Visualizing distances between light field and geometry using projective texture mapping

M. Winter[1], G. Greiner[1], F. Vogt[2], H. Niemann[2], S. Krüger[3]

[1] Computer Graphics Group, Am Weichselgarten 9, 91058 Erlangen, Germany
Email: {winter, greiner}@informatik.uni-erlangen.de
[2] Chair for Pattern Recognition, Martensstrasse 3, 91058 Erlangen, Germany
Email: {vogt, niemann}@informatik.uni-erlangen.de
[3] Chirurgische Universitätsklinik, Krankenhausstrasse 12, 91054 Erlangen, Germany
Email: sophie.krueger@chir.imed.uni-erlangen.de

Abstract

Visualizing spatial relations between geometry-enhanced light fields and other geometric objects can be expedient in a wide range of applications, and provide valuable information to the viewer. In this paper, a method is presented that allows visualizing the distance between some geometric objects and an unstructured lumigraph implementation. For that purpose, it uses a technique related to hardware shadow mapping, or in a broader sense, projective texture mapping. Making use of recent graphics hardware, the method is fast enough to allow interactive frame rates. It also makes use of the data structures provided by the unstructured lumigraph, and requires almost no additional pre-processing steps. The distance visualization works independent of the current view, only depending on the current light field image, and thus providing very accurate results for the measured distances. Furthermore, it provides support for semi-transparent geometry.

1 Introduction

Light fields represent an image-based rendering technique that provides an effective way of generating arbitrary views out of a given set of images [1, 2]. Since their introduction in 1996, many new approaches and enhancements have been published, considering light fields on their own. However, one area that is also of great interest and still to be investigated, is the rendering of light fields in combination with other geometry, and deriving spatial relationships between these different “objects“.

In medical applications which make use of light field rendering techniques, such a combination would be of great value. For example, light fields can be used to visualize the surface of an operating field [19], and can be rendered with some geometric structures that provide additional information about the operation field, e. g. occluded organs under the surface. In this case, it is also of great interest for the surgeon to get information about the distance between these occluded parts and the surface, as this could provide some helpful information for planning the surgery procedure.

In the past, investigations in the area of combining light fields and geometry have been hampered by the image-based nature of light fields, i. e. the lack of depth or geometric information which is necessary to perform combined rendering and reliable distance measuring. This restriction was abolished amongst others by the *unstructured lumigraph (ULG)* [4], a light field model that employs “real“ geometry for rendering, which can be used for the derivation of spatial relationships.

Based on the lumigraph’s geometric information, there are a number of methods to do distance visualization. A quite easy way to visualize distances between two geometric objects in general is to perform a weighted blending when rendering them into a common graphics context, as done in [18]: Using the z-buffer values of these objects at each pixel, the distance between both objects is determined, and the degree of opacity for the object being in front is adjusted depending on that distance.

This method provides a very intuitive way to visualize distances. Unfortunately, it is highly view-dependent, and therefore does not give consistent results when looking at the scene from different viewpoints, which may lead to confusion of the surgeon.

Another completely different approach to show distances between arbitrary objects are so-called *distance fields* [15]. A distance field is a signed or unsigned scalar field that is registered to an object: Each grid value of the field contains the distance to the nearest point on the object‘s surface. A nice survey about distance fields can be found in [14].
Demiralp et al. [16] showed how distance fields can be used to visualize distances between objects by using a combination of color mapping and iso-contouring depending on the distance field’s values. A similar approach was used in [17] for the purpose to visualize differences between a 3D volume and its corresponding reconstructed iso-surface.
Though these approaches result in very accurate distance visualizations, they are quite general methods and developed for static geometry. This makes them less useable for storing a light field’s distance values, as it might change it’s appearance (and therefore its distance information) in each new view.

For an algorithm that performs an accurate and reliable distance visualization between an unstructured lumigraph and other geometry, the following requirements should be met:

- The algorithm should make use of the methods and data structures that are provided by the unstructured lumigraph, aside from depth or geometric information. That way, computational overhead is minimized, and better results can be expected.
- The visual results must be in line with the current light field image, but should be fairly independent from the current camera position. This means that e. g. when looking at a light field “from behind“, the distance visualization must still work with the light field.
- For the area of application described above, the geometry usually is expected to be “behind“ the light field, in order to represent arteries, veins or organs occluded by the light field image. Therefore, an accurate blending between light field and geometry is needed.
- Often, it is preferable to have semi-transparent geometry in order to also get distance information of occluded parts of the geometry, so the algorithm should support transparent geometry, too.
- The algorithm should be fast and require virtually no pre-processing.

