View-dependent Far-Field Level of Detail Rendering for Urban Models

Roland Ruiters*

Institute of Computer Science II, University of Bonn, Germany

Abstract

Rendering of terrain meshes has been an active field of research for several years. Real-time, out-of-core terrain rendering systems based on hierarchical level of detail representations created by continuously simplifying the terrain geometry are available today. Although providing good results at close view distances, these techniques are not well suited to the rendering of distant views of urban models. Once the error-threshold exceeds the building size, they are removed by the simplifier and only represented in the terrain texture. This results in a loss of viewing-direction dependent information and thus a change of the appearance of the city. To overcome this problem, we propose the use of surface light fields as a view-dependent, image based level of detail representation for far distances. We compare two compression techniques, the Per Cluster Factorization and the Linear Mode-3 Tensor Approximation, and we show that the surface light fields are more compact, can be rendered at higher frame rates, and show less aliasing artifacts than the geometry they represent.

Keywords: city rendering, surface light fields, level of detail, out-of-core rendering

1 Introduction

The rendering of digital terrain data has been an active field of research with applications in areas like visualization of scientific data and urban planning but also in computer games and navigation systems. Furthermore, in recent years programs allowing the user to interactively view 3D terrain models, obtained from aerial photography and satellite images, have gained high popularity.

These terrain renderers often have to deal with a large range of different scales, possibly reaching from views of the whole planet on the one hand to close-ups of single objects like cars, trees, or buildings, on the other hand. This requires a lot of storage space making it impossible to keep the whole dataset in memory at the same time. Thus, it is necessary to use level of detail techniques to reduce the amount of data that has to be processed for every single view. Furthermore, out-of-core renderers, which are capable of loading the new data in real-time if the point of view changes, are mandatory in this domain.

Most terrain rendering systems in use today are based on static, hierarchical level of detail techniques (HLOD). These create a tree in which each node contains a representation of all of its sub nodes at a coarser level of detail. During rendering the nodes to be rendered are then chosen based on the screen space error. Only a small subset of all nodes, the *graphical working set* [4], is thus needed to render the scene from a certain point of view. This working set usually changes only slowly between two frames. Therefore, it is possible to keep the working set within the RAM and only load missing nodes from secondary storage. Using this technique, interactive rendering of scenes of nearly arbitrary size is possible as long as all nodes can be sufficiently simplified. This is usually done by geometric simplification and subsampling of the textures.

However, in the context of city rendering this approach is no longer feasible for distant points of view. Once each building is represented by a box, further simplification of individual buildings is not possible any longer as a box is already the simplest reasonable representation for a building. When the error threshold finally exceeds the size of the buildings, they are removed all together.

When the geometry is removed, the appearance of the city can change considerably, though. On the other hand, when it is kept, strong aliasing problems occur.

In contrast to other objects commonly drawn by terrain renderers, for instance mountains or vegetation, the visual impression of buildings is strongly dependent on the viewing direction. From above, primarily the roofs of the buildings are seen, but from points of view closer to the ground the colors of the facades dominate the appearance. This effect is lost if the geometry of the buildings is removed and thus, even if the individual buildings are smaller than a pixel, the visual impression of the city can be changed considerably by this simplification step.

Further simplification is not only necessary to maintain scalability and thus allow for out-of-core rendering but also to avoid aliasing. When a view of the whole city is drawn, each building only occupies a few pixels on the screen, and thus the individual triangles are often smaller than one pixel. Furthermore, the contrast between the facades, which often have bright colors, and the dark colors of the roofs is very high. Together, this causes severe aliasing artifacts, which even the full-scene anti-aliasing of-

*ruiters@cs.uni-bonn.de

fered by modern 3D hardware cannot completely remove.

To avoid these problems, a level of detail representation which on the one hand is capable of reproducing the viewing direction dependent effects but on the other hand is also compact and can be rendered fast enough to be useful for out-of-core terrain rendering systems is needed. We propose the use *surface light fields* (SLFs) in this context.

A surface light field is a texture which stores the color in dependence on the viewing-direction. SLFs are already commonly used for the rendering of materials, but they can also be used for rendering terrain as both problems are related. The appearance of a material is strongly influenced by a thin layer of high geometric complexity. For example, rendering fabric is prohibitively expensive if every single thread of the material is rendered. Still, the geometry cannot be ignored becuase the viewing-direction dependent effects caused by this meso-structure give the fabric its distinct appearance, which simple texture mapping can not reproduce. Rendering of cities is nearly the same problem, only on a much larger scale. It is very expensive to render geometry for each house, but on the other hand the buildings cannot be ignored or rendered as texture.

The viewing direction dependent effects of the buildings can be represented in the terrain texture and thus their geometry has not to be rendered any more. For this, terrain textures which contain the buildings from different viewing-directions are created in a preprocessing step by projecting the buildings onto the terrain. From these textures a surface light field is generated. This SLF is then used during runtime to render the terrain with a texture, which contains the buildings as they would appear to a viewer at the current position.

Since the amount of storage needed for the surface light fields is rather high, they have to be stored on the graphics card in compressed form. Therefore, compression techniques which on the one hand achieve high compression ratios but on the other hand allow for real-time rendering are necessary. We have adapted two techniques, the *Per Cluster Factorization* (PCF) [17] and the *Linear Mode-3 Tensor Approximation* (LTA) [10], which both originally have been developed for the compression of *bidirectional texture functions* (BTFs), which also store light-dependent information, to surface light fields and we compare their performance in this context.

