EUROGRAPHICS 2008 / G. Drettakis and R. Scopigno
(Guest Editors)

*Volume 27 (2008), Number 2*

# Floating Textures

M. Eisemann[1], B. De Decker[2], M. Magnor[1], P. Bekaert[2], E. de Aguiar[3], N. Ahmed[3], C. Theobalt[4], A. Sellent[1]

[1]Computer Graphics Lab, TU Braunschweig, Germany
[2]Hasselt University - Expertise Centre for Digital Media - Transnationale Universiteit Limburg, Belgium
[3]Max-Planck-Institut Informatik, Saarbrücken, Germany
[4]Stanford University, Stanford, USA

**Abstract**

*We present a novel multi-view, projective texture mapping technique. While previous multi-view texturing approaches lead to blurring and ghosting artefacts if 3D geometry and/or camera calibration are imprecise, we propose a texturing algorithm that warps ("floats") projected textures during run-time to preserve crisp, detailed texture appearance. Our GPU implementation achieves interactive to real-time frame rates. The method is very generally applicable and can be used in combination with many image-based rendering methods or projective texturing applications. By using Floating Textures in conjunction with, e.g., visual hull rendering, light field rendering, or free-viewpoint video, improved rendering results are obtained from fewer input images, less accurately calibrated cameras, and coarser 3D geometry proxies.*

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

## 1. Introduction

To take advantage of the continuing progress in graphics hardware capabilities for realistic rendering, ever more detailed model descriptions are needed. Because creating complex models with conventional modeling and animation tools is a time-consuming and expensive process, direct modeling techniques from real-world examples are an attractive alternative. By scanning, or reconstructing, the 3D geometry of an object or scene and capturing its visual appearance using photos or video, the goal of direct modeling techniques is to achieve photo-realistic 3D rendering results at interactive frame rates.

Texture mapping was introduced in computer graphics as early as 1974 as a very effective means to increase visual rendering complexity without the need to increase geometry details [Cat74]. Later on, projective texture mapping overcame the need for explicit texture-to-surface parameterization and enabled applying conventional photographs directly as texture [SKvW*92]. To texture 3D geometry from all around and to reproduce non-Lambertian reflectance effects View-dependent texture mapping [DTM96] and Unstructured Lumigraph Rendering [BBM*01] use multiple photographs taken from different viewpoints. If exact 3D geometry and sufficiently many, well-calibrated and registered images are given, image-based modeling techniques, like [WAA*00, LKG*03], achieve impressive 3D rendering results.

Unfortunately, acquiring highly accurate 3D geometry and calibrated images turns out to be at least as tedious and time-consuming as model creation using software tools. In response, a number of different image-based rendering (IBR) techniques have been devised that make do with more approximate geometry. With a mere planar rectangle as geometry proxy, Light Field Rendering arguably constitutes the most "puristic" image-based rendering technique [LH96]. Here, many images are needed to avoid blurring or ghosting artefacts [CCST00]. If more appropriate depth maps are additionally available, Lumigraph rendering can compensate for parallax between images to yield convincing results from less images [GGSC96, BBM*01]. Other image-based rendering approaches implicitly or explicitly recover approximate 3D geometry from the input images to which the images are applied as texture [MBR*00, VBK05, CTMS03]. In general, however, the price for contending with approximate 3D geometry is that (many) more input images must be available, else rendering quality degrades

**Figure 1:** *Comparison of standard linear interpolation using the Unstructured Lumigraph weighting scheme (left) and the Floating Textures approach (right) for similar input images and visual hull geometry proxy. Ghosting along the collar and blurring of the shirt's front, noticeable in linear interpolation, are eliminated on-the-fly by Floating Textures.*

and artefacts quickly prevail, Fig. 1. Another unsolved challenge is that, while image-based rendering can compensate for coarse 3D geometry if sufficiently many input images are available, all techniques still require well calibrated cameras and input images. Thus, time-consuming camera calibration procedures must precede image-based object acquisition every time. But even with the utmost care taken during acquisition, minute camera calibration inaccuracies, tiny 3D scanning holes, and small registration errors can occur and visibly degrade rendering quality of the model. The only option to rescue rendering quality then is to try to "fix" the model, registration, or calibration by hand, or to repeat the acquisition process all over again.

What is needed is a multi-image texturing algorithm that achieves best-possible results from imprecisely calibrated images and approximate 3D geometry. In the following, we propose a GPU-based multi-view texturing algorithm towards this goal. Because the algorithm runs independently on the graphics card, it can be used in conjunction with many image-based modeling and rendering (IBMR) techniques to improve rendering outcome.

As particular contributions, our paper presents:

- a novel texturing algorithm that constitutes a symbiosis between classical linear interpolation and optical flow-based warping refinement to correct for local texture misalignments and warping the textures accordingly in the rendered image domain;
- a novel weighting and visibility scheme which significantly reduces artefacts at occlusion boundaries;
- a general algorithm that can be applied in conjunction with many IBMR techniques to improve rendering quality;
- an efficient GPU-based implementation of the proposed algorithm which achieves interactive to real-time rendering frame rates;
- a simple extension to our Floating Textures approach for static scenes which reduces the actual rendering part to a simple texture look-up.

The remainder of the paper is organized as follows. After reviewing relevant previous work in Section 2 we examine the underlying problem of ghosting and occlusion artefacts in multi-view projective texture mapping in Section 3. In Section 4 we describe our Floating Textures as a way to eliminate ghosting, occlusion and calibration artefacts. Implementation details are given in Section 5, and experimental evaluation results for a variety of different test scenes and IBMR techniques are presented in Section 6 before we discuss limitations and conclude with Section 7.

## 2. Previous Work

**Sampling Problem:** In light field rendering, a novel view is generated by appropriately re-sampling from a large set of images [LH96, MP04]. Given sufficiently many input images, the synthesized novel view can be of astonishingly high quality. Else, ghosting artefacts or, at best, blurring degrade light field rendering quality. Several researchers have investigated how many input images are minimally needed to create artefact-free light field rendering results [CCST00, LS04]. Alternatively, rendering quality can be enhanced by considering not only input camera positions but also the current viewpoint [ESM07] or by adding back in high frequency components [SYGM03]. Nevertheless, these approaches still introduce at least some image blur if the scene is undersampled.