In this paper, we describe a method that uses a kind of enhanced shadow mapping in order to visualize distances between geometric objects and light fields rendered by an unstructured lumigraph. The method was developed to work in cooperation with the ULG renderer implementation from the lgf3 framework [7], and employs its depth data structures to perform accurate distance visualization. Additionally, we suggest how to combine a light field generated by that renderer and some geometry into a common rendering context.

This paper is organized as follows: Section 2 gives a short overview about previous publications on light field rendering and shadow mapping, Section 3 describes the implementation of the light field renderer that is used in combination with our method. In Section 4, the actual visualization algorithm is described in detail, as well as its combination with light field rendering. Finally, Section 5 shows some performance results of the algorithm’s implementation, and a summary is given in Section 6.

2 Previous work

Light field rendering is a topic of research in the domain of image-based rendering techniques that has grown more and more popular over the last years. The basic approach of light fields is that the scene information is captured by some images, and new views can be reconstructed by using a combination of these images. The first methods for rendering two-plane light fields were published in 1996, using either a hand-held camera [1] or a specialized motion control platform [2] for image aquisition. After that, more sophisticated algorithms have been developed, like free form light fields [3] or light field mapping [5], which also make use of approximated surface geometry in order to achieve better results.
Buehler et al. introduced the unstructured lumigraph [4] which was designed to be very flexible in its requirements concerning the visual and geometric information about a scene: It should be possible to generate new views using either accurate surface meshes with little visual information, or many scene images combined with almost no geometric data. Evers-Senne et al. developed a system for both capturing and rendering light fields using

view-dependent local geometry that is generated from computed depth maps [6]. Similar techniques can be found in the lgf3 framework [7], whose hardware-accelerated ULG renderer was used for this work to render the light field part.

Shadow mapping is a method for shadow creation that is heavily used in the game and film industry [9]. The basic shadow mapping approach was published by Williams et al. [8]. There a screen-based rendering technique is described that utilizes a z-buffer: In a first rendering pass, the scene is rendered from the light's viewpoint into that z-buffer, which afterwards contains the *shadow map* that captures the depth values of the scene. In the second pass, the scene is rendered from the original viewpoint, where each pixel is transformed into the depth map's space, and a comparison is done between the pixel's z-value and the shadow map's corresponding value. Depending on this comparison, the pixel is either lit or in shadow.
Using more recent graphics hardware, it is possible to implement shadow mapping using hardware-accelerated projective texture mapping [10]. This technique uses automatic texture coordinates generation and a texture matrix to compute texture coordinates that enable the impression that a texture is projected from a point in space onto the geometry's surface, like an overhead projector.
Since then, many interesting increments and improvements have been added to shadow mapping, like adaptive, perspective or multiple-depth shadow maps [11, 12, 13], but these techniques are out of the scope of this paper.

3 ULG renderer implementation

For the visualization technique that is presented here, we use an implementation of the unstructured lumigraph that is available in the lgf3 framework [7]. This implementation uses some own data structures and components for handling light fields that will be explained briefly in the following, as some of these structures will be used by our method, too. Figure 1 gives a survey of these structures.

The basic structure of the ULG implementation used here is called a *light field*. A light field in this context means a kind of triangular mesh that contains so-called *views* at its vertices. These views represent camera positions from where the different images of the light field were recorded. Each view has several so-called *layers* associated with it: A radiance layer that contains the actual visual image, a depth layer for optionally holding a depth map of the current view, a confidence layer that provides validity information for each image pixel of the corresponding radiance layer, and a mesh layer that may store a depth mesh for the current image.

