Efficient Free Form Light Field Rendering

Hartmut Schirmacher[1], Christian Vogelgsang[2],
Hans-Peter Seidel[1], Günther Greiner[2]

[1] Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85, 66123 Saarbrücken
[2] Computer Graphics Group, Universität Erlangen
Am Weichselgarten 9, 91058 Erlangen

Abstract

We show a simple and efficient way for rendering arbitrary views from so-called *free-form light fields*, employing a convex free form camera surface and a set of arbitrarily oriented camera planes. This way directionally varying real-world imagery can be displayed without intermediate resampling steps, and yet rendering of free form light fields can be performed as efficiently as for two-plane-parameterized light fields using texture mapping graphics hardware. Comparable to sphere-based parameterizations, a single free form light field can represent all possible views of the scene without the need for multiple slabs, and it allows for relatively uniform sampling. Furthermore, we extend the rendering algorithm to account for occlusion in certain input views. We apply our method to synthetic and real-world datasets with and without additional geometric information and compare the resulting rendering performance and quality to two-plane-parameterized light field rendering.

1 Introduction

Light field rendering has been introduced to computer graphics in 1996 [9, 5] and has since then inspired a considerable body of research work. In the original work, light fields are parameterized by pairs of parallel planes, the so-called light slabs. This parameterization is simple and convenient in many ways and has been used to design some very efficient light field rendering algorithms [5, 14, 18, 15, 12].

Alternative parameterizations have been proposed by several authors (cf. Fig. 1) for different reasons such as coherence/compression, sampling uniformity, and for seamless coverage of all possible viewing directions with just a single light field entity instead of multiple slabs [7, 2, 1, 16]. Furthermore, in the case where the *exact* scene geometry is available in the form of polygons and preprocessing time is not an issue, it has turned out to be very convenient to use the surface parameterization directly as part of the light field parameterization [10, 20].

One practical and important problem of light fields is their acquisition, since it is not easy to set up an array of cameras in such way that they match the parameterization, except by using robot gantries [9] or custom-built camera arrays [21, 17]. If a camera does not meet this constraint, the corresponding image data has to be resampled ("rebinned"), which can be very time-consuming and decrease image quality.

Heigl et al. [6] introduced an algorithm that renders directly from an arbitrary image sequence obtained by a hand-held camera, without any intermediate reparameterization. However, their approach uses a view triangulation with varying topology over time, thus resulting in inconsistent reconstruction of viewing rays. Although Heigl et al. [6] mention that their approach can be efficiently implemented using texture mapping, they did not give any performance hints.

In this paper, we propose a *free form* light field parameterization that avoids this problem by restricting the optical centers of the light field cameras to an arbitrary convex free form surface, while the camera image planes can be oriented and placed freely. The convex free form surface provides a consistent triangulation of the camera views that will always reconstruct the same ray from the same original views.

In addition, we discuss and solve the problem of partial occlusion, ocurring when one or more of the "closest" input views do not see the desired part of

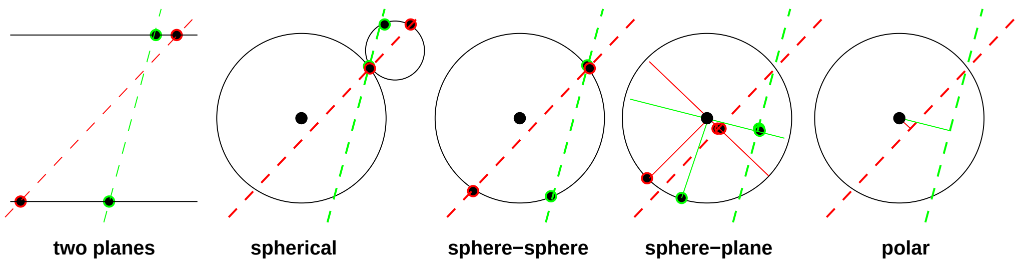

Figure 1: 2-D sketch of light field parameterizations, for two different rays to be parameterized (dashed lines). Please refer to Sec. 2 for further explanation.

the scene.

