Real-time, Free-viewpoint Video Rendering from Volumetric Geometry

Bastian Goldlücke and Marcus Magnor

Max-Planck-Institut für Informatik
Graphics - Optics - Vision
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany
Email: {bg,magnor}@mpii.de

ABSTRACT

The aim of this work is to render high-quality views of a dynamic scene from novel viewpoints in real-time. An online system available at our institute computes the visual hull as a geometry proxy to guide the rendering at interactive rates. Because only a sparse set of cameras distributed around the scene is used to record the scene, only an coarse model of the scene geometry can be recovered. To alleviate this problem, we render textured billboards defined by the voxel model of the visual hull, preserving details in the source images while achieving excellent performance. By exploiting multi-texturing capabilities of modern graphics hardware, real-time frame rates are attained. Our algorithm can be used as part of an inexpensive system to display 3D-videos, or ultimately even in live 3D-television. The user is able to watch the scene from an arbitrary viewpoint chosen interactively.

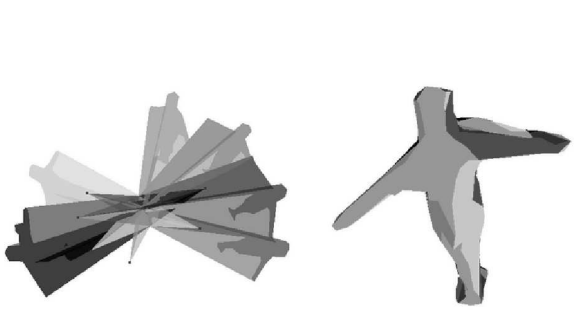
1. INTRODUCTION

The ultimate goal in the investigation of three-dimensional television is free-viewpoint TV: A scene is recorded by several cameras simultaneously, and the user can watch this scene from an arbitrary viewpoint chosen interactively. Nowadays the required building blocks of such a system - affordable recording hardware, robust computer vision algorithms, and sophisticated rendering techniques - become more and more available. Consequently, a lot of research activity can be observed in this exciting field.[1–3]

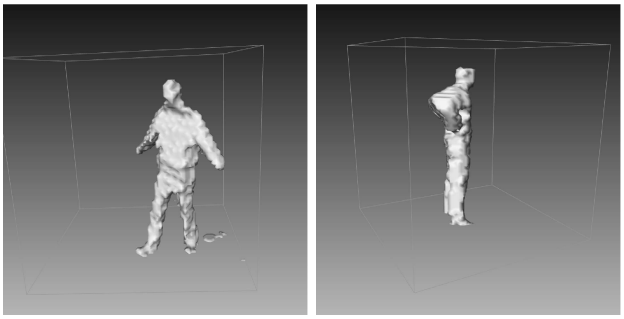
Naturally, rendering techniques employed in 3D-TV are closely related to the field of image-based rendering (IBR). In IBR, the *light field* of a scene, i.e. all rays of light emerging from any point in space into any given direction, is sampled using several conventional photographs. Given enough samples from different viewpoints, one can reconstruct arbitrary views of the scene from outside the convex hull of the recording positions.[4] One major problem is that a very large number of samples is necessary to attain convincing rendering results.[5] A way to reduce this number is to employ computer vision algorithms to reconstruct 3D scene structure. Hybrid model/image-based rendering methods based on the visual hull,[6] per-image depth maps[7] or even a complete 3D scene geometry model[8] achieve realistic rendering results from only a relatively small number of images.

Another challenge requiring considerable hardware effort is to record dynamic light fields and reconstruct 3D models at interactive frame rates. Multiple synchronized video cameras are needed to capture the scene from different viewpoints. An off-line approach based on volumetric reconstruction is presented in[9] where a dynamic voxel model is used to render the scene from novel viewpoints.

This paper focuses on the rendering part and presents a hardware-accelerated algorithm to efficiently render novel views of a person moving through a scene from arbitrary viewpoints. It is akin to the *Microfacet Billboarding* approach of Yamazaki et al,[10] but in contrast to it, only the streamed data from several synchronized, calibrated cameras together with a voxel model of the visual hull is used as an input to the algorithm. The computation of this model as an approximation to scene geometry is discussed in Sect. 2. The visual hull can be computed interactively using a recording system built at our institute, which we introduce in Sect.3. Sect. 4 is devoted to our novel rendering algorithm, whose results are presented in Sect. 5. We conclude with some plans for future work in Sect. 6.