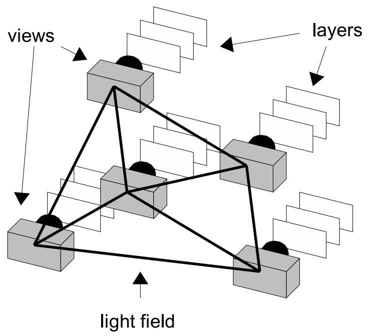

Figure 1: Some components and data structures used in the lgf3 framework for storing light field information

3.1 Light field rendering

In order to generate a new view out of the views that are provided by the light field, the ULG renderer works as follows (Note that we will only consider the use of mesh layers for depth information; Nevertheless, depth layers are supported as well):

- First, a global ranking is performed among all views of the light field, using some approaches described in [6]: Each view is ranked according to its orthogonal distance to the new view, and the angle between both view's viewing directions. Afterwards, only the "best" n views are employed for generating the new view.
- For each selected view, its corresponding depth mesh is rendered into a depth buffer, with the modelview and projection matrices set according to the settings of the new view.
- After the depth buffer has been filled with all meshes, it is sampled uniformly from the new view's position for creating a new depth mesh, the so-called *screen mesh*. Figure 2 shows such a screen mesh from both the view's position and a side view.
- Finally, for each view, this screen mesh is rendered from the new view's position, and the

radiance information of the current view is applied to the mesh by using projective texture mapping. The final image is generated by blending and accumulating all render passes.

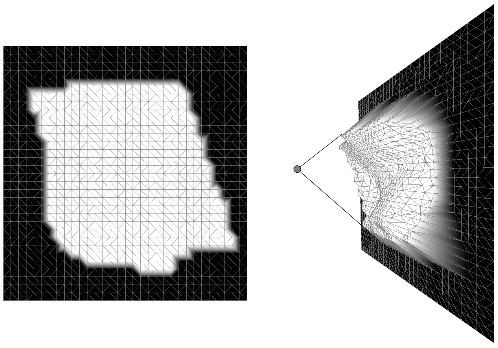

Figure 2: The screen mesh seen from the view's position (left) and from a side view (right). Areas where no depth mesh was rendered are colored black, others white. The mesh's corresponding view is depicted as a gray dot.

4 The distance visualization algorithm

The algorithm that is proposed here is based on the technique of projective texture mapping; More precisely, it is closely related to the shadow mapping algorithm. As described in [10], hardware shadow mapping basically works in two rendering passes: In the first pass, the scene is rendered from the light's position in order to get the shadow map which is the depth map for that scene; In the second pass, this shadow map is projected onto the scene by using the technique of projective texture mapping. During that mapping, an additional depth comparison between each fragment's depth value and the depth map's corresponding value is done in graphics hardware, giving a boolean result whether the fragment's position is in the shadow or not. This result can be used in further post-processing steps, like register combiners or fragment shaders, to compute the fragment's final color.

4.1 Overview of the algorithm

Our visualization method can now be described as follows: Instead of rendering a depth map of the scene, the algorithm employs the views and their depth maps or depth meshes that are provided by the *light field* structure. If depth meshes are to be used, these are rendered into depth maps one time in a pre-processing step. One important point here is that the depth maps are no "real" depth maps copied directly from a depth buffer, but contain linearized values in the range $[0..1]$. This is done to reduce computational costs in the rendering step.

The algorithm also uses the results of the global view ranking performed by the ULG renderer, in order to know the views that are used for rendering the current light field image. Each selected view is then assigned a texture matrix that contains all transformations needed for projective texture mapping from this view's position.

With these informations, the algorithm's visualization procedure is initialized in the following way (cf. Figure 3): For each view selected by the global ranking, the corresponding depth map and texture matrix is bound to a separate texture unit, and automatic generation of texture coordinates is set up for that unit, thus enabling projective texture mapping of each view's depth map onto the geometry's surface (Figure 3a). Then, the visualization is done by rendering the geometry using a fragment shader. For each fragment, this shader performs multiple depth comparisons at once, one for each selected view, similar to the hardware shadow test. However, instead of returning a boolean depth result like in shadow mapping, these comparisons result in a distance value, which is used to access a color texture that maps distances to colors (Figure 3b-c).

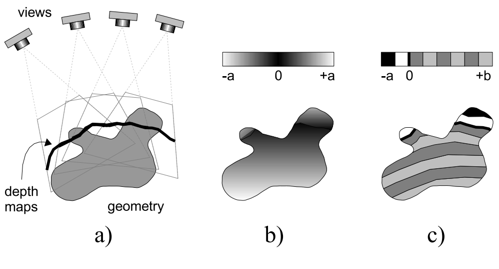

Figure 3: The visualization algorithm at a glance: a) Rendering is initialized with the selected views' depth maps being projected onto the surface. b) Using a fragment shader, depth comparisons are done, resulting in distance values for all surface points. c) Distances are mapped to colors using a color texture.

