ldots, p_n$ with fixed locations in 3D space. In this section, we derive a method to determine color values for each of these points. We examine the meaning of a point in light of our goal: for each viewing location and camera position in a view cell, the image obtained by rendering all points $p_1, \ldots, p_n$ should be a correctly filtered image of the far field objects (i.e., one slice of the plenoptic function $P(s, \varphi)$. Our derivation will show that several colors per point are needed for that.

Most rendering algorithms use a z-buffer or depth ordering to determine the visibility of the primitives they render. A point, however, is infinitely small. Its meaning is mainly determined by its reconstruction. In many reconstruction methods (as, for example, in hardware rendering) only one point is visible per pixel and determines the color of the pixel. Due to finite image resolution, a point can be visible or occluded depending only on the viewing camera orientation. The plenoptic function is not powerful enough to model this form of occlusion.

Consequently, we introduce the *plenoptic image function* $PIF(s, \theta, x)$. The parameter s represents the 3D viewer position, θ is a camera orientation, and x is a 2D pixel coordinate in the local camera coordinate system. Fig. 3 (right) illustrates how the domains of the *PIF* and the plenoptic function relate to each other: any pair (θ, x) corresponds to one ray orientation $\varphi(\theta, x)$ from the plenoptic function, so under ideal conditions (i.e., infinitely precise displays etc.), the *PIF* is related to the plenoptic function P via

$$PIF(s, \theta, x) = P(s, \varphi(\theta, x))$$

Note that this mapping is many to one. The *PIF* with its additional parameters will allow us to incorporate the visibility term inherent to current reconstruction techniques directly into our calculations.

We now interpret points via the images they produce (their image functions) when rendered with a specific rendering algorithm—in our case, z-buffered hardware rendering.

If we consider only one point p_j alone, the point defines an image function $r_j(s, \theta, x)$. This function specifies a continuous image for each set of camera parameters (s, θ). Each of those images is set to the typical reconstruction filter of a monitor, a Gaussian centered at the pixel which the point projects to (see Fig. 4, top). However, it is very crucial to see a point p_j in relation to the other points in the point set. We have to take

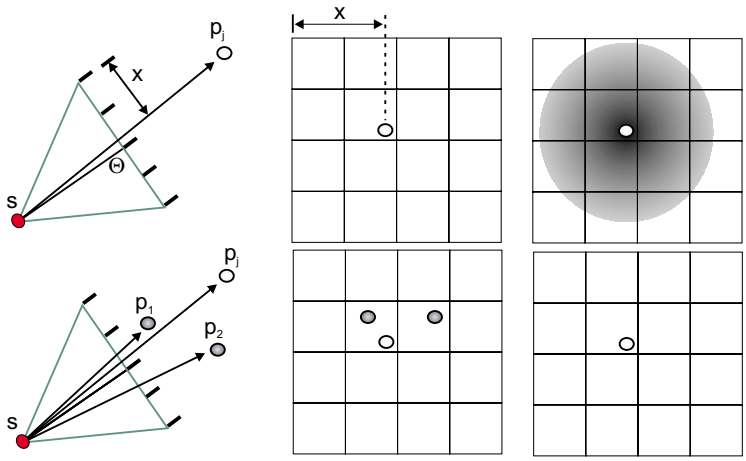

Fig. 4. These figures explain the image function of a point p_j and the influence of visibility. The left column shows a 2D cross-section for fixed camera parameters (s, θ). Note that we only show one component of the 2D image coordinates x. The central column shows the point p_j in the resulting image. The top right figure shows the image function $r_j(s, \theta, x)$. While the top row considers only point p_j, in the bottom row, two additional points p_1 and p_2 have been added. Here, the point p_1 occludes the point p_j due to the z-buffer, causing the visibility term $v_j(s, \theta)$ of point p_j to be 0. Therefore, $Q_j(s, \theta, x)$ (in the bottom right figure) is 0 for all image coordinates x.