The rendering of free form light fields uses the original images directly and involves a single hardware-filtered resampling step through texture mapping. In this way the original image quality and resolution are optimally exploited, and we demonstrate that the rendering can be done very efficiently, at rates comparable to those known from traditional two-plane-parameterized light field rendering.

2 Previous Work

A light field is a 4-D function representing the light leaving or entering a bounded region of 3-D space. The light field parameterization determines how all possible light rays can be characterized. Figure 1 shows some proposed light field parameterizations. In a *two-plane* parameterized light field [9, 5], a ray (dashed lines) is parameterized by its intersection with two parallel planes (solid lines). In order to represent all possible directions, six such pairs of planes are combined into a single light field. *Spherical* light fields [7] use intersection with a positional sphere (large circle) and a directional sphere (small circle) placed at the positional intersection point. *Sphere-sphere* parameters [2] are determined by intersecting twice with the same sphere. *Sphere-plane* coordinates [2] consist of the intersection with a plane, and the normal direction of that plane. The plane is chosen to be perpendicular to the ray and passes through the center, such that its normal can be represented as a position on the surrounding sphere. *Polar* coordinates [16] are given by finding the point on the ray closest to the center and then using the polar angles, the distance, and the rotation of the ray within the tangent plane.

Discretization of the light field data is usually done by a regular (or approximately uniform) sampling in each of the four parameters. For the two-plane parameterization, this means that the light slab can be seen as a regular array of sheared perspective views, with the “eye points” on one of the planes, and “pixels” on the other plane (so they can be called *eye point plane* and *image plane*, respectively). Analogously, the sphere-plane parameterization resembles a collection of orthographic projections, with the projection direction represented on the sphere.

Rendering a light field means extracting and interpolating the right samples from the stored “ray database”. This can be done entirely in software by ray casting (intersecting with the parameterization planes and/or spheres) and performing a quadrilinear interpolation from the 16 nearest samples in the database for every output pixel.

Gortler et al. [5] have proposed to use texture mapping and alpha blending for resampling and interpolating larger fragments of the light field at once. Sloan et al. [14] have extended this technique to irregularly sampled eye point planes, such that eye points (and their associated textures) can be removed or added in order to adapt to a system’s limited resources or to constraints on the rendering time. Camahort [2] has transferred the texture-based rendering to the sphere-plane parameterization by subdividing the directional sphere into polygonal patches that can be treated in very much the same way as the eye point triangles in the two-plane parameterization.

If the light field is not sampled very densely, the interpolation of the ’nearest’ samples yields strong blurring and ghosting artefacts. This can be compensated by depth-correcting the sample coordinates with use of some information about the scene geometry. We call a light field with additional geometric information a *Lumigraph*, as in the original paper on the topic [5]. The geometric information can take the form of a coarse triangle mesh [5], a binary volume [2], or per-pixel depth information [12, 19]. Isaksen et al. [8] demonstrated that several photographic effects can be simulated by using "fake" geometric information to represent an arbitrary focal surface. They also modified the rendering algorithm to allow for arbitrarily oriented cameras, still lying on a common camera plane.

In recent work, Schirmacher et al. [13] use a weaker formulation of two-plane-parameterization. Their work has goals similar to ours here, but they use a global Lumigraph image plane and restrict their approach to warping-based rendering.

One important approach for interactive rendering from an arbitrary set of input images is the view-dependent texture mapping technique by Debevec et al. [4]. Like for Lumigraph rendering, they use the three "best" views for rendering every polygon (even including visibility considerations), but since each polygon is treated independently, smooth transition across polygon edges is not guaranteed.