4.2 Shading in detail

The distance visualization itself is done by rendering the designated geometry using the fragment shader. This shader works as follows: First, for each view v_i, the distance between the current fragment and the view's depth map is computed. For that purpose, a projective depth texture lookup is done using the automatically generated texture coordinates t_i for that view, resulting in a lookup value c_i. Then, a validity test of this lookup result is done, dependent on various aspects, which will be discussed later. If this test fails, the corresponding view is discarded for the following steps. Next, the world space distance between the z-values of t_i and c_i has to be computed, where positive distance values should stand for "$t_i > c_i$". For that purpose, these z-values first have to be transformed into real depth values, using equation (1) for c_i and equation (2) for t_i (remember that the z-value of c_i already is linearized).

$$d_{1,i} = z_i(f_i - n_i) + n_i \tag{1}$$

$$d_{2,i} = \frac{n_i}{1 - z_i(1 - \frac{n_i}{f_i})} \tag{2}$$

After this transformation, both values $d_{1,i}$ and $d_{2,i}$ lie in the range $[n_i..f_i]$ between near and far plane, and thus are comparable. However, a simple $d_{2,i} - d_{1,i}$ isn't sufficient, since the perspective distortion of these values is not yet taken into account. To fix this situation, one has to compute a distortion factor as depicted in Figure 4: The left figure shows the depth value $d_{1,i}$ from c_i indicated as a white point on the depth map's surface, and $d_{2,i}$ from t_i as a black point. The value $d_{o,i} = d_{2,i} - d_{1,i}$ represents the distance between these two points in orthographic view, which is what has been computed until now. However, as a perspective mapping is used, we have to find the length of $d_{p,i}$, which is shown on the right side.

Finding the right formula for $d_{p,i}$ is straighforward: Using a plane E at distance 1 to the view's position, and the theorem on intersecting lines, one can easily figure out that $d_{p,i} = d_{o,i} \cdot f_{d,i}$, where $f_{d,i}$ is the distortion factor. In order to compute this value, a ray is shot from the view's position to t_i and stopped at the intersection with the aforementioned plane E. Then, the value of $f_{d,i}$ equals to the length of this ray.

Putting it all together, the searched distance $d_{p,i}$ between t_i and c_i can be calculated using equation (3), where $v_{l,i}$, $v_{r,i}$, $v_{b,i}$ and $v_{t,i}$ come up to the left, right, bottom and top border values of the current view's camera frustum related to plane E.

$$d_{p,i} = (d_{2,i} - d_{1,i}) \left| \begin{pmatrix} tc_{i,x}(v_{r,i} - v_{l,i}) + v_{l,i} \\ tc_{i,y}(v_{t,i} - v_{b,i}) + v_{b,i} \\ 1 \end{pmatrix} \right| \tag{3}$$

Having computed the distances $d_{p,i}$ of the current fragment for all views v_i, the shader selects the maximal value d_{max} among the distances that have not been discarded. Using this value, the shader performs a lookup into a user-defined color texture that maps distances to color values, and modulates the fragment's original color with the lookup result. If no value for d_{max} could be determined, e. g. due to missing valid distance values, the fragment's color is modulated with the "highest" color texture entry corresponding to the furthermost distance that is supported.

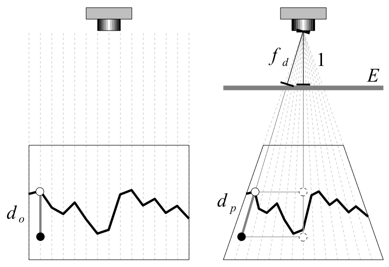

Figure 4: Computation of the distortion factor for transforming distances from orthographic (left) to perspective projection (right). The black and white points represent the depth values of t_i and c_i, respectively.

4.3 Validity tests

As mentioned before, a view v_i may be discarded during shading for various reasons. One reason might be that the corresponding texture coordinates t_i that were generated lie outside the range $[0..1] \times [0..1]$, for which no valid depth information is available.