Our contribution is the development of the techniques necessary to use surface light fields as a level of detail representation in a terrain renderer. For this, we create the surface light fields from the geometry, adapt two BTF compression techniques to surface light fields and compare their compression performance. Then, the surface light fields are integrated into an existing hierarchical level of detail terrain renderer, where they are used to represent the distant points of view. Finally we evaluate the performance improvement obtained with surface light fields.

2 Related Work

The concept of hierarchical level of detail representations was introduced by James H. Clark in [4]. Since then, a lot of different HLOD techniques have been proposed. These use a number of different representations for the levels of detail, different hierarchical structures, and differ in the strategy used for fetching new data. A high level overview on level of detail techniques for meshes is given in [11], and in [8] the use of hierarchical level of detail is explained in more detail.

In this paper, the SCARPED terrain renderer described in [20] is used. This renderer is based on a static, out-of core, hierarchical level of detail system, which uses a quadtree to represent the scene. The lowest nodes of the quadtree contain the terrain geometry, that has been created from elevation data. The other nodes are generated by successively simplifying geometry using the Hausdorff distance to guarantee an upper bound for the error. This threshold is doubled for each level of the quadtree. The buildings have been generated from land register data and are stored in a separate quadtree.

It is possible to mix different representations within the level of detail hierarchy, and thus for example image based representation can be used for distant objects. A lot of different level-of-detail representations exist, which replace the objects with a proxy that is rendered using an image based representation of the original geometry. An overview of these techniques is given in [12].

However, most image based LOD techniques do not store viewing-direction dependent information and thus suffer from similar problems as rendering geometry. In [21] the use of billboards with viewing-direction dependent textures stored using a modified video codec has been proposed, in [22] and [9] point clouds with viewing-direction dependent colors were used and in [15] billboard clouds with BTF textures. A technique similar to the one described in this paper has been introduced in [1], which uses viewing-direction dependent textures in a terrain renderer. In contrast to the surface light fields, these textures are generated at run-time and have to be updated every time the point of view changes. A recent work in the context of city rendering are Blockmaps [3], which use a volumetric representation of the buildings which is rendered with ray-casting.

The use of view-dependent texture mapping has been proposed in [7]. Whereas this technique works by projecting images from different points of view onto the geometry, surface light fields [16] instead store one texture with viewing direction dependent information for each texel. In [16] a DCT based block coding scheme was used to compress the surface light field data, but also techniques based on vector quantization [23] and on the approximation of the surface light field with a lower dimensional subspace [23, 2, 5] have been proposed.

A representation even more general than SLFs are BTFs which also store the dependence on the light direction.

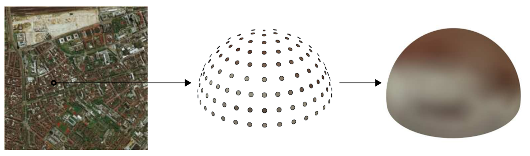


Figure 1: Sampled (center) and interpolated (right) lumisphere corresponding to a point on the surface light field.

However, surface light fields are sufficient in our case since the terrain engine we used renders the scene without dynamic lighting. Several techniques for the compression of BTFs can also be used for SLFs. An overview is given in [18]. In this paper the Per Cluster Factorization (PCF) [17] and the Linear Mode-3 Tensor Approximation (LTA) [10] are used.

3 Surface Light Fields

A surface light field is a texture map that stores the appearance of an object in dependence on the viewing direction. For each texel of the texture, thus a *lumisphere* [23] is stored, which contains for each of a set of discrete viewing directions $\mathbf{d}_1, \ldots, \mathbf{d}_n$ the color that should be visible from this direction. The surface light field can be interpreted as a collection of lumispheres

$$L = \{\mathbf{l}_m \in \mathbb{R}^{3n}\}_{m \in M \subset \mathbb{N}^2}$$

where each $\mathbf{l}_i$ is a vector containing a RGB sample for each viewing-direction and M is a grid of texel positions. During rendering, intermediate viewing-directions are interpolated bilinearly between the four nearest stored samples. Figure 1 shows an illustration of a lumisphere.

The viewing directions are all parameterized relative to the bounding box of the terrain quadtree cell and are chosen on a hemisphere over the cell in such a way that they later can be stored in a texture map using a parabolic parameterization. For the PCF compressed SLFs, 11×11 sized texture patches were used, resulting in 97 valid viewing directions. The remaining points of the texture are not used for this parameterization. In contrast, for the LTA compressed SLFs the use of more viewing directions is cheaper, as they have to be stored only once and not for each cluseter. Therefore, here 16×16 sized patches, parameterized as described in [10], were used.

4 Creation

The surface light fields are created in a preprocessing step for those quadtree cells that contain buildings and belong to a sufficiently coarse level of detail. For this, first the terrain cell is rendered together with all buildings within the cell and the eight adjacent cells from each of the viewing directions $\mathbf{d}_1, \ldots, \mathbf{d}_n$. The resulting images are then projected onto the terrain geometry to create a texture containing the buildings as they would appear if seen from the direction $\mathbf{d}_i$.