Heigl et al. [6] apply a variation of the Lumigraph rendering technique for using arbitrary sequence of images directly (e.g. as given by an image stream from a hand-held camera). For every novel view, they project all camera centers into the desired view and perform a triangulation of these projected cameras. This triangulation is then used in very much the same way as we propose to do in this article. However, since this re-triangulation does not maintain a consistent topology in all views, the same viewing ray may be reconstructed from different sets of input views (inconsistency of reconstruction).

Our approach avoids the above-mentioned problems, since we ensure a static and consistent triangulation, and apply a Lumigraph-style rendering algorithm that ensures spatial and temporal continuity. This approach can take advantage of all kinds of geometry data for performing adaptive depth correction, such as a polygonal scene description, per-pixel depth maps, or a binary volume. Furthermore, our algorithm properly handles self-occlusions of the scene, which has so far only been realized for warping-based rendering approaches [12, 13] and view-dependent texture mapping [4].

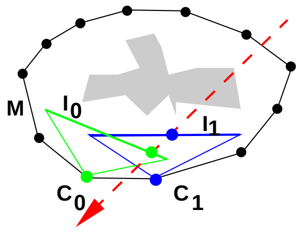

Figure 2: Finding the nearest samples for a given ray involves intersecting the ray with the camera mesh M as well as with the images planes I_0, I_1 associated with the nearest camera vertices C_0, C_1 (in 3D there are three nearest C_i).

3 Free Form Parameterization

Our proposed free form light field is a simple and convenient generalization of the two-plane parameterization, and also shares some ideas with sphere-to-plane light fields. Inspired by the acquisition process using real cameras, a free form light field is defined by a *convex* polygon mesh M called *camera mesh*. The camera mesh consists of N *camera vertices* $C_i, i \in \{0 \dots N-1\}$, each representing a camera's center of projection. Associated with each camera vertex C_i is an *image plane* I_i. All samples through one camera vertex C_i form a perspective image on the associated image plane I_i (cf. Fig. 2). The cameras must be set up in such way that every camera sees the *full silhouette* of the scene (no part of the scene is outside the camera's viewing frustum).

Imagine a continuous light field with a densely populated camera mesh (i.e. every surface point on the mesh is a camera vertex). In this case an arbitrary ray passing through the light field can be thought of as parameterized by its first intersection (s, t) with the camera mesh and its intersection (u, v) with the associated image plane $I_{(s,t)}$. Like in the two-plane parameterization, we can imagine (s, t) to be an eye point, and (u, v) to be a pixel.

In practice the light field contains only of a finite number of camera vertices and pixels, so we must find and interpolate the "nearest" light field samples in order to represent the desired sample (s, t, u, v). This is done by first finding the three camera vertices C_0, C_1, C_2 closest to the intersection point (s, t). Then for each of these camera vertices C_k we can determine the intersection (u_k, v_k) of the desired ray with the corresponding image plane I_k,

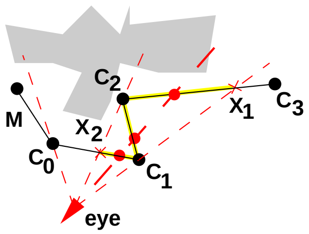

Figure 3: If the camera mesh is concave, a ray intersects the mesh multiple times upon entering the scene. The contributions from (X_2, C_1), (C_1, C_2), and (C_2, X_1) overlap.

and use the four nearest pixels on that plane for interpolation (cf. Fig. 2).

Figure 3 shows why we assume the camera mesh to be convex. If this constraint is violated, a ray may intersect multiple times with the camera mesh until it reaches the scene, so that the contributions from the three patches (X_2, C_1), (C_1, C_2), and (C_2, X_1) overlap as seen from the eye. For software rendering, this could be taken into account by a modified dynamic interpolation computation. In order to yield a smooth transition between the contributions from C_1 and C_3, the algorithm would need to "blend out" the contribution from C_3 at X_1, which requires the computation of C_1's projection onto (C_2, C_3). So non-convex camera meshes require a lot more computational effort, either for projecting the camera vertices into the viewer's image plane and performing a retriangulation for every frame as in Heigl et al. [6], or for normalizing an arbitrary number of blending weights, an operation which is not available in the case of hardware-supported interpolation. As we show in the next section, alpha blending can be used for convex meshes because the interpolation weights can be statically precomputed for every camera patch.