Another reason is that there can be invalid areas in the depth map, like areas marked in the confidence map or areas outside the corresponding depth mesh's border (cf. Figure 2, left). In order to find these areas and handle them, each depth map contains an additional alpha channel where a valid pixel has $a = 1.0$, otherwise zero. Invalid texture lookups can thus be found by testing if $a < 1.0$ Figure 5 exemplifies this: The figure shows a depth map with "untouched" areas, i. e. where the depth value is 1.0, that is used for distance visualization. The top row shows the result without using an alpha channel: All areas are used for visualization, resulting in wrong mapping at the frustum's borders (top right). The bottom row demonstrates the usage of alpha values (white points) and linear interpolation to exclude all areas with $a < 1.0$, which results in correct distance mapping (bottom right).

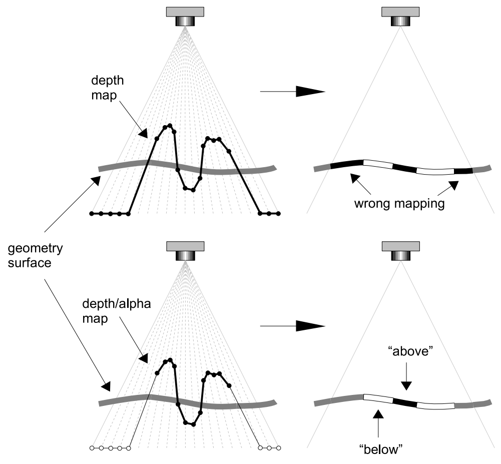

Figure 5: Doing distance visualization without respecting invalid areas in depth map results in wrong color mapping (top). Correct mapping is done by setting alpha to zero for invalid depth entries, indicated as white points (bottom).

4.4 Enhancing techniques

The algorithm described above works very well, but can be quite slow on some configurations (cf. Section 5); Especially the fragment shader is very expensive due to its heavy usage of texture units. Additionally, recent graphics hardware limits the number of available texture coordinates per vertex to eight, thus also limiting the number of usable views in the shader. Therefore, several techniques have been considered and implemented in order to accelerate and generalize the basic algorithm.

The first approach is to simply use less views for distance visualization when the camera position is moving, i. e. when the user changes his viewpoint. In order to reduce visual discrepancy, the views were discarded beginning with their lowest ranking position. This approach is easy to implement and quite fast, but at the cost of accuracy, especially if the missing views provide a significant portion of depth information.

Another similar approach is to use a composite view for the distance visualization. For that purpose, before doing the visualization, a new view is computed out of the views that would be used otherwise, and a composite depth texture is generated by rendering the corresponding depth meshes from the new view's position. The composite view's position and viewing direction are computed by averaging the corresponding values of the original views. Then, the camera frustum for the view is constructed such that it contains all frustums from the old views (see Figure 6). This way, it is guaranteed that the depth meshes of these views will be rendered completely inside the new view's depth texture.
This approach is very fast, as it reduces the number of used views to one, and the time for generating the composite view including depth texture for each new frame is negligible. Furthermore, one major advantage of this approach is that it abolishes the restriction of being limited to the number of available texture coordinates. However, it suffers from the same problems as the previous approach, although the visual differences are much less obvious than when simply excluding views. Figure 7 shows a comparison of the visualization results between a composite view and its corresponding separate views.

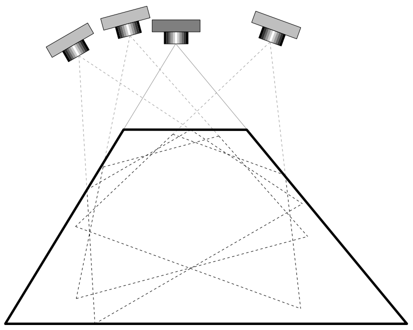

Figure 6: The new camera's frustum containing the original frustums

In addition to changing the number of views, one other possibility to gain more performance is to change the rendering resolution when moving the camera position: The screen's viewport is scaled down by a certain factor s, then the normal rendering process is performed. After rendering, the resulting image is copied into a texture, and drawn over the whole screen, using either nearest-neighbor look-ups or linear interpolation.
This method provides the best consistency with the corresponding still image, and it allows fair rendering speed. As it is a different approach to the ones described before, it can be combined with them to achieve even higher frame rates.