It is necessary to render the eight adjacent cells together with the cell itself because it can happen that from certain viewing directions buildings are projected into adjacent cells and thus have to be represented in their textures. For the rendering, an orthographic projection is used since the same viewing direction has to be sampled for all pixels. This projection is scaled along the vertical axis to use the available rendering area as efficient as possible, which is especially necessary for very shallow viewing angles. In these cases, the cell would otherwise be projected into a rectangle of very low height, and thus it would not be possible to sample at least one point on the terrain surface for each texel of the texture.

To project the buildings into the terrain texture, the same cell is rendered again but this time without buildings. Then, the depth buffer is read back and used to calculate for each pixel of the rendered view the 3D position of the point on the terrain geometry that would be visible when no buildings were rendered. From this position, the texture coordinates of the point are determined and the color of the pixel is stored for the texel corresponding to these coordinates.

For each texel of the created texture all samples are averaged to anti-alias the resulting image. Because of the scaled projection and the fact that the views are rendered at eight times the texture resolution, texels that were not occluded by the terrain will always receive enough samples. On the other hand, if texels have been occluded, it is necessary to use the colors from neighboring texels to fill the resulting holes.

5 Compression

When stored uncompressed, the 128×128 surface light field for one terrain cell, sampled from 97 viewing directions, would need about 4.7 MB, making it prohibitively expensive to store. Therefore, efficient compression techniques which reduce the size of each cell considerably but at the same time allow for real time rendering are needed.

5.1 Per Cluster Factorization

Several works [23, 2, 5] compress surface light fields by finding a lower dimensional affine linear subspace that approximates the data best in a least squares sense, and then representing the data in this subspace. Further improvements of the compression ratio can be achieved by performing an additional clustering step and then determining an independent affine subspace for each cluster, instead of approximating all lumispheres with one subspace. This technique was introduced in [13] for machine learning purposes. Its application to compression was suggested inde-

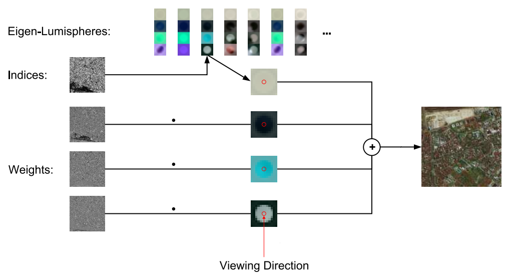


Figure 2: The operations performed by the PCF Shader

pendently in [19], where it is called clustered PCA and used to compress precomputed radiance transfer matrices, and in [17], where the term local PCA is used and the technique is applied to the compression of BTFs.

The local PCA algorithm is a modified version of the k-means algorithm. It starts by choosing k random, c-dimensional, affine linear subspaces, each defined by an origin $\mathbf{r}_j$ and a set of c basis-vectors $\mathbf{e}_{i,j}$. Each vector is assigned to the cluster with the smallest squared reconstruction error:

$$\varepsilon(\mathbf{x}, j) = \left\| \mathbf{x} - \mathbf{r}_j - \sum_{i=0}^{c} < \mathbf{x} - \mathbf{r}_j, \mathbf{e}_{i,j} > \mathbf{e}_{i,j} \right\|^2$$

Then, for each cluster the center $\mathbf{r}_j$ is recalculated as the mean of the vectors belonging to the cluster, and a set of new basis-vectors $\mathbf{e}_{i,j}$ is determined by performing a PCA on these vectors. This is iterated until the change in reconstruction error between two iterations falls below a given threshold.

For the compression of surface light fields, the local PCA is performed with the set L of all lumispheres as input. Each lumisphere $\mathbf{l}_m$ can then be represented by a cluster index j_m and c weights $w_{i,m}$. Analogous to the terms Eigen-Texture and Eigen-BRDF, we will use the term Eigen-Lumisphere for the basis-vectors $\mathbf{e}_{i,j}$.

To improve the perceived visual quality of the PCF compressed surface light fields, the elements of the lumisphere vectors can be weighted before performing the local PCA. This is done by multiplying the colors for each viewing direction by the cosine of the angle between the normal and the viewing direction before the local PCA and dividing the elements of the basis vectors by this value afterwards. This weighting was chosen because it is proportional to the size of the projection of the cell onto the screen.

We used three basis vectors to represent the lumisphere. This way, during rendering the indices and weights can be stored in one RGBA texture map. The cluster centers and the Eigen-Lumisphere are arranged in 11×11 blocks, which are all stored together in a second texture map. Each block contains one hemisphere of viewing directions. This way the bilinear texture filtering of the graphics hardware can be used to interpolate viewing directions.

For real-time rendering of the PCA compressed textures, GLSL shaders are used. In Figure 2 the operations performed by the shader are illustrated. In the vertex shader, the viewing direction $\mathbf{d}$ in the local coordinate system of the terrain cell is calculated and passed to the fragment shader in a varying variable.