**Geometry aided IBR:** Instead of relying on sampling density, another approach to increase rendering quality from sparsely sampled image data is to make use of a geometry proxy representing the scene. Prominent examples are Lumigraph Rendering by Gortler *et al.* [GGSC96], Unstructured Lumigraph Rendering by Buehler *et al.* [BBM*01] and Debevec *et al.*'s View-dependent Texture Mapping [DTM96]. Isaaksen *et al.* [IMG00] showed that if scene depth estimation is precise enough and no occlusion occurs, any single scene element can, in theory, be reconstructed without ghosting artefacts. Several other researchers built upon this insight to improve rendering results [ZKU*04, VBK05]. An interesting approach which not only blends colours on a geometry proxy but attempts to reconstruct a consistent, view-dependent geometry of the scene with the aid of bilateral filtering was only recently presented by Waschbüsch *et al.* [WWG07].

In image-based modeling approaches, high-quality 3D geometry scans of real-world objects are used and textured with a collection of photographs [RCMS99, WAA*00, Bau02, ZWT*05]. Acquiring these detailed models is time-consuming and, of course, is possible only for scenes that

hold still during acquisition. In any case, image calibration inaccuracies, subcritical sampling, and geometry acquisition errors remain potential sources of rendering artefacts.

**Occlusion handling:** In settings with very sparse camera setups, occlusion and registration artefacts become annoyingly obvious. Carranza *et al.* [CTMS03] therefore proposed to use a visibility map, computing visibility for every vertex of the mesh from several camera views that are slightly displaced in the image plane. Lensch *et al.* [LKG*03] search for depth discontinuities to discard samples close to them, as they are prone to errors. In Virtual Viewpoint Video [ZKU*04] these discontinuities are rendered separately using Bayesian image matting to compute fore- and background colours along with opacities. All of these approaches either assume very precise underlying geometry, or they need extensive pre-processing to achieve interactive rendering frame rates.

**Warping Techniques:** If complete correspondences between image pixels can be established, accurate image warping becomes possible [CW93, MB95]. Often, however, current methods for automatic camera calibration and depth acquisition are too imprecise. For very similar images, optical flow techniques have proven useful [HS81, LK81]. They can be employed to create smooth morphs between images, e.g. [VBK05].

The work with the closest resemblance to ours was proposed by Aliaga *et al.* in the context of creating virtual walkthroughs [AFYC02, AYFC03]. Aliaga *et al.* capture a very dense set of omnidirectional images, with an average distance between capturing positions of 4 cm. For image synthesis, they identify corresponding features in the different images, triangulate them in one reference view, and warp this mesh according to the current viewpoint and the interpolated feature positions. Our approach differs markedly in a number of important points. First, Floating Textures do not rely on previously tracked features which can be difficult to establish, especially for natural scenes. Second, we explicitly take occlusion effects into account, instead of relying on dense sampling. Third, instead of using only a sparse set of features to establish the necessary information for warping, we search for a globally optimal solution, which makes our approach less dependent on scene properties.

## 3. Problem Description

In a slightly simplified version, the plenoptic function $\mathbf{P}(x,y,z,\theta,\phi)$ describes radiance as a function of 3D position in space $(x,y,z)$ and direction $(\theta,\phi)$ [AB91]. The notion of image-based rendering now is to approximate the plenoptic function with a finite number of discrete samples of $\mathbf{P}$ for various $(x,y,z,\theta,\phi)$ and to efficiently re-create novel views from this representation by making use of some sort of object geometry proxy.

Any object surface that we choose to display can be described as a function $\mathbf{G} : (x,y,z,\theta,\phi) \rightarrow (x_o,y_o,z_o)$, i.e., by a mapping of viewing rays $(x,y,z,\theta,\phi)$ to 3D coordinates $(x_o,y_o,z_o)$ on the object's surface. Of course, the function $\mathbf{G}$ is only defined for rays hitting the object, but this is not crucial since one can simply discard the computation for all other viewing rays. With $\mathbf{G_O}$ we denote the function of the true surface of the object, and with $\mathbf{G_A}$ we denote a function that only approximates this surface, Fig. 2.

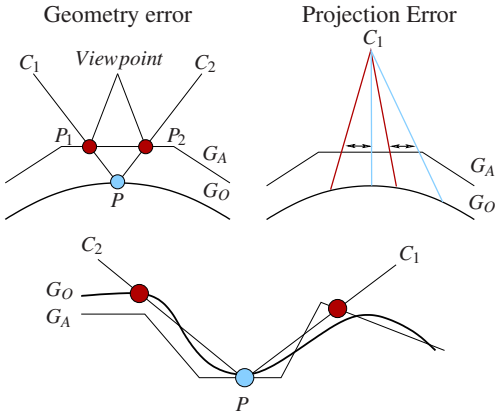


**Figure 2:** *Top left: Geometry inaccuracies cause ghosting artefacts. Point P on the original surface* $\mathbf{G_O}$ *is erroneously projected to 3D-position* $P_1$ *from camera* $C_1$ *and to 3D-position* $P_2$ *from camera* $C_2$ *if only the approximate geometry* $\mathbf{G_A}$ *is available. Top right: Small imprecisions in camera calibration can lead to false pixel projections (red lines, compared to correct projections displayed as blue lines). This leads to a texture shift on the object surface and subsequently to ghosting artefacts. Bottom: Visibility errors. Given only approximate geometry* $\mathbf{G_A}$*, point P is classified as being visible from* $C_2$ *and not visible from camera* $C_1$*. Given correct geometry* $\mathbf{G_O}$*, it is actually the reverse, resulting in false projections.*

Next, a variety of camera calibration techniques exist to establish the projection mapping $\mathbf{P}_i : (x,y,z) \rightarrow (s,t)$ which describes how any 3D point $(x,y,z)$ is mapped to its corresponding 2D-position $(s,t)$ in the $i$-th image. From its projected position $(s,t)$ in image $\mathbf{I}_i$, the 3D point's colour value $(r,g,b)$ can be read out, $\mathbf{I}_i : (s,t) \rightarrow (r,g,b)$. Then, any novel view $\mathbf{I}^V_{\text{Linear}}$ from a virtual viewpoint $V$ synthesized by a weighted linear interpolation scheme, as employed by most IBR systems, can be formulated as

$$\mathbf{I}^V_{\text{Linear}}(x,y,z,\theta,\phi) = \sum_i \mathbf{I}^V_i(x,y,z,\theta,\phi)\,\omega_i \qquad (1)$$

with

$$\mathbf{I}^V_i(x,y,z,\theta,\phi) = \mathbf{I}_i(\mathbf{P}_i(\mathbf{G_A}(x,y,z,\theta,\phi))) \qquad (2)$$

$$\omega_i = \delta_i(\mathbf{G_A}(x,y,z,\theta,\phi))\, w_i(x,y,z,\theta,\phi) \qquad (3)$$