The full-silhouette constraint is needed to make sure that each of the "nearest" three cameras in the mesh actually has data for the desired image patch; otherwise the term "nearest view" would not seem suitable. One exception from this rule is self-occlusion of the scene, which is handled later in Sec. 4. Self-occlusion can only be avoided by choosing the appropriate light field sampling density (refer to [3] for more details about light field sampling).

Looking closely at the sample reconstruction from a free form light field, we see that it is very similar to that for existing parameterizations, except that multiple image planes are used instead of a single plane or sphere, and the camera mesh does not need to be a regularly sampled simple shape. If you take two parallel planes M and I, define all cameras C_i to lie on M and all images I_k to cover the same domain on I (using sheared perspective views), the free form light field exactly reduces to one slab of a two-plane-parameterized light field. In a similar way you can construct a full six-slab lightfield by making M a cube surface and using six image planes.

4 Efficient Rendering

View Selection and Blending. Texture-based rendering of free form light fields can be done in much the same way as in the original Lumigraph framework [5], and as proposed later in Heigl et al. [6] in the context of unstructured light fields. We observe that for all rays passing through one *camera patch* $P = (C_0, C_1, C_2)$ on M, we want to render P's projection onto each of the three image planes I_0, I_1, I_2 (cf. Fig. 4). Each of these projections corresponds to the image on P as viewed from one of the cameras C_i. Similar to the previous work, we draw these three texture-mapped patches, but each on its own image plane. Viewed from the desired eye point, it makes no difference whether or not these patches lie in a common plane (as in the two-plane case). In addition to bilinear texture filtering, alpha values are used to perform the interpolation between the samples. For the texture associated

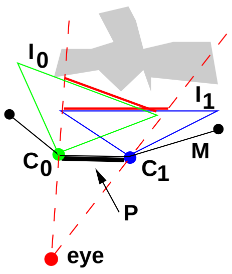

Figure 4: Rendering. See from the eye, the whole patch P should be interpolated from its viewpoints C_0, C_1, so we project P onto I_0 and I_1 and draw the resulting patches with the corresponding alpha-blended textures.

with camera C_k, we use an alpha value of 1 for the projection of C_k onto I_k, and alpha values of 0 for the other two vertices of the patch.

In order to use only those camera patches that actually see the front-facing side of the scene, we perform a simple back-facing test, either by comparing the camera patch normal, or the camera's optical axis to the viewing direction.

Adaptive Depth Correction. If the sampling of the light field is not very dense (e.g. if we only have captured a few views of the scene), the pure light field rendering approach leads to serious blurring and ghosting artefacts. This can be compensated by performing a so-called *depth correction*, if in addition to the image data some approximation of the scene geometry is available. We shoot rays through the camera vertices and find the depth values of the ray's first intersection with the scene geometry by actually performing intersection tests [5], reading back the OpenGL Z-buffer [18], searching in a binary volume [2], or searching in a per-pixel depth map [19]. With the depth information for the triangle, we move the vertices to their true 3-D location in order to take advantage of correct perspective interpolation as featured in concurrent graphics hardware (see Fig. 5).

In addition to the three vertices, we also generate a depth ray for the center of the triangle. If all four depth values are approximately the same, we draw the triangle as is is. If the values differ by more than a certain depth threshold, or if one, two, or three of the four rays do not intersect the scene at all, we subdivide the triangle and perform

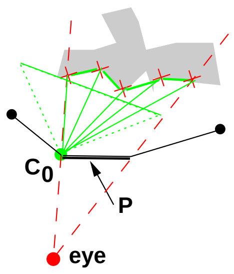

Figure 5: Adaptive depth correction for the image associated with vertex C_0 and patch P. After subdivision, small depth-corrected texture triangles are drawn that faithfully represent the scene geometry for the current view.