The fragment shader then reconstructs the color $\mathbf{C}(\mathbf{x}, \mathbf{d})$ of the texel at the texture coordinates $\mathbf{x}$ for the viewing direction $\mathbf{d}$. First, the index $j(\mathbf{x})$ and the weights $w_i(\mathbf{x})$ are fetched. Then, the direction is renormalized, and mapped into the Eigen-Lumisphere texture. These texture coordinates are used to fetch the value of the cluster center and the Eigen-Lumispheres at the position $\mathbf{x}$ that corresponds to the viewing-direction $\mathbf{d}$. If $\mathbf{R}_{j(\mathbf{x})}(\mathbf{d})$ is the color of the cluster center and $\mathbf{E}_{i,j(\mathbf{x})}(\mathbf{d})$ are the colors of the Eigen-Lumispheres, the following sum has to be evaluated to reconstruct the color of the texel:

$$\mathbf{C}(\mathbf{x}, \mathbf{d}) = \mathbf{R}_{j(\mathbf{x})}(\mathbf{d}) + \sum_{i=1}^{c} w_i(\mathbf{x}) \cdot \mathbf{E}_{i,j(\mathbf{x})}(\mathbf{d})$$

For each additional Eigen-Lumisphere used during compression, thus an additional texture lookup is needed. In contrast, increasing the number of clusters does not increase the number of texture lookups. Therefore, it is desirable to use a high number of clusters. As an 8-bit texture is used to store the cluster index, it is possible to use up to 256 clusters, but storing the Eigen-Lumispheres for such a high number of clusters for each terrain cell requires too much storage. Instead, the surface light fields of nine terrain cells are grouped and use together one set of 256 shared clusters, which is stored in one 512×512 sized DXT5 compressed texture. If the alpha channel is additionally used, it is possible to store four Lumispheres in the space otherwise needed for three Lumispheres. This way, the number of texture lookups can be reduced further. With this representation, the 128×128 surface light field for one terrain cell requires about 90 KB.

The use of clustering to increase the compression performance has one serious disadvantage, though. The texture filtering hardware cannot be used any more on the clustered textures as it would interpolate between weights belonging to different clusters. Therefore, the filtering has to be done in the fragment shader, instead. For this, a simple 5-tap anisotropic filtering is used and thus the lumisphere color has to be evaluated five times for each rendered pixel, increasing the number of texture accesses needed to 20. The rendering of PCF compressed surface light fields is thus rather expensive.

5.2 Linear Mode-3 tensor Approximation

A second technique which avoids the clustering and thus can utilize the filtering hardware is the Linear Mode-3 Tensor Approximation introduced in [10]. Here the lumispheres are not represented by a matrix but instead by a tensor $\mathscr{P} \in \mathbb{R}^{k \times t \times v}$, where k is the number of color channel, t is the number of texels and v is the number of viewing directions. To compress this tensor, it is approximated by a sum of rank-1 tensors:

$$\mathscr{P} = \sum_{j=1}^{r} \sigma_j \cdot \mathbf{c}_j \circ \mathbf{i}_j \circ \mathbf{w}_j \tag{1}$$

where $\{\sigma_j\}_{j=1\ldots r}$ is a set of scalar weighting coefficients, $\{\mathbf{c}_j\}_{j=1\ldots r}$ a set of base colors, $\{\mathbf{i}_j\}_{j=1\ldots r}$ a set of weights textures and $\{\mathbf{w}_j\}_{j=1\ldots r}$ a set of scalar Eigen-Lumispheres. This sum is called *Tensor Product Expansion* (TPE). The first summand is determined by finding the best rank-1 approximation of $\mathscr{P}$ using an alternating least squares algorithm [14, 6]. This approximation is then subtracted from $\mathscr{P}$ to calculate the remainder, from which then the next summand can be calculated. This is iterated until the first c rank-1 tensors have been determined.

To achieve a better perceptual quality, the visual model described in [10], is used. This performs, additionally to the viewing direction weighting described above, a transform into the YC_bC_r color space to weight the luminance and chroma channels independently and a wavelet transform to weight components of different frequency according to the contrast sensitivity function of the human visual system. This way, a better visual quality can be achieved with the same number of Eigen-Lumispheres.

Since, in contrast to the PCF, no clustering is used for the LTA, the anisotropic filtering can be performed in the texture units of the graphics hardware for each of the weight textures independently. Therefore, the shader used to render LTA compressed surface light fields is far simpler as it neither needs to look up the cluster indices nor has to perform texture filtering.

Assuming $w_i(\mathbf{x})$ is the weighting coefficient at position $\mathbf{x}$ on the surface light field corresponding to the i^{th} Eigen-Lumisphere, $e_i(\mathbf{d})$ is the scalar value of the i^{th} Eigen-Lumisphere in viewing direction $\mathbf{d}$ and $\mathbf{c}_i$ is its base color premultiplied with the coefficient σ_j, the shader has to calculate the following sum:

$$C(\mathbf{x},\mathbf{v}) = \sum_{i=1}^{c} w_i(x) \cdot e_i(\mathbf{v}) \cdot \mathbf{c}_i$$

To calculate this efficiently, four weights textures are packed in the RGBA channels of one texture. Thus, when using 24 Eigen-Lumispheres, 6 RGBA weight textures are needed. Similarly, four scalar Eigen-Lumispheres at a time are combined into the RGBA channels of a second texture. The base colors of four lumispheres are stored in three `vec4` in the constant registers of the shader, each containing the values of one color channel. Both the weights- and the lumispheres textures use 8 bit precision and thus require scaling and shifting of the values into the range $[0,1]$. Since the base colors are stored in float registers, it is possible to divide these by the scale factors, and thus it is not necessary to reverse the scaling of the weight and lumisphere textures in the shader.