into account a visibility term v_j which evaluates to 1 if the point p_j is visible in the image defined through the camera parameters (s, θ) and 0 otherwise. This gives us the actual image function Q_j of a point (see Fig. 4, bottom):

$$Q_j(s, \theta, x) = v_j(s, \theta) r_j(s, \theta, x)$$

From an algebraic point of view, we can regard the functions $Q_j, 1 \leq j \leq n$, as basis functions spanning a finite dimensional subspace of the space of all *PIF*s. One element PIF_{finite} of this finite dimensional subspace can be written as a weighted sum of the basis functions of the points:

$$PIF_{finite}(s, \theta, x) = \sum_j c_j Q_j(s, \theta, x)$$

Note that the weight c_j is the color value assigned to the point p_j. The weights c_j should be chosen so as to make PIF_{finite} as close as possible to *PIF*. This can be achieved by minimizing $\|PIF_{finite} - PIF\|$ (with respect to the Euclidean norm on the space of functions over (s, θ, x)), and the resulting weights can then be found via the dual basis functions as

$$c_j = \iiint q_j(s, \theta, x) PIF(s, \theta, x) ds\, d\theta\, dx$$

where $q_j(s, \theta, x)$ is the dual basis function (the dual basis functions are defined via $\langle q_j, Q_k \rangle = \delta_{jk}$).

This representation only assigns one color value per point, regardless of the location s in the view cell. However, because of view-dependent shading effects and

micro-geometry as discussed in section 4.1, a point might need to show very different appearance from different viewing locations. Thus, we make the model more powerful by replacing the weights c_j by functions $c_j(s)$. In order to represent those functions, they are discretized with respect to the space of possible viewing locations (this is similar in spirit to the camera aperture necessary when sampling light fields). Given a basis $B_i(s)$ for this space, we can write each $c_j(s)$ in terms of this basis, with coefficients found via the dual basis $b_i(s)$:

$$c_j(s) = \sum_i c_{ij} B_i(s), \text{ where } c_{ij} = \int c_j(s) b_i(s) ds$$

One possible choice for the basis $B_i(s)$ is to use a regular subdivision of a plane with a normal directed from the view cell towards the objects. The weights c_{ij} for each point p_j can then be stored in a texture map. This means that $c_j(s)$ will be looked up in a texture map, which is typically reconstructed with a bilinear kernel, so $B_i(s)$ is actually just a bilinear basis.

Putting all parts together, the rendering process is described with the formula

$$PIF_{finite}(s, \theta, x) = \sum_{i,j} c_{ij} Q_j(s, \theta, x) B_i(s)$$

and the weights to make the PIF_{finite} resemble most closely the *PIF* are calculated as

$$c_{ij} = \iiint PIF(s, \theta, x) q_j(s, \theta, x) b_i(s) ds \, d\theta \, dx \tag{1}$$

The final task is finding the dual basis functions. If our basis functions $Q_j B_i$ were orthogonal, we would have $q_j = Q_j$ and $b_i = B_i$ (apart from normalization issues). In the non-orthogonal case, however, calculating the real duals is tedious: geometric occlusion means each one would be different; one would really have to invert a large matrix for each one to find a (discretized) version of the dual. We have opted for an approximate approach inspired by signal theory: both the bilinear basis B_i and the monitor reconstruction filter r_j can be regarded as an approximation to the ideal (and orthogonal) reconstruction filter, a sinc-function. As is common practice, we approximate the ideal lowpass filter (the signal-theoretic version of dual basis functions) using Gaussians.

The integral (1) can be estimated using Monte Carlo integration with b_i and q_j as importance functions. Samples $PIF(s, \theta, x) = P(s, \varphi(\theta, x))$ are calculated with a simple ray tracer.

5 Implementation

In this section we describe the choices made in our particular implementation of the method.

5.1 Obtaining geometric samples

To place the three LDI cameras used to obtain geometric samples, a sampling plane is chosen to parameterize viewer positions within the view cell: as in light fields, we use a plane oriented from the view cell towards the far field. Then, we calculate the

intersection of the supporting planes of the view cell and the far field with the sampling plane. We select three cameras on a triangle that tightly bounds the resulting polygon (i.e., the triangle with minimum circumference).