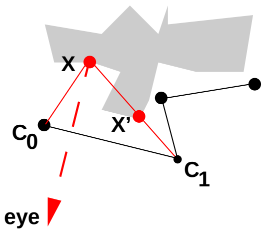

Figure 6: Occlusion. Although C_0 and C_1 are the two closest views for the desired reconstruction, C_1 sees X' rather than the desired point X due to scene self-occlusion.

depth correction recursively[1]. Furthermore, we stop at a minimum triangle size (specified in the viewer's image space), and enforce an initial subdivision by means of a maximal triangle size. This way the image plane triangulation adapts locally to the scene geometry as far as needed for the reconstruction of the current view, and the three cameras that define the subdivided patch finally use the same depth corrected image plane mesh (the polyline between the small crosses in Fig. 5). In order to avoid artefacts at the patch boundaries, we also perform a T-vertex removal by further splitting edges that are subdivided in one patch, but not in the neighbouring patch.

Occlusion Correction. In addition to depth-correcting the texture coordinates, one must also determine if the desired scene parts are actually visible from the input views that have been chosen for the reconstruction. An example of this problem is illustrated in Fig. 6. Although C_0 and C_1 are the closest views for reconstructing the desired scene part (e.g. point X), using C_1's image data would blend a completely different part of the scene with the desired one, resulting in a wrong reconstruction. The solution to this problem is rather straightforward. During the depth correction we already determine the depth of each texture vertex (along the ray from the eye point). Now we can check if a ray from this point (e.g. X in our example) to each of the cameras intersects the scene geometry. In the case of per-pixel depth maps, we simply check if the 3D location of the vertex in the different input views are the same as that of the desired point by reprojecting

[1]The depth value for camera vertices can be determined by a simple depth map lookup. However, for abitrary other points (e.g. during subdivision), a depth map search has to be performed in order to find the corresponding point (inverse mapping). We solve this problem along the lines of Vogelgsang et al. [19].

and comparing the corresponding pixels with their respective depth values. With polygonal geometry or a binary volume representation, an additional ray casting step is needed to detect an intersection with the scene geometry.

If a vertex is occluded from a camera view C_i, we set the blending weight for the texture at this vertex to 0, and adjust the two other weights in the corresponding triangle in such way that they still sum up to 1 (so the patch will not appear darker), and the remaining weights maintain their original ratio (so their relative importance is not changed). For example, let the three weights be x, y, z, and the z vertex must be removed due to occlusion. In that case we add $z \cdot x/(x + y)$ to weight x, add $z \cdot y/(x+y)$ to weight y, and set z to 0. If a vertex is occluded in two views, the remaining view receives a weight of 1. If a vertex is occluded in all three views, we set all three weights to 0.

5 Results

In order to validate our approach, we have implemented the described techniques and performed some experiments with real-world and synthetic light field data. All tests were performed on standard PC hardware with a 1GHz Athlon processor and GeForce2 GTS or GeForce2 MX 32 graphics. If not stated otherwise, all input images have a resolution of $256x256$ pixels, and geometry information is given as per-pixel depth maps.

Fig. 7 compares two-plane (left) to free-form (right) rendering, both Lumigraphs being generated from the same set of 33 photographs. Usually the two approaches yield roughly the same image quality. In this example the two-plane resampling step significantly degrades the image quality, and a more sophisticated warping/reconstruction software is necessary in order to maintain the maximum image fidelity. Fig. 8 shows a view from a different, 169-image free form Lumigraph ("Petruschka"). Again the visual quality is very high, and the object boundaries appear sharp and clear.