The three color channels are processed independently because this way four terms of the sum can be evaluated at a time. First, the four weight values and the four scalar Eigen-Lumisphere values corresponding to the position on the surface and the viewing direction are fetched and then multiplied component-wise. Then, for each of the color channel the scalar product between this value and the register containing the four base-color values is calculated and added to the final color. This way, only two texture lookups, one component-wise multiplication, three dot products, and six additions are needed to calculate four terms of the sum[1]. Thus, 12 texture lookups are needed when 24 Eigen-Lumispheres are used. In contrast, the PCF compressed surface light fields needs 20 texture lookups when 3 components are used and texture filtering is performed in the shader. In Figure 3 the operations performed by the shader are illustrated.

The rendering performance can be further improved by assuming a constant viewing direction for the whole cell because $E_i(\mathbf{d}) \cdot \mathbf{c}_i$ can then be evaluated on the CPU and the result stored in the constant registers of the shader. This reduces the number of texture lookups to six, and avoids the component-wise multiplication and two additions. The assumption of constant viewing directions for all texels within one cell is only a good approximation if the viewer is far enough away from the cell, though. However, surface light fields are only used for levels of detail which are seen from distant points of view. In the dataset used for this paper, the angular error in the viewing direction does not exceed 4.5°.

The weights textures are further compressed by performing a vector quantization (VQ), using a codebook shared between four cells. This vector quantization is de-

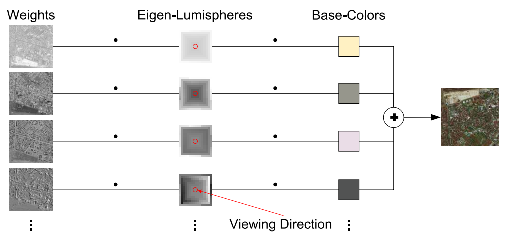


Figure 3: The operations performed by the LTA Shader.

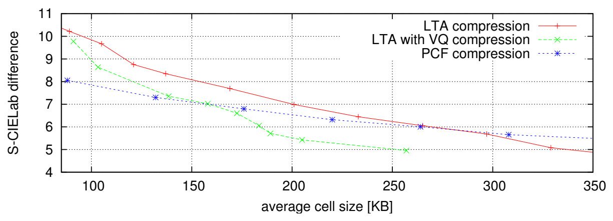


Figure 4: Comparison of the S-CIELab difference in units of just noticeable difference (JND) between PCF and LTA in dependence on the size of the compressed surface light field.

[1]One of the additions is necessary to calculate the new texture coordinates and two are used to shift the values fetched from the textures into the range $[0.5, 0.5]$

compressed on the CPU before uploading the cells to the GPU and thus only reduces IO times and quadtree cache efficiency but not the amount of graphics memory needed. However, decompressing the weights textures on the CPU has the advantage that the texture filtering hardware can be fully used and therefore LTA compressed SLFs render faster than PCF compressed ones. When the first 24 terms of the the TPE are kept and the error threshold during vector quantization is chosen as $\varepsilon = 0.0125$, about 140 KB are needed for each cell of the quadtree.

5.3 Comparison of Compression Results

It is difficult to measure the quality of images which were compressed using a perceptive visual model objectively since it is necessary to use a model of the human visual system for the measurement itself. In [10], a system which uses the S-CIELAB quality metric [24] to assess BTF compression techniques is proposed. In Figure 4, the reconstruction errors for PCF and the LTA, as measured by this system, are compared.

For the measurement a 21 inch display with a resolution of 1280×1024 and a distance from the viewer to the screen of about 60 cm was assumed. Furthermore, the measurement was done for a one-to-one mapping between texels and pixels.

For the measurements, four sample terrain cells from the dataset, which all contained a high number of buildings, were used. The PCF and LTA compression was done with different numbers of components. The results for LTA with Vector Quantization were obtained by first performing LTA compression with 24 components, and then using VQ with different error thresholds. For the LTA, a codebook shared among the four cells was used, and the PCF compression was done with 32 shared Clusters.

The measurements have shown that the additional vector quantization does in fact offer a better visual quality at the same compressed size than can be obtained with the LTA alone, and that for very high compression ratios, the PCF performs better than both LTA based techniques.

6 Rendering

The integration into the terrain renderer is quite similar for both surface light field compression techniques. The surface light fields are stored in the quadtree, replacing the original textures. During rendering, the SLFs are then used to texture the terrain and the buildings are no longer rendered at all.

When switching from buildings rendered with SLFs to geometry, popping artifacts are visible. This is caused, on the one hand, by the quite high compression used on the cells. On the other hand, even if an uncompressed surface light field created for exactly the correct viewing-direction is used, still a difference is visible as the surface light field is first resampled to 128×128 texels and then rendered using texture filtering, which both creates a blurred image.

Therefore, it is necessary to use blending to hide the transition from the user. For this, firstly the buildings are faded-in using alpha blending, and secondly the surface light fields are crossfaded with the textures of the next finer level of detail. These two operations are both controlled by the pixel error ε of the cell. In the terrain engine, this error is calculated during the traversal of the quadtree for each cell, and if it is higher than the threshold ε_{max} chosen by the user, a finer level of detail is displayed.

However, instead of simply switching between levels of detail, now the transition is done over a certain range of pixel errors. When the pixel error reaches $0.85\varepsilon_{max}$ the buildings start to fade in. This is done by linearly increasing their alpha component with the pixel error until it reaches $1.1\varepsilon_{max}$. Similarly, the crossfading of the surface light fields starts at a pixel error of ε_{max} and ends at an error of $1.1\varepsilon_{max}$.