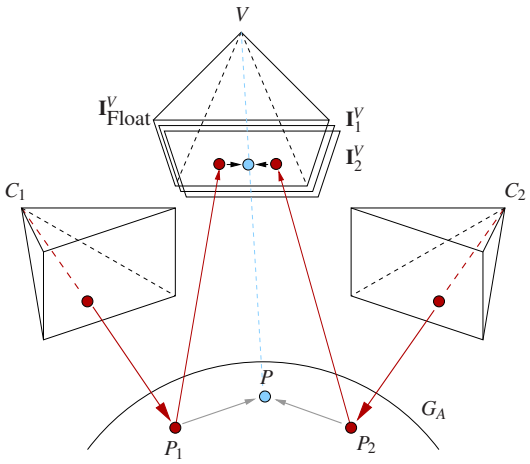


**Figure 3:** *Rendering with Floating Textures. The input photos are projected from camera positions $C_i$ onto the approximate geometry $\mathbf{G_A}$ and onto the desired image plane of viewpoint $V$. The resulting intermediate images $\mathbf{I}^V_i$ exhibit mismatch which is compensated by warping all $\mathbf{I}^V_i$ based on the optical flow to obtain the final image $\mathbf{I}^V_{Float}$.*

and $\sum_i \omega_i = 1$. The notation $\mathbf{I}^V_i$ is used to denote the image rendered for a viewpoint $V$ by projecting the input image $\mathbf{I}_i$ as texture onto $\mathbf{G_A}$. Note that $\delta_i$ is a visibility factor which is one if a point on the approximate surface $\mathbf{G_A}$ is visible by camera $i$, and zero otherwise. $w_i$ is the weighting function which determines the influence of camera $i$ for every viewing ray.

Note that (1) is the attempt to represent the plenoptic function as a linear combination of re-projected images. For several reasons, weighted linear interpolation cannot be relied on to reconstruct the correct values of the plenoptic function:

1. Typically, $\mathbf{G_O} \neq \mathbf{G_A}$ almost everywhere, so the input to (2) is already incorrect in most places, Fig. 2 top left.
2. Due to calibration errors, $\mathbf{P}_i$ is not exact, leading to projection deviations and, subsequently, erroneous colour values, Fig. 2 top right.
3. In any case, only visibility calculations based on the original geometry $\mathbf{G_O}$ can provide correct results. If only approximate geometry is available, visibility errors are bound to occur, Fig. 2 bottom.

In summary, in the presence of even small geometry inaccuracies or camera calibration imprecisions, a linear approach is not able to correctly interpolate from discrete image samples.

## 4. Floating Textures

In the following, we describe our approach to reduce blurring and ghosting artefacts caused by geometry and calibration inaccuracies using an adaptive, non-linear approach. In a nutshell, the notion of Floating Textures is to correct for local texture misalignments by determining the optical flow between projected textures and warping the textures accordingly in the rendered image domain. Both steps, optical flow estimation and multi-texture warping, can be efficiently implemented on graphics hardware to achieve interactive to real-time performance.

As input, the algorithm requires nothing more but a set of images, the corresponding, possibly imprecise, calibration data, and a geometry proxy. For simplicity, we will first assume an occlusion-free scene and describe how occlusion handling can be added in Sect. 4.2. Without occlusion, any novel viewpoint can, in theory, be rendered from the input images by warping. To determine the warp fields, we are safe to assume that corresponding pixels in different images have a set of similar properties, like colour or gradient constancy, so that the following property holds:

$$\mathbf{I}_j = \mathbf{W}_{\mathbf{I}_i \to \mathbf{I}_j} \circ \mathbf{I}_i \quad , \tag{4}$$

where $\mathbf{W}_{\mathbf{I}_i \to \mathbf{I}_j} \circ \mathbf{I}_i$ warps an image $\mathbf{I}_i$ towards $\mathbf{I}_j$ according to the warp field $\mathbf{W}_{\mathbf{I}_i \to \mathbf{I}_j}$. The problem of determining the warp field $\mathbf{W}_{\mathbf{I}_i \to \mathbf{I}_j}$ between two images $\mathbf{I}_i$, $\mathbf{I}_j$ is known as optical flow estimation [HS81, LK81]. If pixel distances between corresponding image features are not too large, algorithms to robustly estimate per-pixel optical flow are available [BBPW04].

For our Floating Textures approach, we propose a symbiosis of linear interpolation and optical flow-based warping. We first project the photographs from cameras $C_i$ onto the approximate geometry surface $\mathbf{G_A}$ and render the scene from the desired viewpoint $V$, creating the intermediate images $\mathbf{I}^V_i$, Fig. 3. Note that while corresponding image features do not yet exactly line up in $\mathbf{I}^V_i$ they are much closer together than in the original photos (disparity compensation). We can apply optical flow estimation to the intermediate images $\mathbf{I}^V_i$ to robustly determine the pairwise flow fields $\mathbf{W}_{\mathbf{I}^V_i \to \mathbf{I}^V_j}$.

To compensate for more than two input images, we linearily combine the flow fields according to (6), apply these to all intermediate images $\mathbf{I}^V_i$ and blend them to obtain the final rendering result $\mathbf{I}^V_{\text{Float}}$. To reduce computational cost, instead of establishing for $n$ input photos $(n-1)n$ flow fields, it often suffices to consider only the 3 closest input images to the current viewpoint. If more than 3 input images are needed, we can reduce the quadratic effort to linear complexity by using intermediate results, which we however did not incorporate in our current system.