Fig. 9 shows a view of a synthetic statue, rendered from a complete (all-around) free form Lumigraph consisting of 107 images. Here you can observe the effect of depth and occlusion correction on the image quality (leftmost: no correction; second-left: correction enabled). We also visualize the depth-correction grid as well as the color-coded blending weights. As expected, the weights are discontinuous where occlusion in one or more input views occurs.

The performance of free-form light field rendering is comparable to that of two-plane parameterized rendering with irregular eye point meshes. The basic difference is that for free form light fields every eye mesh vertex (or the corresponding depth-corrected 3-D point) has to be projected into three different planes instead of one. Without depth correction, this does not increase the rendering time significantly, since the number of eye points is fairly small. In our current implementation all the light fields shown here can be rendered at up to 100 frames per second without depth correction.

When depth correction is switched on, lots of eye mesh vertices are introduced by the adaptive subdivision scheme, and so the free-form overhead is more noticable. For the "coke" sequence (Fig. 7), both two-plane and free form Lumigraphs of the demonstrated quality can be rendered at roughly 30 frames per second. For the "Petruschka" scene (Fig. 8), the free-form frame rates vary between 14 (780 rendering triangles) and 1.7 (5000 triangles). The corresponding frame rates for two-plane rendering are a factor of 1.2 – 2.5 times higher. The statue scene (Fig. 9) can be rendered at between 60 (500 triangles) and 4.6 (3500 triangles) frames/second, again with a performance gain of 1.5 – 2.1 for two-plane rendering. 7–10% of the total free-form rendering time is spent for occlusion correction.

6 Conclusions and Future Work

We have introduced *free form light fields* that combine the efficiency and simplicity of traditional two-plane light slabs with the large flexibility and ease-of-use of an almost free camera arrangement and seamless "all-around" viewing. We have briefly described the modified texture-based rendering algorithm including adaptive depth correction and occlusion correction, and demonstrated the performance and quality of our technique using several data sets.

We believe that free-form light fields are by far the most convenient approach for representing and rendering light fields, especially from real-world data. They allow for arbitrary camera arrangements, and all intrinsic and extrinsic camera parameters such as position, orientation, focal length, and resolution may vary between the different images. Hardware-supported texture filtering is exploited efficiently to resample this data into the desired view. Furthermore, implementation of free-form light fields is no more complicated than that of other light field techniques.

The most useful extension of our approach would be to remove the convexity restriction of our parameterization without sacrificing reconstruction consistency, e.g. for integrating close-up views, which usually do not cover the complete scene silhouette. Furthermore, it is important to investigate how to automatically construct a suitable camera mesh, addressing topics such as mesh generation, view selection, and acquisition planning.

Acknowledgements

The second author's work is sponsored by the Deutsche Forschungsgemeinschaft (DFG) through the project SFB603/C2. Thanks to: Markus Weber for rendering the statue; Christian Rössl for help with the meshing; Manfred Ernst and Frank Firsching for modeling and rendering "Petruschka"; Ingo Scholz, Michael Reinhold, Benno Heigl and many others for providing such image sequences as in our "coke" example [11].

References

[1] E. Camahort and D. Fussell. A geometric study of light field representations. Technical Report TR99-35, Department of Computer Sciences, University of Texas, 1999.

[2] E. Camahort, A. Lerios, and D. Fussell. Uniformly sampled light fields. In *Rendering Techniques '98 (Proc. Eurographics Workshop on Rendering)*, pages 117–130. Springer, 1998.

[3] J. Chai, X. Tong, S. Chan, and H. Shum. Plenoptic sampling. In *Proc. SIGGRAPH 2000*, pages 307–318. ACM Press / Addison Wesley Longman, 2000.

[4] P. Debevec, G. Borshukov, and Y. Yu. Efficient view-dependent image-based rendering with projective texture mapping. In *Rendering Techniques '98 (Proc. Eurographics Workshop on Rendering)*. Springer Verlag, 1998.