4.5 Combining light field and geometry

Looking at the working method of the ULG renderer described in Section 3, one can easily recognize that a simple combination of the unstructured lumigraph and the textured geometry is possible by rendering both into the same graphics context with depth test enabled, since the actual light field is implemented as a screen mesh that can be rendered like “ordinary“ geometry. However, this simple combination is useless if the geometry is to be placed behind the opaque light field, which is the typical case for the medical scenario described in the introduction. Using the straightforward solution of first rendering the geometry, and afterwards rendering the light field with blending enabled, wasn't an option either, because the distance visualization is considered an add-on to the ULG renderer and therefore should be dependent on its rendering results, not the other way around.
Therefore, a more sophisticated approach was developed that performs an “intelligent“ blending between light field and geometry using the stencil buffer. This approach works as follows:

- First, the light field's screen mesh is rendered the normal way into depth and color buffers
- Second, the geometry is rendered into the stencil buffer with depth test disabled, resulting in a silhouette stencil of the whole geometry
- Next, the geometry is rendered into the depth and color buffers with depth test enabled. Additionally, all stencil values are cleared where the corresponding fragments passed the depth test. After this step, only those stencil values are marked where the geometry is occluded by the light field.
- Now, the depth buffer is cleared, and the geometry is rendered again, employing a user-defined blending function that makes the color buffer content (the light field) appear semi-transparent. Additionally, rendering is restricted to those areas that are still marked in the stencil buffer.

Figure 8 illustrates the resulting combination of an ULG light field and a geometric object, rendered with distance visualization and using the blending approach described above.

5 Results

The visualization method was implemented and tested on a P4 with 2.20GHz and 1GB memory, using a Quadro FX 1000 with 128MB video memory. Table 1 shows the frames per second for different test scenarios that were achieved on this configuration: The first line shows the number of views that were used for each test cycle; The next one illustrates the performance of the ULG renderer on its own for each number of views. In the line “ULG + vis“, the results of the combined rendering of both light field and geometry using distance visualization are given, and the last two lines show the results for some enhancing techniques, namely the composite view and scaled rendering using a factor $s = 2$, as depicted in Section 4.4.
One can recognize that the most performance loss in combined rendering is caused by the fragment shader and its heavy usage of texture resources: The performance decreases radically with the number of used views. In contrast, when combining the separate views into a composite view, the frame rate remains almost constant, because there only one view is used for visualization. As can be seen in the last line, scaled rendering is not nearly as efficient as expected, showing again the bottle neck in the fragment shader.

Figure 2 shows the results for the same test scenarios on a P4 with 3.0GHz and 1 GB memory, using a GeForce 6800 with 128MB memory. Here, it is recognizable that the fragment shader is much better supported by the graphics hardware, which results in good performance values for the visualization itself. In addition, the acceleration techniques provide some performance gain, especially when the number of used views increases.

Table 1: The performance values for a P4, 2.20GHz / Quadro FX 1000 (in fps)

# views	3	5	8
ULG	16.2	14.3	11.1
ULG + vis	4.5	2.5	1.4
ULG + vis (composite)	7.4	7.1	6.4
ULG + vis (scaled)	4.7	2.8	1.4

Table 2: The performance values for a P4, 3.00GHz / GeForce 6800

# views	3	5	8
ULG	24.1	21.8	19.0
ULG + vis	17.9	14.0	9.7
ULG + vis (composite)	17.7	16.2	13.0
ULG + vis (scaled)	18.2	15.9	12.0

6 Conclusion

We have presented a method for visualizing distances between geometry-enhanced light fields and arbitrary geometry that is easy to implement and fast enough to allow interactive frame rates. The visualization is done using an enhanced approach of shadow mapping that makes use of recent graphics hardware.

One major drawback is its excessive usage of texture units and texture coordinates, as these are limited resources even on today's graphics hardware. However, these limitations can be avoided by combining views (cf. Section 4.4), though this technique does not provide exactly the same results. Therefore, a possible improvement of the visualization method is to develop a more sophisticated combining algorithm that minimizes visual differences between the composite view and its embedded views.