We use an angular weighting scheme as proposed in [BBM*01, DTM96] because it has been found to yield better results for coarse geometry than weighting schemes based on normal vectors [CTMS03]. Our Floating Textures are, in fact, independent of the weighting scheme used as long as the different weights sum up to 1 for every pixel, which is necessesary to ensure that corresponding features

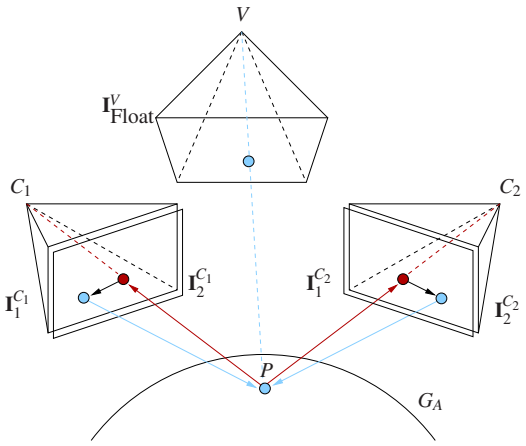


**Figure 4:** *Rendering with Floating Textures for static scenes. In a preprocessing step, the images $\mathbf{I}_j$ of every camera are projected onto the approximate surface $\mathbf{G_A}$ and into every other input camera $C_i$, resulting in the intermediate images $\mathbf{I}^{C_i}_j$. Then the warp fields between these images in every camera view is calculated. For rendering the image $\mathbf{I}^V_{Float}$ from viewpoint V, the warp field of each camera is queried for the texture coordinate offset (black arrows) of every rendered fragment, and the corrected texture value (blue dot in image plane) is projected back onto the object and into the novel view.*

coincide in the output image, and given a smooth change of camera influences if the virtual camera moves, otherwise snapping problems could occur.

The processing steps are summarized in the following functions and visualized in Fig. 3:

$$\mathbf{I}^V_{\text{Float}} = \sum_{i=1}^{n} (\mathbf{W}_{\mathbf{I}^V_i} \circ \mathbf{I}^V_i)\omega_i \tag{5}$$

$$\mathbf{W}_{\mathbf{I}^V_i} = \sum_{j=1}^{n} \omega_j \mathbf{W}_{\mathbf{I}^V_i \rightarrow \mathbf{I}^V_j} \tag{6}$$

$\mathbf{W}_{\mathbf{I}^V_i}$ is the combined flow field which is used for warping image $\mathbf{I}^V_i$. (5) is therefore an extension of (1) by additionally solving for the non-linear part in $\mathbf{P}$.

Note that our Floating Textures deliberately do not satisfy the epipolar constraint anymore. To make use of epipolar geometry constraints one has to presume perfectly calibrated cameras, which is seldom the case. Instead, by not relying on epipolar geometry, Floating Textures can handle imprecise camera calibration as well as approximate geometry.

### 4.1. Acceleration for Static Scenes

For static scenes it might seem unnecessary to re-compute the flow fields for every frame. But for a coarse geometry proxy, one cannot simply assign constant texture coordinates to every vertex and every input image. Instead, we propose a slight variation of our Floating Texture generation which can be computed during preprocessing.

Instead of computing flow fields between input images after they have been projected into the desired viewpoint, we render the scene from each camera position $C_i$ and project all other input images into its viewpoint, i.e, we render $\mathbf{I}^{C_i}_j$, $j \in \{1,\ldots,n\}$. The flow fields $\mathbf{W}_{\mathbf{I}^{C_i}_i \rightarrow \mathbf{I}^{C_i}_j}$ are then established between the rendered image $\mathbf{I}^{C_i}_i$ and every $\mathbf{I}^{C_i}_j$, with $j \neq i$. As the views from the cameras do not change over time for static scenes, image synthesis for a new viewpoint reduces to simple projective texturing using warped texture coordinates, Fig. 4:

$$I^V_{Float} = \sum_{i=1}^{n} ((\sum_{j=1}^{n} (\omega_j \mathbf{W}_{\mathbf{I}^{C_i}_i \rightarrow \mathbf{I}^{C_i}_j})) \circ \mathbf{I}^{C_i}_i)^V \omega_i \tag{7}$$

Note that in comparison to viewpoint-centered warping (5) rendering quality may be slightly reduced. On the other hand, the online rendering computations are reduced to two simple texture lookups per fragment and camera.

### 4.2. Soft Visibility

Up to now, we have assumed only occlusion-free situations, which is seldomly the case in real-world scenarios. Simple projection of imprecisely calibrated photos onto an approximate 3D geometry model typically cause unsatisfactory results in the vicinity of occlusion boundaries, Fig. 5 top left: texture information from occluding parts of the mesh project incorrectly onto other geometry parts. With respect to Floating Textures, this not only affects rendering quality but also the reliability of flow field estimation.

A common approach to handle the occlusion problem is to establish a binary visibility map for each camera, multiply it with the weight map, and normalize the weights afterwards so they sum up to one. This efficiently discards occluded pixels in the input cameras for texture generation [CTMS03, LKG*03]. One drawback of this approach is that it must be assumed that the underlying geometry is precise, and cameras are precisely calibrated. In the presence of coarse geometry, the usage of such binary visibility maps can create occlusion boundary artefacts at pixels where the value of the visibility map suddenly changes, Fig. 5 left column.

To counter these effects, we create a "soft" visibility map $\Omega$ for the current viewpoint and every input camera using a distance filter on the binary map:

$$\Omega(x,y) = \begin{cases} 0 & \text{if } \delta(x,y) = 0 \\ \frac{occDist(x,y)}{r} & \text{if } occDist(x,y) \leq r \\ 1 & \text{else} \end{cases} \tag{8}$$

Here $r$ is a user-defined radius, and $occDist(x,y)$ is the distance to the next occluded pixel. If $\Omega$ is multiplied with the