[5] S.J. Gortler, R. Grzeszczuk, R. Szeliski, and M.F. Cohen. The Lumigraph. In *Proc. SIGGRAPH 1996*, pages 43–54. ACM Press / Addison Wesley Longman, 1996.

[6] B. Heigl, M. Pollefeys, J. Denzler, and L. van Gool. Plenoptic modeling and rendering from image sequences taken by a hand-held camera. In *Proc. 21. Symposium für Mustererkennung (DAGM '99)*, pages 94–101, 1999.

[7] I. Ihm, S. Park, and R.K. Lee. Rendering of spherical light fields. In *Proc. Pacific Graphics '97*, 1997.

[8] A. Isaksen, L. McMillan, and S.J. Gortler. Dynamically reparameterized light fields. In *Proc. SIGGRAPH 2000*, pages 297–306. ACM Press / Addison Wesley Longman, 2000.

[9] M. Levoy and P. Hanrahan. Light field rendering. In *Proc. SIGGRAPH 1996*, pages 31–42. ACM Press / Addison Wesley Longman, 1996.

[10] G.S.P. Miller, S. Rubin, and D. Ponceleon. Lazy decompression of surface light fields for precomputed global illumination. In *Rendering Techniques '98 (Proc. Eurographics Workshop on Rendering)*, pages 281–292. Springer, 1998.

[11] M. Reinhold, Ch. Drexler, and H. Niemann. Image database for 3-D object recognition. Technical Report LME-TR-2001-02, University of Erlangen, Computer Science Department, Chair for Pattern Recognition, May 2001.

[12] H. Schirmacher, W. Heidrich, and H.-P. Seidel. High-quality interactive Lumigraph rendering through warping. In *Proc. Graphics Interface 2000*, Montreal, Canada, 2000. CHCCS.

[13] H. Schirmacher, Li Ming, and H.-P. Seidel. On-the-fly processing of generalized Lumigraphs. In *Proc. Eurographics 2001*. Blackwell, 2001.

[14] P.-P. Sloan, M.F. Cohen, and S.J. Gortler. Time critical Lumigraph rendering. In *Proc. 1997 Symposium on Interactive 3D Graphics*, pages 17–24. ACM SIGGRAPH, 1997.

[15] P.-P. Sloan and C. Hansen. Parallel Lumigraph reconstruction. In *Proc. Symposium on Parallel Visualization and Graphics*, pages 7–15. IEEE, 1999.

[16] G. Tsang, S. Ghali, E.L. Fiume, and A.N. Venetsanopoulos. A novel parameterization of the light field. In *Image and Multidimensional Digital Signal Processing '98 (Proc. 10th IMDSP Workshop)*. infix, 1998.

[17] Stanford University. Light field camera project. `http://www.graphics.stanford.edu/projects/lightfield/`.

[18] C. Vogelgsang and G. Greiner. Hardware accelerated light field rendering. Technical Report 11, IMMD 9, Universität Erlangen-Nürnberg, 1999.

[19] C. Vogelgsang and G. Greiner. Adaptive lumigraph rendering with depth maps. Technical Report 3, IMMD 9, Universität Erlangen-Nürnberg, 2000.

[20] D.N. Wood, D.I. Azuma, K. Aldinger, B. Curless, T. Duchamp, D.H. Salesin, and W. Stuetzle. Surface light fields for 3D photography. In *Proc. SIGGRAPH 2000*, pages 287–296. ACM Press / Addison Wesley Longman, 2000.

[21] J.C. Yang and L. McMillan. Light fields on the cheap. SIGGRAPH 2000 technical sketch.