References

[1] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen "The Lumigraph", *23rd annual conference on Computer graphics and interactive techniques*, ACM Press, pp. 43-54 , 1996

[2] Marc Levoy, and Pat Hanrahan, "Light field rendering", *23rd annual conference on Computer graphics and interactive techniques*, ACM Press, pp. 31-42, 1996

[3] Hartmut Schirmacher, Christian Vogelgsang, Hans-Peter Seidel, and Günther Greiner, "Efficient Free Form Light Field Rendering", *Proceedings of the Vision Modeling and Visualization Conference 2001*, Aka GmbH, pp. 249-256, 2001

[4] Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler, and Michael Cohen, "Unstructured lumigraph rendering" *28th annual conference on Computer graphics and interactive techniques*, ACM Press, pp. 425-432, 2001

[5] Wei-Chao Chen, Jean-Yves Bouguet, Michael H. Chu, and Radek Grzeszczuk, "Light field mapping: efficient representation and hardware rendering of surface light fields", *ACM Transactions on Graphics. 21 (3)*, ACM Press, pp. 447-456, 2002

[6] J.-F. Evers-Senne, and R. Koch, "Image Based Interactive Rendering with View Dependent Geometry", *Eurographics 2003 Proc., Volume 22, Issue 3*, September 2003

[7] Christian Vogelgsang, "The lgf^3 Project: A Versatile Implementation Framework for Image-Based Modeling and Rendering", PhD Thesis, *Arbeitsberichte des Instituts für Informatik, Friedrich-Alexander-Universität Erlangen-Nürnberg, Band 38 (1)*, May 2005

[8] Lance Williams, "Casting curved shadows on curved surfaces", *Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, ACM Press, pp. 270-274, 1978

[9] William T. Reeves, David H. Salesin, and Robert L. Cook, "Rendering antialiased shadows with depth maps", *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, ACM Press, pp. 283-291, 1987

[10] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul Haeberli, "Fast shadows and lighting effects using texture mapping", *ACM SIGGRAPH Computer Graphics, Volume 26 , Issue 2*, ACM Press, pp. 249-252, 1992

[11] Randima Fernando, Sebastian Fernandez, Kavita Bala, and Donald P. Greenberg, "Adaptive shadow maps", *SIGGRAPH 2001 Conference Proceedings*, Addison Wesley, August 2001

[12] Marc Stamminger, and George Drettakis, "Perspective Shadow Maps", *Proceedings of ACM SIGGRAPH 2002*, ACM Press/ ACM SIGGRAPH, July 2002

[13] Christian Azambuja Pagot, João Luiz Dihl Comba, and Manuel Menezes de Oliveira Neto, "Multiple-Depth Shadow Maps", *XVII Brazilian Symposium on Computer Graphics and Image Processing*, pp. 308-315, 2004

[14] Miloš Šrámek, "Distance Fields in Visualization and Graphics", VISKOM, Austrian Academy of Sciences

[15] Sarah F. Frisken, Ronald N. Perry, Alyn P. Rockwood, and Thouis R. Jones, "Adaptively sampled distance fields: a general representation of shape for computer graphics", *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., pp. 249-254, 2000

[16] Cagatay Demiralp, G. Elisabeta Marai, Stuart Andrews, David H. Laidlaw, Joseph J. Crisco, and Cindy Grimm, "Modeling and Visualization of Inter-Bone Distances in Joints", *Visualization '01 Work in Progress Proceedings*, pp. 24-25, October 2001

[17] Mario Botsch, David Bommes, Christoph Vogel, and Leif Kobbelt, "GPU-based Tolerance Volumes for Mesh Processing", *12th Pacific Conference on Computer Graphics and Applications*, pp. 237-243, 2004

[18] André Neubauer, Stefan Wolfsberger, Marie-Thérèse Forster, Lukas Mroz, Rainer Wegenkittl, and Katja Bühler, "STEPS - An Application for Simulation of Transsphenoidal Endonasal Pituitary Surgery", *Proceedings of IEEE Visualization 2004*, pp. 513-520, October 2004

[19] F. Vogt, S. Krüger, J. Schmidt, D. Paulus, H. Niemann, W. Hohenberger, and C. H. Schick, "Light Fields for Minimal Invasive Surgery Using an Endoscope Positioning Robot", *Methods of Information in Medicine, 43(4)*, pp. 403-408, 2004