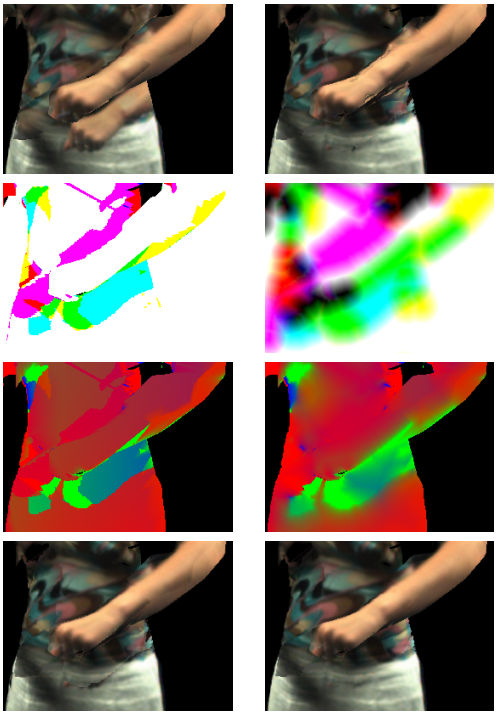

**Figure 5:** *Top left: Projection errors if occlusion is ignored. Top right: Optical flow estimation goes astray if occluded image regions are not properly filled. Second row, left: Visualization of a binary visibility map for three input cameras. Second row, right: Visualization of a soft visibility map for three input cameras. Third row, left: Weight map multiplied with the binary visibility map. Third row, right: Weight map multiplied with the soft visibility map; note that no sudden jumps of camera weights occur anymore between adjacent pixels. Bottom left: Final result after texture projection using weight map with binary visibility. Bottom right: Final result after texture projection using weight map with soft visibility. Note that most visible seams and false projections have been effectively removed.*

weight map, (8) makes sure that occluded regions stay occluded, while hard edges in the final weight map are removed. Using this "soft" visibility map the above mentioned occlusion artefacts effectively disappear, Fig. 5 bottom right.

To improve optical flow estimation, we fill occluded areas in the projected input images $\mathbf{I}_i^V$ with the corresponding colour values from that camera whose weight $\omega$ for this pixel is highest. Otherwise, the erroneously projected part could seriously influence the result of the Floating Texture output as wrong correspondences could be established, Fig. 5 top right. With hole filling, the quality of the flow calculation is strongly improved, Fig. 5 bottom right. One could also think about using already warped texture values from surrounding "secure" areas for iterative hole filling.

## 5. GPU Implementation

The following is a description of our complete algorithm for dynamic scenes. A block diagram is given in Fig. 6. The extension to static scenes is straightforward. We assume that camera parameters, input images and a geometry proxy is given. The geometry representation can be of almost arbitrary type, e.g., a triangle mesh, a voxel representation, or a depth map (even though correct occlusion handling with a single depth map is not always possible due to the 2.5*D* scene representation).

First, given a novel viewpoint, we query the closest camera positions. For sparse camera arrangements, we typically choose the 2 or 3 closest input images. We render the geometry model from the cameras' viewpoints into different depth buffers. These depth maps are then used to establish for each camera a binary visibility map for the current viewpoint, similar in spirit to [CTMS03]. We use these visibility maps as input to the soft visibility shader. The calculation of $\Omega$ can be efficiently implemented in a two-pass fragment shader. Next, a weight map is established by calculating the camera weights per output pixel. We use an angular weighting scheme similar to [BBM*01]. The final camera weights for each pixel in the output image are obtained by multiplying the weight map with the visbility map and normalizing it so that the weights sum up to 1.

To create the input images for the flow field calculation, we render the geometry proxy from the desired viewpoint several times into multiple render targets, in turn projecting each input photo onto the geometry. If the weight for a specific camera is 0 for a pixel, the colour from the input camera with the highest weight at this position is used instead.

To compute the optical flow between two images we rely on a GPU-optimized implementation of the optical flow technique by Brox *et al.* [BBPW04]. We found this algorithm not only very accurate but also quite robust to noise. Optical flow computation time depends on image resolution as well as on the amount of texture mismatch. Per rendered frame and three input images, we need to compute six flow fields. Even though this processing step is computationally expensive and takes approximately 90% of the rendering time, we still achieve between 5 and 24fps at $1024 \times 768$-pixel rendering resolution on an NVidia GeForce 8800GTX. Compared to the results of a CPU implementation presented in [BBPW04], we observe a speedup factor of 70. While we also experimented with faster optical flow algorithms, like a multi-scale implementation of the well known technique by Horn and Schunck [HS81], we found that results were not as satisfactory. Tests have shown that the limiting speed factor is, in fact, not computational load, but the high number of state changes necessary to compute the flow fields.

Once all needed computations have been carried out, we can combine the results in a final render pass, which warps and blends the projected images according to the weight map and flow fields.

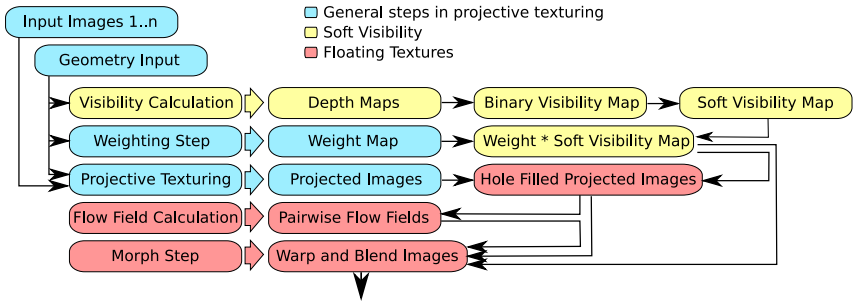


**Figure 6:** *Complete overview of our Floating Textures algorithm on GPU. See text for details.*

## 6. Results

To evaluate the proposed texturing approach, we have tested Floating Textures in conjunction with a number of different image-based rendering approaches. All tests were carried out using an OpenGL/GLSL implementation of our algorithm on a NVidia GeForce 8800GTX graphics card. Floating Texture frame rates vary between 5 and 24 fps, depending on the number of input images used and the amount of mismatch between textures which influences the number of iterations needed for the optical flow estimation algorithm to converge. The different IBR approaches with which we evaluated Floating Textures are (Figure 8, from top to bottom):

1. Synthetic Data Set: 49 input images synthesized from a textured 3D model, ground truth available;
2. Polyhedral Visual Hull Rendering: shape-from-silhouette reconstruction [FB03];
3. Free-Viewpoint Video: a parameter-fitted high-resolution 3D human model [CTMS03];
4. SurfCap: high-resolution geometry reconstructed using silhouette, feature, and stereo cues [SH07];
5. Light Field Rendering: a sub-sampled version of the Stanford Buddha light field data set [LH96].