(a) The silhouettes in the source images define generalized cones in space. Their intersection is the visual hull.



(b) The voxel volume defined by the visual hull and its bounding box for two different frames.

Figure 1. *Construction of the visual hull and the corresponding voxel volume.*

2. VISUAL HULL RECONSTRUCTION

We use the visual hull as an approximate geometric model of the foreground objects in the scene. The concept of the visual hull was introduced by Laurentini[11] to characterize the best geometry approximation that can be achieved using a *shape-from-silhouette* reconstruction method. These methods exploit the fact that the silhouette of an object in an arbitrary view reprojects onto a generalized cone in 3D-space, Fig. 1(a). All the space occupied by the object must lie somewhere inside this cone. If we intersect all cones from all possible views, we obtain a conservative estimate of the object's geometry, which is called the *visual hull*. In real-world applications, it is of course only possible to intersect cones from a finite number of source images, thus obtaining an approximate visual hull.

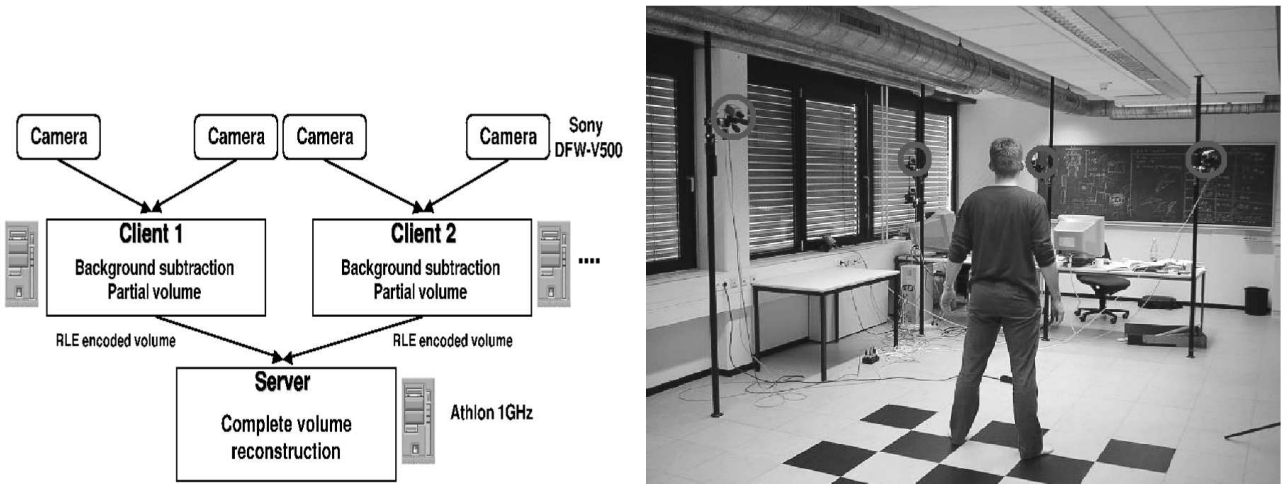
An important first step before the visual hull can be computed is the extraction of the foreground object silhouettes from the source images. We assume that the background of the scene is static to be able to employ a method based on image statistics originally proposed by Cheung et al.[12] This algorithm computes color mean and standard deviation of each background pixel from a sequence of images with no foreground object present. In the source frames, foreground objects are then identified by a large deviation from their background statistics. A pixel is classified as foreground if it differs in at least one color channel by more than an upper threshold factor τ from the background distribution,

$$\|p(x,y) - \mu(x,y)\| \geq \tau \cdot \sigma(x,y).$$

One principal problem remains: If no additional criterion is applied, shadow pixels are wrongly classified as foreground. We solve this problem by characterizing shadows as the set of pixels with a large intensity difference compared to the background, but only a small difference in hue. Pixels classified as shadow are removed from the foreground silhouette.

After the background subtraction, our system approximates the visual hull in the form of a cubic binary voxel volume, where a voxel value of 1 indicates that the corresponding scene point might be occupied. We assume that the foreground objects are contained in a cubic bounding box uniformly subdivided into voxels. The computation of the visual hull is then performed as follows: Let $C_i, i = 0, \ldots, N$ be a number of cameras. Consider an object which is projected onto the silhouettes S_i by the respective cameras. We assume that all cameras are fully calibrated, in particular we know their projection matrices. For each voxel v in the volume and each camera C_i, we can therefore compute the projection v_i in the camera's image. Note that because v is a cube, in general v_i is a polygon. If $v_i \subset S_i$ for all i, the voxel v is marked as occupied, otherwise it is marked as empty. Two example results are shown in Fig. 1(b).