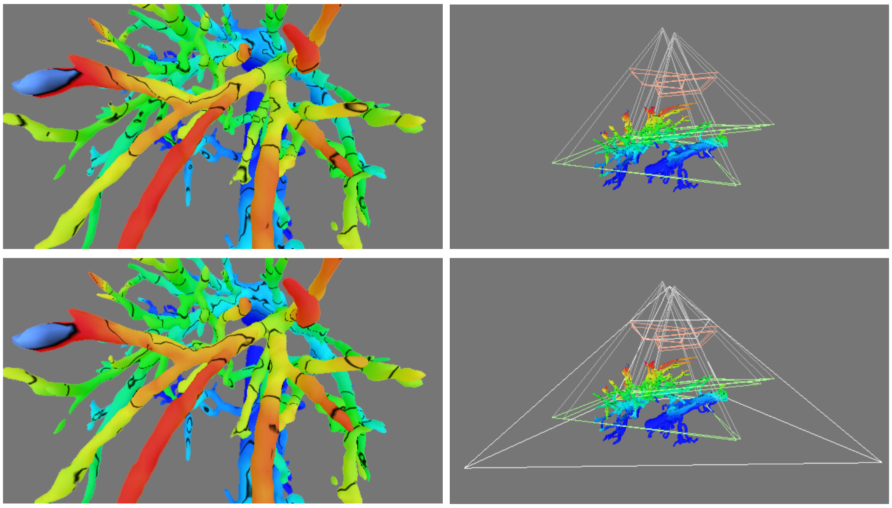
Figure 7: Visual differences between a composite view and its embedded views: The upper row illustrates the visualization result using separate views (left) as well as the corresponding camera frustums (right). The lower row shows the same scenario using a composite view, where the corresponding view frustum is drawn in white color (right).

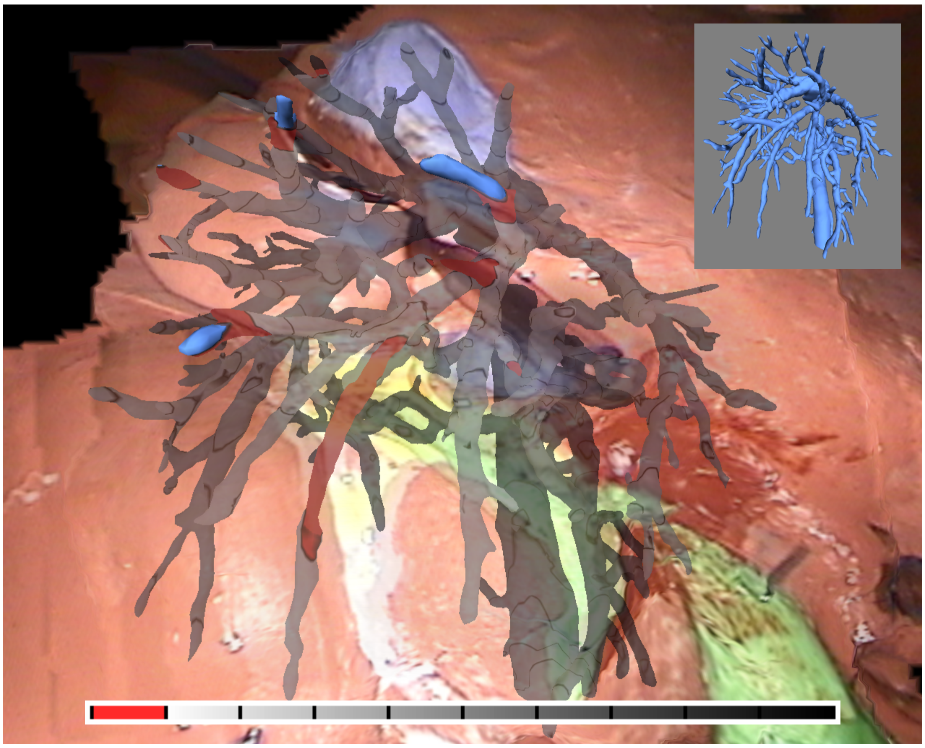
Figure 8: A combination of light field and geometry using distance visualization, presented together with the employed color map (bottom) and the original geometric object (upper right). The color map is set such that the left end of the map corresponds to a penetration depth of 0cm, and the right side to 10cm.