In order to determine the LDI resolution necessary to avoid holes in the reprojection of the sampled points, our method offers a sampling mechanism based on a 2D observation: suppose we have two cameras on both ends of a segment of possible viewing locations in 2D, and a short edge viewed by the two cameras. These two cameras are used to sample the endpoints of the edge. Resampling happens from a camera placed anywhere on the segment between the two cameras. It can now be shown that the "worst" discrepancy between the resampling angle and the sampling angle appears if the edge is parallel to the segment joining the two cameras, and if the center of the edge is equidistant to both cameras.

Inspired by this, we have developed a heuristic in 3D which takes the following effects into account:

- Movement within the view cell: for three cameras looking at a plane parallel to the sampling plane, the worst mutual sampling resolution occurs at the point on the plane that is equidistant from the three cameras. This point is the circumcenter of the triangle defined by the three cameras, projected on the plane in question. The sampling angle of the three cameras has to be chosen so that neighboring samples project to less than a pixel when viewed directly from the circumcenter (where the maximum projection occurs).
- Perspective: perspective cameras have varying angular resolutions in space: a feature (e.g., a triangle) projects to more pixels at the border of the viewing frustum than in the center when seen under the same viewing angle. The sampling resolution has to be increased accordingly by the factor between angular resolution in the center of the camera and at the border of the viewing frustum. For a field of view of 45° for example, this factor is about 1.17.
- Sampling pattern orientation: if perspective cameras are used for sampling, the sampling pattern on objects in space is tilted with respect to the viewing camera. Therefore, given a minimum sampling angle from a point, the resolution has to be calculated from the pixel diagonal and not the pixel edge. This accounts for a factor of $\sqrt{2}$.

Sampling resolution is chosen according to these factors. In practice, we employ a simple heuristic to reduce the number of points: many points are sampled sufficiently already by a single camera. This means that a triangulation of the sampling pattern from this camera in a small neighborhood contains no edge which projects to more than a pixel in any view. If a point recorded by one camera lies in a region which is sampled better and sufficiently by another camera, we remove it. This typically reduces the number of points by 40–60%, leading to a ratio of about 2–3 points projected to a screen pixel.

5.2 Monte Carlo integration of radiance fields

The goal of the Monte Carlo integration step is to evaluate integral (1) to obtain the weights c_{ij} of the reconstruction basis functions. The domain in s is a rectangle on the sampling plane. The index c_{ij} corresponds to one point p_j and one rectangle on the sampling plane represented by a texel t_i in the texture map for point p_j.

We select a viewpoint s on this rectangle (according to $b_i(s)$, a Gaussian distribution), a random camera orientation θ in which the point is in the frustum, and shoot

an occlusion ray to test whether this point is visible in the selected camera (this corresponds to an evaluation of $v_j(s,\theta)$). If it is visible, we select a ray according to $q_j(s,\theta,x)$ (a Gaussian distribution centered over the pixel which the point projects to in the selected camera) and add its contribution to t_i. Rays are shot until the variance of the integral falls below a user-selectable threshold.

5.3 Compression and Rendering

Points are rendered with z-buffered OpenGL hardware. For each point, the directional appearance information is saved in a texture, parameterized by the sampling plane. The texture coordinates of a point are calculated as the intersection of a viewing ray to the point with the sampling plane. This can also be interpreted as the perspective projection of the sampling point into a viewing frustum where the apex is defined by the viewpoint and the borders of the projection plane by the four sides of the bounding rectangle on the sampling plane.

Perspective projections can be expressed using the 4x4 homogeneous texture-matrix provided by OpenGL. However, since switching textures for every point is costly, we pack as many point textures into one bigger texture as the implementation allows. This requires adding a fixed offset per point to the final texture coordinate, which, although not available in standard OpenGL, can be done using the vertex program extension [5].