During the crossfading of the textures, the four cells at the finer level of detail are rendered instead of the cell containing the surface light field, but these cells are drawn with the corresponding quarter of the texture of their parent cell linearly interpolated with their own textures. For this interpolation, the pixel error of the parent cell and not of the four cells themselves has to be used.

This is necessary to avoid popping artifacts that would occur otherwise. When the parent cell reaches a pixel error of ε_{max}, the four child cells are rendered instead. These four cells have different pixel errors as their centers are at different distances to the viewer. Thus, it can happen that the cell nearest to the viewer has a pixel error which is smaller than the threshold and the cell's own texture is displayed without any fading. This problem can be avoided if the pixel error of the parent cell is used instead, and all four cells are thus blended using the same error.

7 Results

For the evaluation of the techniques described in this paper, a dataset of the German city Munich was used. This dataset contained about $50,000$ buildings, which were represented by more than 1.5 million textured triangles, requiring 170 MB of storage in total on the finest level of detail. For this dataset, surface light fields were generated for terrain cells with an extent of about 1.6 km, of which each requires about 1.6 MB of storage.

In Table 1 the amount of data and the time needed to load an initial view in which the whole city is represented by surface light fields is shown for the different representations. The column *size* contains the total amount of data that has to be loaded for this view, including both geometry and textures. The time necessary for this is given in the column *I/O time* and in *instantiation time* the time needed to decode the loaded data and upload these to the graphics card is shown.

The use of surface light fields reduces the amount of data that has to be loaded considerably and thus also reduces the time necessary to load the data. In comparison to the I/O time, the time needed for decoding the vector quantized textures of the LTA compressed SLFs is negligible.

In Figure 6, the frame rates during a flight from a distant point of view from which the whole visible scene is rendered with surface light fields to a point from which nearly all buildings are drawn as geometry is shown. For all three techniques, all frames were rendered from exactly the same points of view. The measurements were performed on a computer with an Intel Core 2 Duo T7100 processor, 4 GB RAM, and a NVIDIA GeForce 8600M GT graphics card.

Usually, the terrain renderer uses asynchronous I/O and thus has to render a representation at a lower level of detail until the data for the current LOD have been loaded. This way, smoother rendering is achieved, but if the data cannot be fetched fast enough, the visual quality degrades. If asynchronous I/O is used, the frame rates measured for the different techniques cannot be compared, though, as it is possible that for the same point of view different LODs are chosen. Therefore, we used synchronous I/O for these measurements. However, this results in strongly varying framerates and thus the framerates shown here are averaged over 25 frames at a time to increase the readability.

The use of surface light fields increases the framerate, but during the transition of the two representations the framerate can actually drop below the framerate obtained by using only geometry because both representations have to be rendered together.

We also compared our approach based on surface light fields to a technique similar to the Far Voxels described in [9]. Here, the buildings are represented by a point cloud with viewing direction dependent colors. We used the PCF compression to store the Lumispheres for each point in this cloud. When the fact that certain viewing directions will never be visible is considered during compression, a reasonably compact representation (see Table 1) and fast rendering (see Figure 6) of the Far Voxels is possible.

However, during our experiments, Far Voxels suffered from serious sampling artifacts. The points of the cloud cannot be rendered with sub-pixel accuracy and thus the problems are even worse than when rendering geometry. A border of 1-pixel width is created by this effect which results in the occlusion of fine structures, like the facades of other buildings (See Figure 5). Therefore, it would actually be necessary to store viewing direction dependent coverage information for each voxel, and then the points would have to be rendered using alpha-blending. However, this would increase the amount of storage needed and requires different rendering techniques to avoid sorting the point cloud by depth.

Representation	Size	I/O Time	Inst. Time
Geometry	60.1 MB	1.84 s	0.155 s
PCF	1.7 MB	0.14 s	0.004 s
LTA with VQ	2.6 MB	0.20 s	0.002 s
Far Voxel	3.8 MB	0.346 s	0.026 s

Table 1: Size of the cells needed for an initial view of the city and the time necessary to fetch them from hard disk

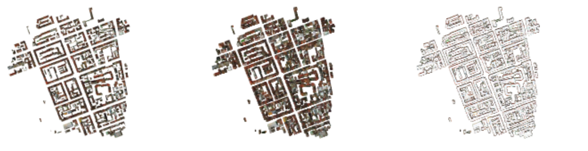

(a) Original (b) Point cloud (c) Difference image

Figure 5: Problems of point cloud rendering

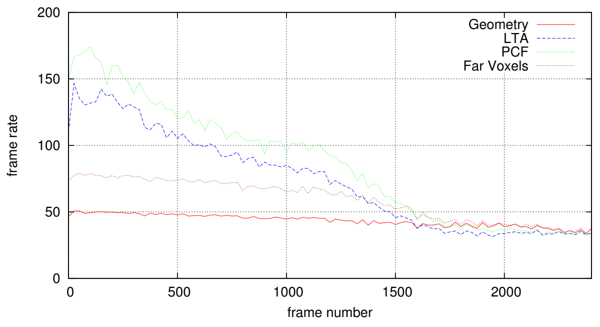


Figure 6: Comparison of the frame rates for the different rendering techniques.