A distributed client-server system which computes the visual hull in real-time using the above algorithm is presented in the next section.



(a) System architecture.

(b) A photo of the vision studio.

Figure 2. *The system used for video acquisition and online visual hull reconstruction.* The image-based visual hull is reconstructed online using several clients to compute partial volumes, which are combined on a server into the final voxel model of the scene.

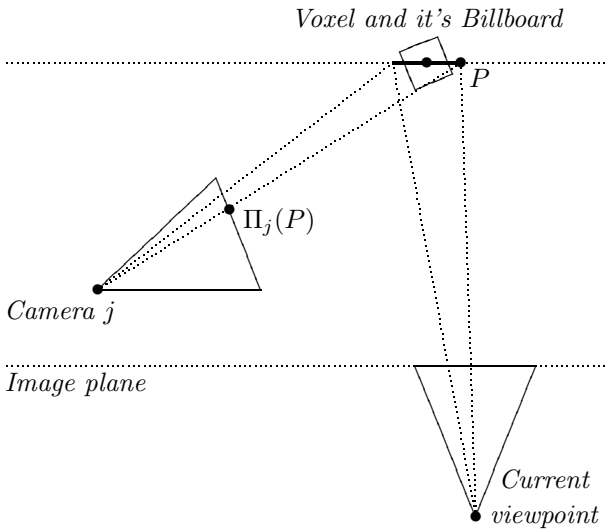
3. VIDEO ACQUISITION HARDWARE

The system acquires synchronized video streams via pairs of cameras connected to client PCs, Fig. 2. Each client consists of a 1GHz single processor Athlon PC connected to two Sony DFW-V500 IEEE1394 video cameras that run at a resolution of 320×240 pixels in RGB color mode. For synchronization of the cameras, a control box was built that distributes an external trigger pulse from the parallel port of a host computer to all 6 cameras. The clients are connected to a 1 GHz server PC by a 100 MBit/s Ethernet network.

For our application, the internal and external camera parameters of each of the imaging devices must be known. To achieve this, a calibration procedure based on Tsai's algorithm is applied.[13] The external camera parameters are estimated using a 2x2 m checkerboard calibration pattern, which also defines the bounding box of the voxel volume. The corners of the pattern are detected automatically by employing a sub-pixel corner detection algorithm on the camera images showing the pattern. The internal camera parameters can in theory also be calculated from the large pattern using Tsai's calibration method, but we achieve a more accurate calibration of the internal parameters by using a small checkerboard pattern that is positioned to cover a large part of each camera view.

Each of the clients separates the foreground from the known static background of the scene for both of the cameras connected to it. Using the scheme described in the previous section, it computes the partial visual hulls from the two silhouettes obtained from the background subtraction. The partial voxel model is then RLE-encoded and transmitted to the server PC, which intersects the partial models to obtain the final voxel model of the visual hull. The server also sends the trigger signals to the cameras for synchronization. The whole architecture scales easily to more cameras and more clients.

The system was designed and built for online human motion capture.[14] It is capable of online performance and computes the voxel-based visual hull at a rate of 15 frames per second. The input to our rendering algorithm in each frame consists of the current camera images and the voxel model of the visual hull. We also pre-compute the visibility, i.e. whether a voxel is visible from a given camera or not.



(a) The billboard associated to a voxel lies parallel to the image plane. It's corner P has the texture coordinate $\Pi_j(P)$ in the image of camera j. The projection Π_j can be pre-computed.

State	Value
General	
BlendEquation	FUNC_ADD
BlendFunc	SRC_ALPHA, ONE_MINUS_SRC_ALPHA
Texture unit 1	
SHADER_OPERATION	TEXTURE_3D
TEXTURE_ENV_MODE	NONE
Texture unit 2	
SHADER_OPERATION	DEPENDENT_AR_ _TEXTURE_2D_NV
TEXTURE_ENV_MODE	COMBINE
COMBINE_RGB	REPLACE
SOURCE0_RGB	TEXTURE
OPERAND0_RGB	SRC_COLOR
COMBINE_ALPHA	REPLACE
SOURCE0_ALPHA	PREVIOUS
OPERAND0_ALPHA	SRC_ALPHA
SOURCE1_ALPHA	TEXTURE
OPERAND1_ALPHA	SRC_ALPHA

(b) OpenGL and texture stage state during billboard rendering.