Interpolation between the texture samples (which corresponds to the basis function $B_i(s)$) is done using bilinear filtering. A lower quality, but faster preview of the representation can be rendered by using only one color per point and no texture. However, directional information will be lost in this case.

To compress our representation, we calculate the variance of the color information of a texture to identify points that only require one color for the whole view cell. Further compression can be obtained by using hardware supported vector quantization [6], which provides for a fixed compression ratio of 8:1. Note that this adapts to the scene complexity: regions with low perceived geometric complexity will always be represented by simple colored points.

6 Results

We used an 800 MHz Pentium III with a GeForce II GTS graphics card for our tests. The vertex program used to calculate texture coordinates is simulated in software. To give an impression of the expected performance of a hardware implementation, we also rendered simple textured points.

Three different test scenes are used to demonstrate the behavior of point-based impostors. Table 1 shows results for each scene, based on an output screen resolution of 640x480 pixels. It includes the number of points in the impostor, the approximate number of pixels covered by the impostor for a view in the center of the view cell (based on the screen bounding box), and the resulting number of points per projected pixel. The table also shows the time required for computing the impostor, and the memory requirements. The last two rows represent average frame rates achieved for some positions within the view cell.

Generally, it can be observed that it takes about 75,000 points to cover a screen area of about 300x100 pixels.

Although we are using an unoptimized ray tracer, preprocessing times are still reasonable. For each point, an 8x2 texture map was found to sufficiently capture view-dependent effects for the view cells considered (determining the size of the texture

Results	scene1	scene2	scene3
#points	80,341	87,665	31,252
#points/screen pixel	2.35	2.42	2.8
approx. #screen pixels	34,000	36,000	11,000
Preproc. time (min)	22	41	31
Memory (MB)	1.6	1.75	0.6
Rendering performance SW (Hz)	36	32	98
Rendering performance HW (Hz)	60	54	160

Table 1. The table shows results from impostors of three scenes, calculated for a screen resolution of 640x480 pixels. Hardware rendering is emulated by rendering simple textured points.

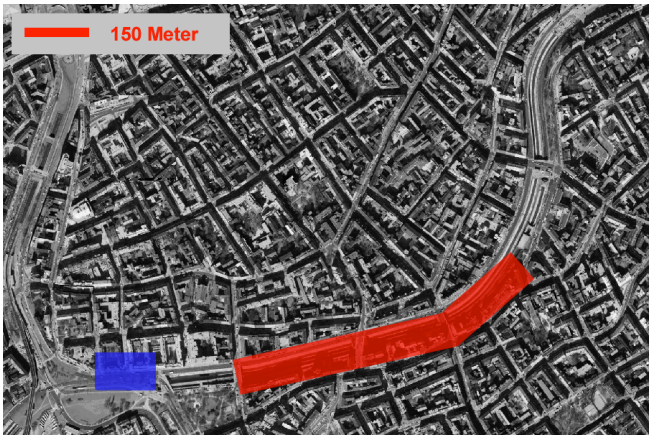

Fig. 5. The figure shows the impostor from scene 3 placed in the city. The view cell shown in the dark rectangle is 63 meters wide and 120 meters long. The impostor shown in the light polygon is about 700 meters long and placed 200 meters from the view cell.

map automatically would warrant further investigation). The memory requirements are listed without applying the variance-based reduction of textures to single colors.

One of the strong points of our representation is high rendering quality in the presence of detailed geometry. Fig. 7 (see also appendix) shows the filtering of many thin tree branches in front of a moving specular highlight. Fig. 6 (see also appendix) demonstrates correct antialiasing even for extreme viewing angles on building fronts.

Fig. 8 (appendix, bottom) shows how a point-based impostor can be used to improve the rendering quality of the far field in an urban scene which contains several polygons per pixel. It should be noted that the improvement over geometry is even more noticeable when moving the viewpoint. Furthermore, the impostor in Fig. 8 (bottom) replaces a geometric model of about 95,000 vertices, but consists only of about 30,000 points. This shows that the impostor not only improves the rendering quality of the far field, but also reduces the rendering load on the graphics hardware. Fig. 5 shows the placement of this impostor and its view cell in the city model and gives an impression of the typical relative sizes of view cells and impostors.