Figure 7 depicts the corresponding geometry proxies for each IBR method. For each of these five different IBR techniques, we compared four different texturing approaches (Figure 8, from left to right):

1. Bandlimited Texturing [CCST00],
2. Filtered Blending [ESM07],
3. Unstructured Lumigraph Rendering [BBM*01], and
4. Floating Textures.

The viewpoint was chosen so that the angular distance to the used input views was maximized.

**Synthetic data set** The ground truth model of the Stanford Bunny consists of 65k triangles. As input images, we rendered 49 views from random directions applying a coloured checkerboard pattern as texture. We then reduced the mesh to 948 triangles to use it as coarse geometry proxy. Bandlimited reconstruction as well as Filtered Blending introduce considerable blurring along texture discontinuities. Unstructured Lumigraph rendering, on the other hand, leads to ghosting artefacts (aliasing). Floating Textures, instead, is able to compensate for most texture mismatch and generates a crisp texture.

**Polyhedral Visual Hull Rendering** We tested different texturing approaches for the exact polyhedral visual hull reconstruction approach by Franco and Boyer [FB03], both in an off-line setup as well as in a real-time "live" system. Our setup of the "live" system consists of 8 cameras with a resolution of 1024x786 pixels that are connected in pairs of two to 4 PCs (the camera nodes), arranged in an approximate quarter-dome. One PC is used to calculate the approximate geometry of the scene from silhouettes obtained from the camera nodes. The camera nodes calculate these silhouettes by downsampling the input images and performing background subtraction. The walls of the scene are covered in green cloth to facilitate this process. The approximate geometry is sent to a PC which renders the final image. The rendering algorithm takes the images from 3 cameras to texture the approximate geometry so only these three images are sent over the network to conserve bandwidth. The system allows to capture and render scenes at approximately 10 fps and $640 \times 480$-pixel output resolution. Even though the reconstructed visual hulls are only very approximate geometry models, the Floating Textures method is able to remove most of the ghosting artefacts prevailent in Unstructured Lumigraph rendering, Fig. 1.

In the offline acquisition setup, we recorded a dancer using eight cameras arranged in a full circle. Excessive blurring is the result if bandlimited and filtered blending are applied, Fig. 8. With a linear blending scheme, ghosting and projection errors degrade rendering quality of the face and at the shoulders. These artefacts are efficiently removed by Floating Textures without introducing any additional blur.

**Free-Viewpoint Video** For Free-Viewpoint Video acquisition, eight cameras are regularly spaced around a full circle [CTMS03]. Due to the cameras' far spacing, bandlimited and filtered blending eliminate all texture details, Fig.8. Since in Free-Viewpoint Video, a generic, 3D model is fit to the video streams by adapting only a good handful of

animation parameters, the model surface corresponds only approximately to the person's actual 3D geometry, even though model geometry is very detailed. This causes noticeable ghosting artefacts if linear blending schemes are applied. The Floating Textures approach corrects for the projective texture mismatch and yields well-defined facial details, Fig.8.

**SurfCap** This data set was kindly provided to us from the SurfCap: Surface Motion Capture project [SH07]. Again, eight cameras are regularly spaced all around a full circle. In computationally elaborate off-line processing, a highly tesselated, smooth geometry mesh is reconstructed, Fig. 7. Far camera spacing prevents bandlimited and filtered blending to preserve details. Even though the mesh consists of 260k+ triangles, ghosting and occlusion artefacts still degrade rendering quality if Unstructured Lumigraph rendering is applied. With Floating Textures, in contrast, virtually artefact-free rendering results are obtained.

**Light Field Rendering** We down-sampled the Stanford Buddha light field data set to $8 \times 8$ images. While bandlimited rendering indiscriminately blurs away all details, more details are preserved in filtered blending, Fig. 8. Unstructured Lumigraph rendering (which corresponds to quadralinear interpolation for light field rendering) introduces ghosting, as the assumption of dense sampling is violated. The simple planar proxy is not enough to focus the light rays. With Floating Textures, in contrast, we achieve rendering results that are visually almost indistinguishable from the ground-truth. By using the Floating Textures approach in conjunction with light field rendering, comparable rendering results are obtainable from considerably fewer input images.

## 7. Discussion & Conclusions

We have presented a new, general method to improve projective texturing using multi-view imagery in conjunction with some 3D geometry proxy. Our Floating Textures approach strongly reduces ghosting and occlusion artefacts and achieves improved rendering quality from coarser 3D geometry, fewer input images, and less accurate calibration. This saves memory, bandwidth, acquisition time, and money.

While we did not observe any problems during our evaluation experiments, it is obvious that strongly specular surfaces or badly colour-calibrated cameras will cause problems for the optical flow estimation. Also, if texture mismatch (ghosting) is too large, due to a very coarse geometry proxy or too few input images, the optical flow algorithm might not be able to find correct correspondences. But with more sophisticated, future optical flows techniques, the robustness of the Floating Textures will increase accordingly. In general, however, the results with Floating Textures will not be worse than linear interpolation schemes. In our Floating Textures method, we deliberately disregard the epipolar constraint and allow textures to locally "float" in all directions. This is not a bug but a feature: this way, Floating Textures can compensate for imprecise camera calibration, while the small shifting of the texture on the surface is visually completely imperceptible.

One source for artefacts remaining in rendering dynamic scenes is that of temporally incoherent geometry. In the future, we intend to investigate how Floating Textures might be extended to compensate for temporal inconsistencies of the geometry proxy. Finally, for highly reflective or transparent objects, motion layer decomposition promises to be another interesting research direction.

## 8. Acknowledgements

We would like to thank Jonathan Starck for providing us with the SurfCap test data (www.ee.surrey.ac.uk/CVSSP/VMRG/surfcap.htm). The Max-Planck-Institute für Informatik for letting us use their studio for the dance scene, and the EDM at the university of Hasselt for allowing us to use their real-time acquisition studio.

## References

[AB91] ADELSON E. H., BERGEN J. R.: The Plenoptic Function and the Elements of Early Vision. *M. Landy and J. A. Movshon, (eds) Computational Models of Visual Processing* (1991), 3–20.

[AFYC02] ALIAGA D. G., FUNKHOUSER T., YANOVSKY D., CARLBOM I.: Sea of images. In *VIS '02: Proceedings of the conference on Visualization '02* (Washington, DC, USA, 2002), IEEE Computer Society.