In the color plates, close-ups of the same buildings from different points of view are compared. For example, the parallel buildings clearly show view-dependent effects, if rendered with surface light fields. When seen from above, only the roofs are visible, but when seen from different sides, the corresponding facades become visible. Furthermore, views of the whole city rendered with geometry and with SLFs are compared. The images rendered with SLFs are less sharp, but also show less aliasing artifacts than the images rendered with geometry.

8 Conclusion

The use of surface light fields allows to render distant views of cities with higher frame rates than possible by rendering geometry. The amount of data that has to be loaded is reduced considerably and scales linearly with the texture size. Therefore, SLFs are well suited for out-of-core renderers. Furthermore, most aliasing artifacts, geometry suffers from, can be avoided because texture filtering can be used.

Both compression techniques have advantages and disadvantages. PCF compression should be preferred if high compression ratios are more important, like for example if the cells are transmitted over the Internet, and LTA compression is to be preferred if a high rendering performance is necessary, especially when slower graphics hardware should be supported.

The surface light fields have certain limitations, though. The use of surface light fields results in a loss of details because of the compression. The technique can only be used if the distance between the viewer and the object is very high. As the viewer approaches the objects, the influence of parallax effects gets stronger and the surface light fields can no longer be compressed efficiently. Similar problems occur for very shallow viewing angles. Furthermore, SLFs can only be used if a geometry on which the surface light fields can be projected is available and only if fixed lighting is used. If dynamic lighting is necessary, BTFs have to be used instead, increasing the amount of data needed for each cell.

It can be concluded that surface light fields are certainly a viable alternative as a level of detail representation for rendering distant views of cities.

9 Acknowledgments

I thank Remote Sensing Solutions GmbH for making the Munich dataset available for this work and Prof. Dr. Reinhard Klein for supervising my diploma thesis.

References

[1] Baoquan Chen, J. Edward Swan II, Eddy Kuo, and Arie Kaufman. Lod-sprite technique for accelerated terrain rendering. In *VIS '99: Proceedings of the conference on Visualization '99*, pages 291–298, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[2] Wei-Chao Chen, Jean-Yves Bouguet, Michael H. Chu, and Radek Grzeszczuk. Light field mapping: efficient representation and hardware rendering of surface light fields. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 447–456, New York, NY, USA, 2002. ACM Press.

[3] Paolo Cignoni, Marco Di Benedetto, Fabio Ganovelli, Enrico Gobbetti, Fabio Marton, and Roberto Scopigno. Ray-casted blockmaps for large urban models streaming and visualization. *Computer Graphics Forum*, **26**(3):405–413, Sept. 2007.

[4] James H. Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, **19**(10):547–554, 1976.

[5] Greg Coombe, Chad Hantak, Anselmo Lastra, and Radek Grzeszczuk. Online construction of surface light fields. In Oliver Deussen, Alexander Keller, Kavita Bala, Philip Dutré, Dieter W. Fellner, and Stephen N. Spencer, editors, *Rendering Techniques*, pages 83–90. Eurographics Association, 2005.

[6] L. De Lathauwer, B. De Moor, and J. Vandewalle. On the Best Rank-1 and Rank-(R, R,..., R) Approximation of Higher-Order Tensors. *SIAM Journal on Matrix Analysis and Applications*, **21**:1324–1342, 2000.

[7] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 11–20, New York, NY, USA, 1996. ACM Press.

[8] Carl Erikson, Dinesh Manocha, and William V. Baxter III. Hlods for faster display of large static and dynamic environments. In *I3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 111–120, New York, NY, USA, 2001. ACM.

[9] Enrico Gobbetti and Fabio Marton. Far voxels: a multiresolution framework for interactive rendering of huge complex 3d models on commodity graphics platforms. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 878–885, New York, NY, USA, 2005. ACM Press.

[10] Michael Guthe, Martin Schneider, Gero Müller, and Reinhard Klein. Btf-cielab: a psychovisual difference measure for quality assessment and compression of btfs, 2008. Submitted to *Computer Graphics Forum*.

[11] Tan Kim Heok and Daut Daman. A review on level of detail. *CGIV '04: Proceedings of the International Conference on Computer Graphics, Imaging and Visualization (CGIV'04)*, pages 70–75, 2004.

[12] Stefan Jeschke, Michael Wimmer, and Werner Purgathofer. Image-based representations for accelerated rendering of complex scenes. In Y. Chrysanthou and M. Magnor, editors, *EUROGRAPHICS 2005 State of the Art Reports*, pages 1–20. EUROGRAPHICS, The Eurographics Association and The Image Synthesis Group, August 2005.

[13] Nandakishore Kambhatla and Todd K. Leen. Dimension reduction by local principal component analysis. *Neural Computation*, **9**(7):1493–1516, 1997.

[14] Peter M. Kroonenberg and Jan de Leeuw. Principal component analysis of three-mode data by means of alternating least squares algorithms. *Psychometrika*, **45**(1):69–97, 1980.

[15] J. Meseth and R. Klein. Memory efficient billboard clouds for btf textured objects. In B. Girod, M. Magnor, and H.-P. Seidel, editors, *Vision, Modeling, and Visualization 2004*, pages 167–174. Akademische Verlagsgesellschaft Aka GmbH, Berlin, November 2004.

[16] Gavin S. P. Miller, Steven M. Rubin, and Dulce B. Ponceleon. Lazy decompression of surface light fields for precomputed global illumination. In George Drettakis and Nelson L. Max, editors, *Proceedings of the 9th Eurographics Workshop on Rendering*, pages 281–292, 1998.