In the current experimental system, view cells are formed from street segments, and impostors placed at the ends of street segments, in a fashion similar to previous impostor systems [8, 26]. Our test scene is 4 square kilometers large and consists of

2.1 million polygons. After the application of a conservative region-visibility algorithm [27], we identified view cells with too many visible polygons [2]. A rough estimate of the memory requirements using a brute force impostor placement strategy results in about 165 MB used for 437 impostors. Note, however, that the development of good impostor placement strategies is not straightforward and subject to ongoing research. Our future work also aims at reducing memory requirements by sharing information between neighboring view cells, and finding guarantees to ensure a minimum frame rate.

7 Conclusions

We have introduced point-based impostors, a new high-quality image-based representation for real-time visualization.

The value of this representation lies in the separation between the geometric sampling problem and the representation of appearance. Sampling is performed by combining layered depth images to obtain proper coverage of the image for the entire view cell. Based on the mathematical analysis of point-based models, we compute the rendering parameters using Monte Carlo integration, eliminating most of the aliasing artifacts. Rendering information is compactly encoded and can be rendered on contemporary hardware. Point-based impostors show great promise for all situations in which a geometrically very complex model constitutes the far field, such as in urban walkthroughs of detailed models. The rendering times indicate that this representation is applicable to real-time visualization applications, where frame times above 60 Hz are required.

Acknowledgements

This research was supported by the EU Training and Mobility of Researchers network (TMR FMRX-CT96-0036) “Platform for Animation and Virtual Reality” and by the Austrian Science Fund (FWF) contract no. P13867-INF.

References

1. Daniel Aliaga, Jon Cohen, Andrew Wilson, Eric Baker, Hansong Zhang, Carl Erikson, Keny Hoff, Tom Hudson, Wolfgang Stürzlinger, Rui Bastos, Mary Whitton, Fred Brooks, and Dinesh Manoclia. MMR: An interactive massive model rendering system using geometric and image-based acceleration. In *1999 Symposium on interactive 3D Graphics*, pages 199–206, 1999.
2. Daniel G. Aliaga and Anselmo Lastra. Automatic image placement to provide a guaranteed frame rate. *Computer Graphics*, 33:307–316, 1999.
3. Jin-Xiang Chai, Xin Tong, Shing-Chow Chan, and Heung-Yeung Shum. Plenoptic sampling. In *Siggrapph 2000, Computer Graphics Proceedings*, pages 307–318, 2000.
4. Chun-Fa Chang, Gary Bishop, and Anselmo Lastra. LDI tree: A hierarchical representation for image-based rendering. In *Siggraph 1999, Computer Graphics Proceedings*, pages 291–298. ACM Siggraph, 1999.
5. NVIDIA Corporation. Nv_vertex_program extension specification, 2000. available at http://www.nvidia.com/Marketing/Developer/DevRel.nsf/Pro-grammingResourcesFrame.
6. NVIDIA Corporation. Using texture compression in opengl, 2000. available at http://www.nvidia.com/Marketing/Developer/DevRel.nsf/WhitepapersFrame.
7. Lucia Darsa, Bruno Costa Silva, and Amitabh Varshney. Navigating static environments using image-space simplification and morphing. In *1997 Symposium on Interactive 3D Graphics*, pages 25–34, 1997. ISBN 0-89791-884-3.