[AYFC03] ALIAGA D. G., YANOVSKY D., FUNKHOUSER T., CARLBOM I.: Interactive Image-Based Rendering Using Feature Globalization. In *I3D '03: Proceedings of the 2003 symposium on Interactive 3D graphics* (New York, NY, USA, 2003), ACM Press, pp. 163–170.

[Bau02] BAUMBERG A.: Blending images for texturing 3d models. In *Proceedings of the British Machine Vision Conference* (2002), Rosin P. L., Marshall A. D., (Eds.).

[BBM*01] BUEHLER C., BOSSE M., MCMILLAN L., GORTLER S., COHEN M.: Unstructured Lumigraph Rendering. In *Proc. SIGGRAPH '01* (2001), pp. 425–432.

[BBPW04] BROX T., BRUHN A., PAPENBERG N., WEICKERT J.: High accuracy optical flow estimation based on a theory for warping. In *Computer Vision - ECCV 2004, 8th European Conference on Computer Vision, Prague, Czech Republic, May 11-14, 2004. Proceedings, Part IV* (2004), pp. 25–36.

[Cat74] CATMULL E.: *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, Departement of Computer Sciences, University of Utah, 12 1974.

[CCST00] CHAI J.-X., CHAN S.-C., SHUM H.-Y., TONG X.: Plenoptic Sampling. In *Proc. SIGGRAPH '00* (2000), pp. 307–318.

[CTMS03] CARRANZA J., THEOBALT C., MAGNOR M., SEIDEL H. P.: Free-viewpoint video of human actors. In *Proc. SIGGRAPH '03* (2003), pp. 569–577.

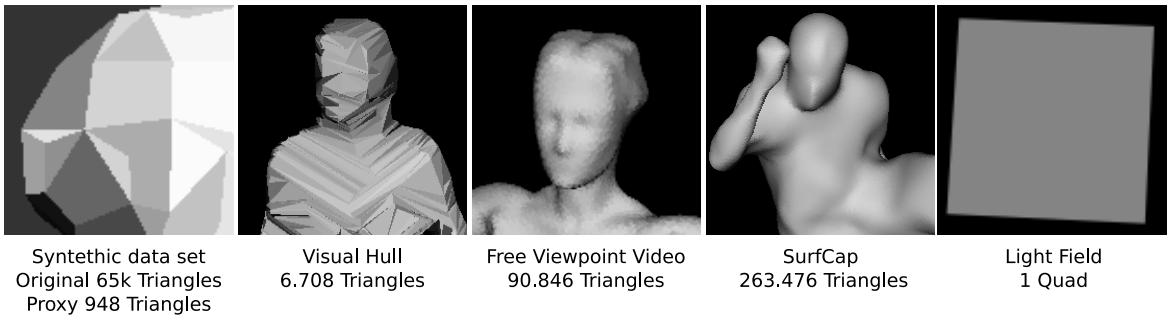


**Figure 7:** *Geometry proxies corresponding to the different IBR methods evaluated in our experiments, Fig. 8.*

[CW93] CHEN S. E., WILLIAMS L.: View Interpolation for Image Synthesis. *Proc. SIGGRAPH '93* (1993), 279–288.

[DTM96] DEBEVEC P. E., TAYLOR C. J., MALIK J.: Modeling and Rendering Architecture from Photographs: A Hybrid Geometry- and Image-Based Approach. In *Proc. SIGGRAPH '96* (1996), pp. 11–20.

[ESM07] EISEMANN M., SELLENT A., MAGNOR M.: Filtered Blending: A new, minimal Reconstruction Filter for Ghosting-Free Projective Texturing with Multiple Images. *Proc. Vision, Modeling, and Visualization (VMV'07), Saarbrücken, Germany* (11 2007), 119–126.

[FB03] FRANCO J.-S., BOYER E.: Exact polyhedral visual hulls. In *Proceedings of the Fourteenth British Machine Vision Conference* (September 2003), pp. 329–338. Norwich, UK.

[GGSC96] GORTLER S. J., GRZESZCZUK R., SZELISKI R., COHEN M. F.: The Lumigraph. In *Proc. SIGGRAPH '00* (1996), pp. 43–54.

[HS81] HORN B., SCHUNCK B.: Determining Optical Flow. *Artificial Intelligence 16*, 1–3 (August 1981), 185–203.

[IMG00] ISAKSEN A., MCMILLAN L., GORTLER S. J.: Dynamically Reparameterized Light Fields. In *Proc. SIGGRAPH '00* (2000), pp. 297–306.

[LH96] LEVOY M., HANRAHAN P.: Light Field Rendering. In *Proc. SIGGRAPH '96* (1996), pp. 31–42.

[LK81] LUCAS B., KANADE T.: An iterative image registration technique with an application to stereo vision. In *Proc. Seventh International Joint Conference on Artificial Intelligence* (1981), pp. 674–679.

[LKG*03] LENSCH H. P. A., KAUTZ J., GOESELE M., HEIDRICH W., SEIDEL H.-P.: Image-based reconstruction of spatial appearance and geometric detail. *ACM Trans. Graph. 22*, 2 (2003), 234–257.

[LS04] LIN Z., SHUM H.-Y.: A Geometric Analysis of Light Field Rendering. *International Journal of Computer Vision 58*, 2 (2004), 121–138.

[MB95] MCMILLAN L., BISHOP G.: Plenoptic Modeling: An Image-Based Rendering System. In *Proc. SIGGRAPH '95* (1995), pp. 39–46.

[MBR*00] MATUSIK W., BUEHLER C., RASKAR R., GORTLER S., MCMILLAN L.: Image-based visual hulls. In *Proc. SIGGRAPH '00* (2000), pp. 369–374.

[MP04] MATUSIK W., PFISTER H.: 3D TV: A scalable system for real-time acquisition, transmission, and autostereoscopic display of dynamic scenes. In *Proc. SIGGRAPH '04* (2004), pp. 814–824.

[RCMS99] ROCCHINI C., CIGNOMI P., MONTANI C., SCOPIGNO R.: Multiple textures stitching and blending on 3D objects. In *Proceedings of the Eurographics Workshop on Rendering* (1999), pp. 119–130.