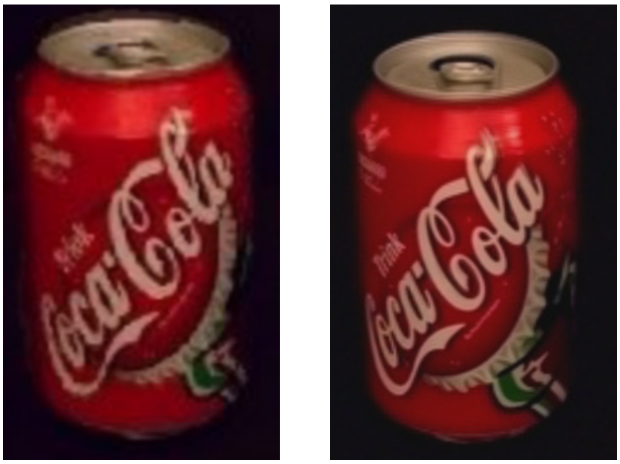

Figure 7: Light field view of the “coke” scene, reconstructed from 33 real images. Left: two-plane parameterization Right: free-form parameterization. The two-plane Lumigraph has been resampled into 30 plane-aligned images in a preprocess, whereas the free form Lumigraph on the right directly uses the original camera frustra.

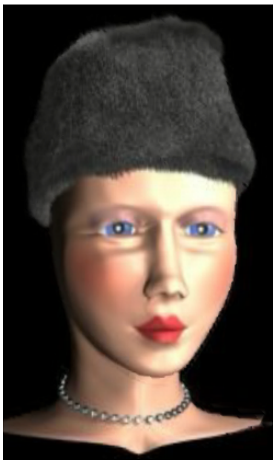

Figure 8: “Petruschka” scene, reconstructed from 169 synthetic images using the free form Lumigraph rendering approach.

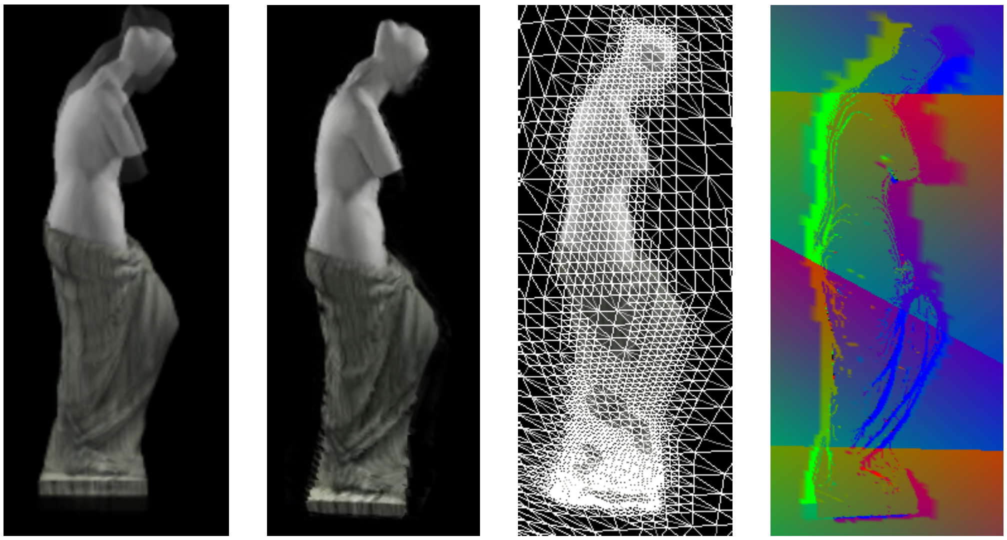

Figure 9: View from a complete (all-around) free form Lumigraph of a statue (107 synthetic images). Left to right: without depth correction (pure light field approach); with adaptive depth correction and occlusion correction; depth-correction grid (image plane triangles); occlusion-corrected blending weights (color coded). Note the typical ghosting in the upper part of the uncorrected reconstruction (left image), and the discontinuous occlusion-corrected weights along the sharp object boundaries (rightmost image).