8. Xavier Decoret, François Sillion, Gernot Schaufler, and Julie Dorsey. Multi-layered impostors for accelerated rendering. *Computer Graphics Forum*, 18(3):61–73, 1999. ISSN 1067-7055.
9. Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. In *SIGGRAPH 96 Conference Proceedings*, pages 43–54, 1996. held in New Orleans, Louisiana, 04-09 August 1996.
10. J. P. Grossman and William J. Dally. Point sample rendering. In *Rendering Techniques '98*, pages 181–192, 1998.
11. Marc Levoy and Pat Hanrahan. Light field rendering. In *SIGGRAPH 96 Conference Proceedings*, pages 31–42, 1996. held in New Orleans, Louisiana, 04-09 August 1996.
12. Marc Levoy and Turner Whitted. The use of points as a display primitive. Technical Report TR 85-022, University of Carolina at Chapel Hill, 1985.
13. Dani Lischinski and Ari Rappoport. Image-based rendering for non-diffuse synthetic scenes. In *Rendering Techniques '98*, pages 301–314, 1998.
14. P. Maciel and P. Shirley. Visual navigation of large environments using textured clusters. *SIGGRAPH Symposium on Interactive 3-D Graphics*, pages 95–102, 1995.
15. William R. Mark, Leonard McMillan, and Gary Bishop. Post-rendering 3D warping. In *1997 Symposium on Interactive 3D Graphics*, pages 7–16, 1997. ISBN 0-89791-884-3.
16. Nelson Max. Hierarchical rendering of trees from precomputed multi-layer Z-buffers. In *Eurographics Rendering Workshop 1996*, pages 165–174, 1996. ISBN 3-211-82883-4.
17. Leonard McMillan and Gary Bishop. Plenoptic modeling: An image-based rendering system. In *SIGGRAPH 95 Conference Proceedings*, pages 39–46, 1995. held in Los Angeles, California, 06-11 August 1995.
18. Alexandre Meyer and Fabrice Neyret. Interactive volumetric textures. In *Rendering Techniques '98*, pages 157–168, 1998.
19. Gavin Miller, Steven Rubin, and Dulce Ponceleon. Lazy decompression of surface light fields for precomputed global illumination. In *Rendering Techniques '98*, pages 281–292, 1998.
20. Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar, and Markus Gross. Surfels: Surface elements as rendering primitives. In *Siggraph 2000, Computer Graphics Proceedings*, pages 335–342, 2000.
21. Szymon Rusinkiewicz and Marc Levoy. QSplat: A multiresolution point rendering system for large meshes. In *Siggraph 2000, Computer Graphics Proceedings*, pages 343–352, 2000.
22. G. Schaufler and W. Stürzlinger. A three-dimensional image cache for virtual reality. In *Proceedings of EUROGRAPHICS'96*, 1996.
23. Gernot Schaufler. Per-object image warping with layered impostors. In *Rendering Techniques '98*, pages 145–156, 1998.
24. Jonathan Shade, Dani Lischinski, David Salesin, Tony DeRose, and John Snyder. Hierarchical image caching for accelerated walkthroughs of complex environments. In *SIGGRAPH 96 Conference Proceedings*, pages 75–82, 1996. held in New Orleans, Louisiana, 04-09 August 1996.
25. Jonathan W. Shade, Steven J. Gortler, Li-wei He, and Richard Szeliski. Layered depth images. In *SIGGRAPH 98 Conference Proceedings*, pages 231–242, 1998. ISBN 0-89791-999-8.
26. François Sillion, G. Drettakis, and B. Bodelet. Efficient impostor manipulationfor real-time visualization of urban scenery. *Computer Graphics Forum*, 16(3):207–218, 1997. Proceedings of Eurographics '97. ISSN 1067-7055.
27. Peter Wonka, Michael Wimmer, and Dieter Schmalstieg. Visibility preprocessing with occluder fusion for urban walkthroughs. In *Rendering Techniques 2000*, pages 71–82, 2000.
28. Daniel N. Wood, Daniel I. Azuma, Ken Aldinger, Brian Curless, Tom Duchamp, David H. Salesin, and Werner Stuetzle. Surface light fields for 3D photography. In *Siggraph 2000, Computer Graphics Proceedings*, pages 287–296, 2000.