[SH07] STARCK J., HILTON A.: Surface capture for performance based animation. *IEEE Computer Graphics and Applications 27*, 3 (2007), 21–31.

[SKvW*92] SEGAL M., KOROBKIN C., VAN WIDENFELT R., FORAN J., HAEBERLI P.: Fast Shadows and Lighting Effects using Texture Mapping. In *Proc. SIGGRAPH '92* (1992), pp. 249–252.

[SYGM03] STEWART J., YU J., GORTLER S. J., MCMILLAN L.: A New Reconstruction Filter for Undersampled Light Fields. In *Proceedings of the Eurographics Workshop on Rendering* (2003), pp. 150–156.

[VBK05] VEDULA S., BAKER S., KANADE T.: Image based spatio-temporal modeling and view interpolation of dynamic events. *ACM Transactions on Graphics 24*, 2 (April 2005), 240–261.

[WAA*00] WOOD D. N., AZUMA D. I., ALDINGER K., CURLESS B., DUCHAMP T., SALESIN D. H., STUETZLE W.: Surface light fields for 3d photography. In *Proc. SIGGRAPH '00* (2000), pp. 287–296.

[WWG07] WASCHBÜSCH M., WÜRMLIN S., GROSS M.: 3D Video Billboard Clouds. *Computer Graphics Forum (Proceedings of the Eurographics 2007) 26*, 3 (2007).

[ZKU*04] ZITNICK C., KANG S., UYTTENDAELE M., WINDER S., SZELISKI R.: High-quality video view interpolation using a layered representation. In *Proc. SIGGRAPH '04* (2004), pp. 600–608.

[ZWT*05] ZHOU K., WANG X., TONG Y., DESBRUN M., GUO B., SHUM H.-Y.: Texturemontage: Seamless texturing of arbitrary surfaces from multiple images. In *Proc. SIGGRAPH '05* (2005), pp. 1148–1155.

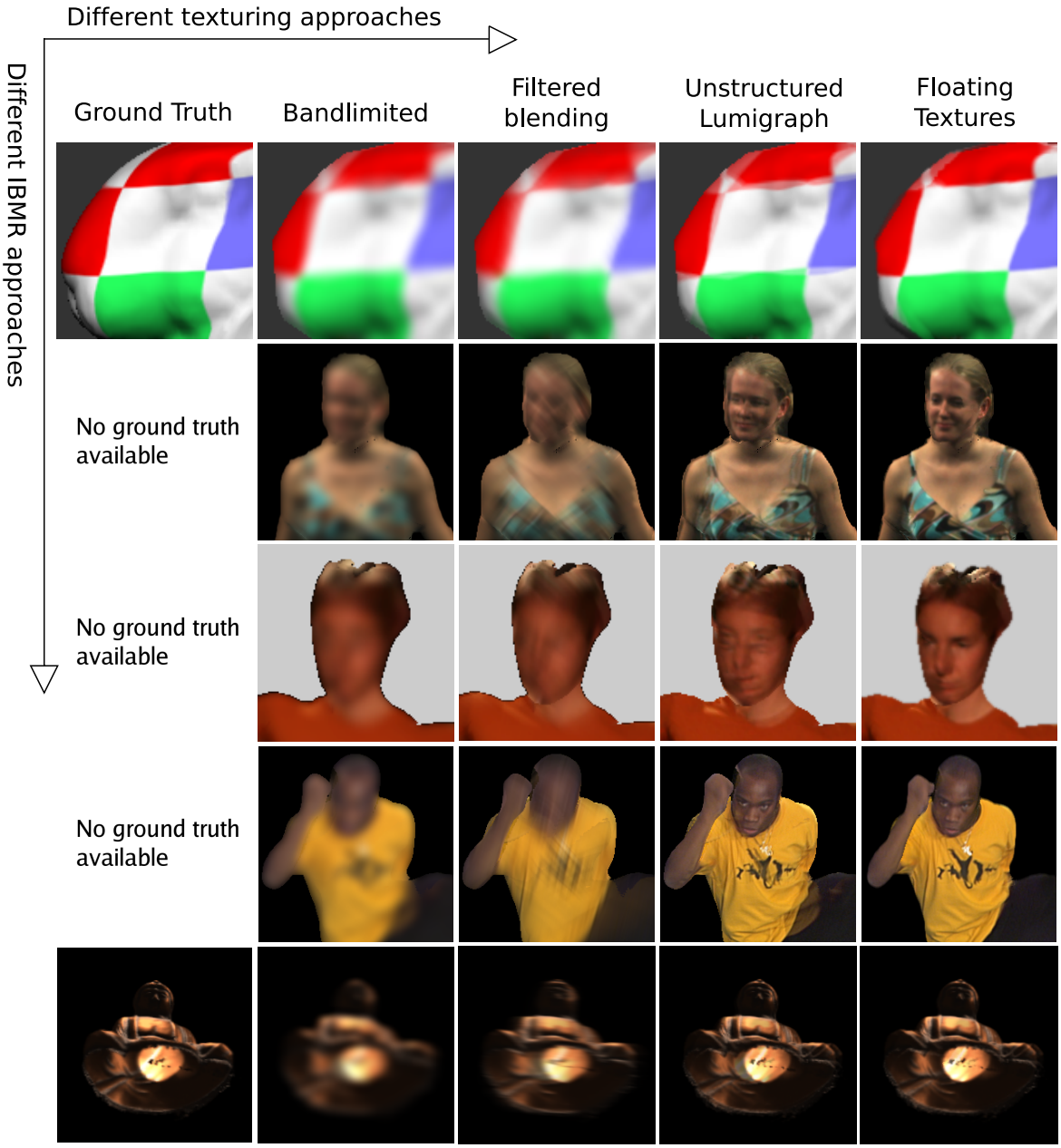


**Figure 8:** *Comparison of different texturing schemes in conjunction with a number of IBMR approaches. From left to right: Ground truth image (where available), bandlimited reconstruction [CCST00], Filtered Blending [ESM07], Unstructured Lumigraph Rendering [BBM*01], and Floating Textures. The different IBMR methods are (from top to bottom): Synthetic data set, Polyhedral Visual Hull Rendering [FB03], Free-Viewpoint Video [CTMS03], SurfCap [SH07], and Light Field Rendering [LH96].*