[17] Gero Müller, Jan Meseth, and Reinhard Klein. Compression and real-time rendering of measured btfs using local pca. In T. Ertl, B. Girod, G. Greiner, H. Niemann, H.-P. Seidel, E. Steinbach, and R. Westermann, editors, *Vision, Modeling and Visualisation 2003*, pages 271–280. Akademische Verlagsgesellschaft Aka GmbH, Berlin, November 2003.

[18] Gero Müller, Jan Meseth, Mirko Sattler, Ralf Sarlette, and Reinhard Klein. Acquisition, synthesis and rendering of bidirectional texture functions. In Christophe Schlick and Werner Purgathofer, editors, *Eurographics 2004, State of the Art Reports*, pages 69–94. INRIA and Eurographics Association, September 2004.

[19] Peter-Pike Sloan, Jesse Hall, John Hart, and John Snyder. Clustered principal components for precomputed radiance transfer. *ACM Trans. Graph.*, **22**(3):382–391, 2003.

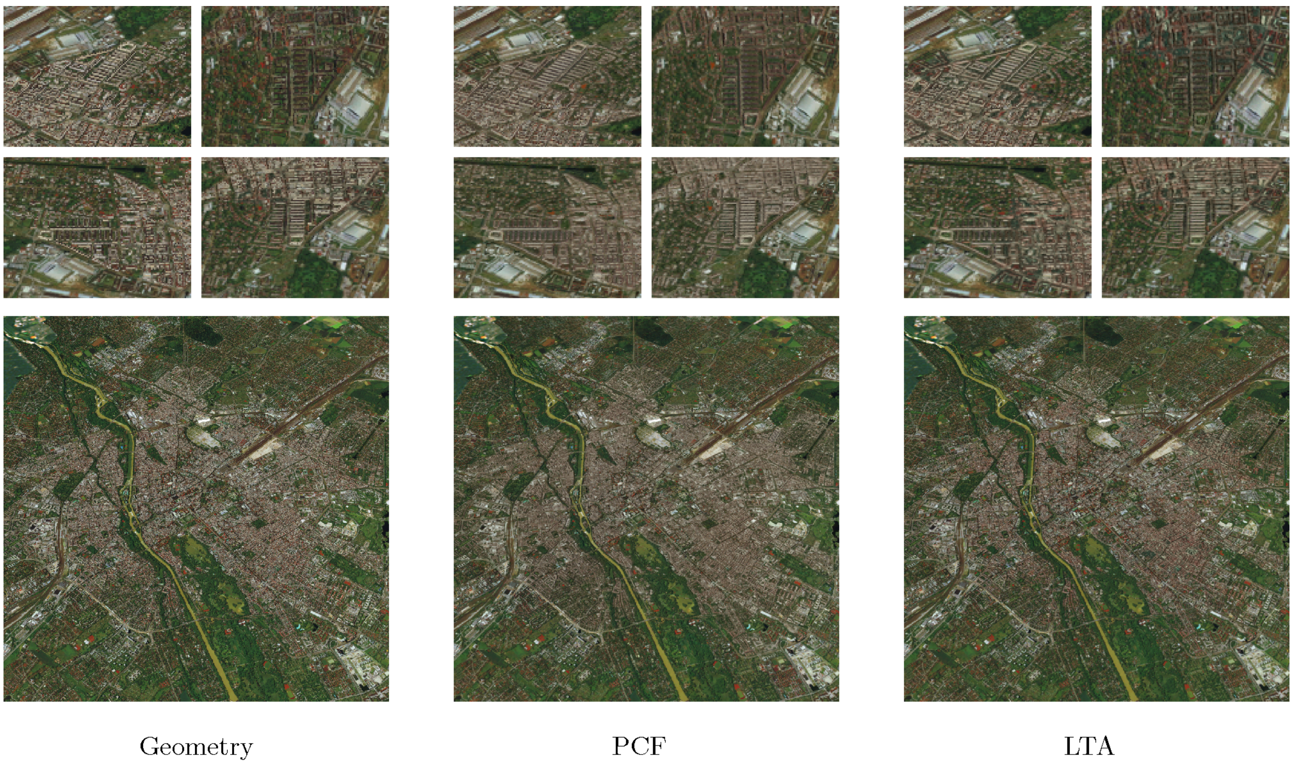
[20] Roland Wahl, Manuel Massing, Patrick Degener, Michael Guthe, and Reinhard Klein. Scalable compression and rendering of textured terrain data. *Journal of WSCG*, **12**(3):521–528, February 2004.

[21] Andrew Wilson, Ketan Mayer-Patel, and Dinesh Manocha. Spatially-encoded far-field representations for interactive walkthroughs. In *MULTIMEDIA '01: Proceedings of the ninth ACM international conference on Multimedia*, pages 348–357, New York, NY, USA, 2001. ACM Press.

[22] Michael Wimmer, Peter Wonka, and François X. Sillion. Point-based impostors for real-time visualization. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, pages 163–176, London, UK, 2001. Springer-Verlag.

[23] 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 Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 287–296. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[24] Xuemei Zhang and Brian A. Wandell. A spatial extension of CIELAB for digital color image reproduction. *Society for Information Display Symposium Technical Digest*, **27**:731–734, 1996.



Comparison of the rendering techniques. For all images 16x full-scene anti-aliasing was used.