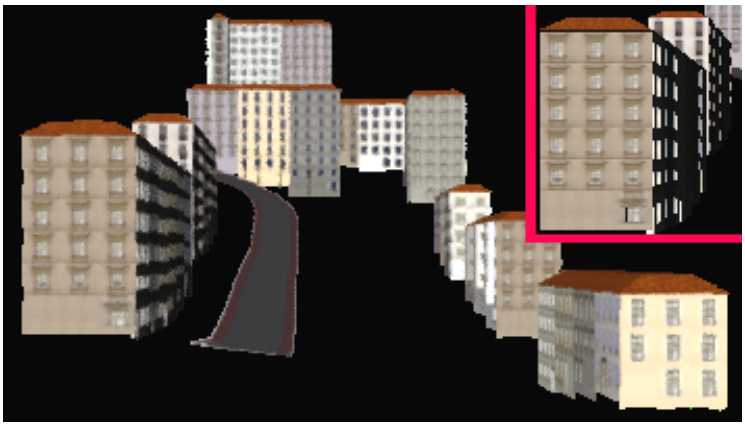

Fig. 6. An impostor for a number of buildings. The inset shows the aliasing that would result if geometry were used.

Fig. 7. Note the correct filtering of the trees against the building front and the specular highlight in the windows for the impostor (top), and severe aliasing in the trees for geometry (bottom).

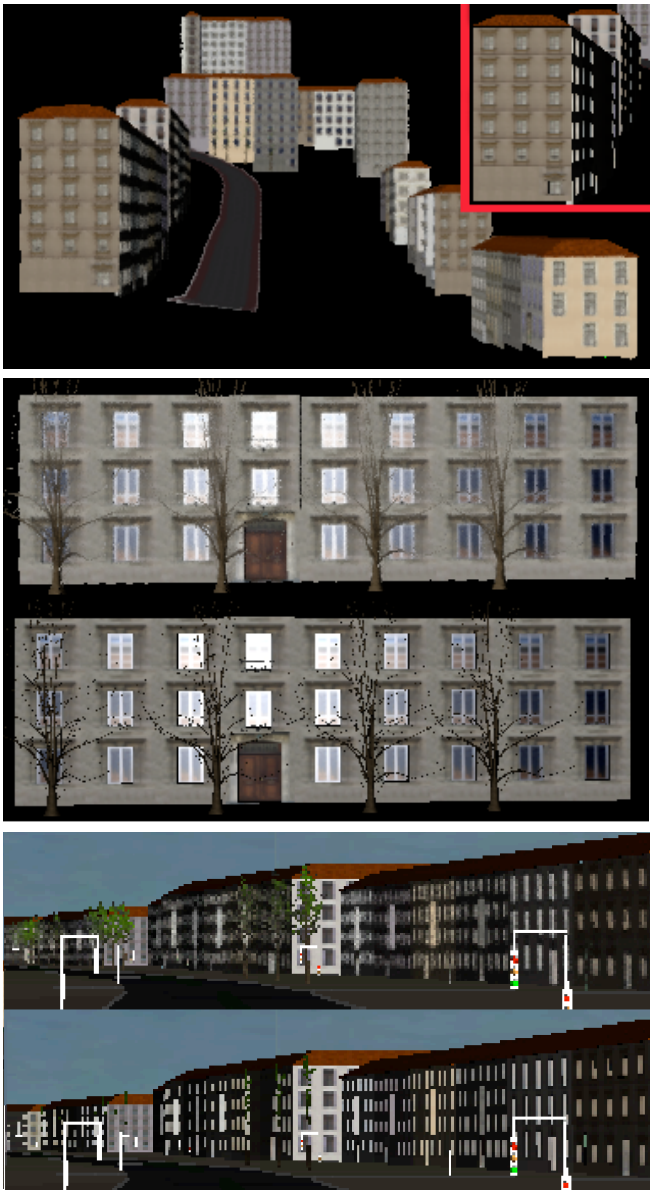

Fig. 8. *Top:* An impostor for a number of buildings (inset: geometry). *Center*: Filtering of trees for impostor (top) against geometry (bottom). *Bottom:* Impostor for a city walkthrough. Note the correct filtering of the impostor (top) compared to geometry (bottom).