Figure 3. *Illustration of billboard texturing and the graphics hardware configuration*

4. HARDWARE ACCELERATED RENDERING

For each occupied voxel we render a single geometric primitive called a billboard, which is a rectangle parallel to the image plane having the same center as the voxel. Since the coordinates of the billboard's corners in 3D-space are known, we can compute their locations in the camera views and use this information to texture the billboards, Fig. 3(a). The cameras are immobile, so this projection Π can be pre-computed and evaluated very efficiently using hardware-accelerated dependent texturing and customizable vertex transformations. In the remainder of the section we describe the texture setup in more detail.

For the sake of simplicity of notation we assume that the voxel volume lies in the unit cube $[0,1]^3$. The formulas can easily be adapted to arbitraty positions by inserting additional transformations at the appropriate locations. In a preprocessing step, the unit cube is divided into s^3 smaller cubes. The 3D textures T_i^Π which discretize the mappings Π_i will be of size $s \times s \times s$ and are defined for the centers of the cubes. For each camera i and each cube, we compute $\Pi_i(P)$ for its center P, encode the resulting homogenous 2D texture coordinate into the alpha and red channel of a texture and store them at the location P of the current 3D texture T_i^Π. This initial setup needs to be performed only once.

For each frame in the sequence, the images I_i currently viewed by the cameras are retrieved from hard disk and loaded as texture images T_i^I onto the graphics card. Since we want to use for texturing only those pixels which were used to construct the voxel volume, we assign to all other pixels an alpha value of zero denoting full transparency, while an alpha value of one is assigned to all silhouette pixels. In order to smooth the blending in areas where the texture is in transition from one camera image to the next, an alpha value of 0.5 is assigned to pixels belonging to the boundary of the silhouettes.

During the last step the occupied voxel volume is rendered. This is the only part of the algorithm which depends on the current viewing direction. For each voxel of diameter d centered on V, we select two cameras j and k according to the following criteria:

- Any pixel of the voxel is visible in I_j and I_k, and
- The angles α_j and α_k between the viewing direction and the optical axes of cameras j and k, respectively, are minimal.

Note that in general, the selection of the two different cameras depends on V, so different voxels are textured by different cameras. The voxel will be textured with the images of these two cameras, which are blended depending on the similarity of the camera's optical axes to the current viewing direction. The blending weights ω_j and ω_k for the cameras are set to

$$\omega_{j,k} \;:=\; 1 - \frac{\alpha_{j,k}}{\alpha_j + \alpha_k}.$$

This way, a camera's image is reproduced exactly if the viewing direction coincides with its optical axis, and transitions are reasonably smooth when the selection of the two cameras changes due to a change in viewing direction – although not perfectly so, since to obtain smooth transitions everywhere requires knowledge about all the boundary lines between different camera selections, which is too much computational effort to determine.

Rendering the voxel requires two passes and at least two hardware texture units. In the first pass, the texture T_j^{Π} is set up in texture unit 1 with dependent texturing enabled. Unit 1 computes texture coordinates for texture unit 2, to which we assign the camera image T_j^I. Blending is enabled to ensure that background pixels are transparent, and voxels have to be drawn in back-to-front order to correctly blend them one upon the other. The geometry transferred for the voxel is a single billboard centered on V parallel to the current image plane and of size $\sqrt{3}d \times \sqrt{3}d$ in order to cover the projection of the voxel with the projection of the billboard in the worst case. With texture coordinates set equal to the vertex positions, the graphics hardware now takes care of the rest.

Similarly, in the second pass, T_k^{Π} in unit 1 outputs texture coordinates for T_k^I in unit 2. We now have to blend correctly between the contribution of the first camera and this one, so the blending weight ω_k of the second image relative to the first one is stored in the alpha channel of the current color, which is modulated by the output of texture unit 2. The billboard is then rendered again. The correct setup for the texture shaders and environments in an example implementation using nVidia's OpenGL extension `NV_texture_shader` is summarized in Fig. 3(b). This setup can also be used for the first pass with the primary color's alpha value set to one, so no time-consuming state changes are required.

One also has to take great care to minimize the number of changes in texture images to optimize caching. In order to achieve this, a slight modification of the above scheme can be applied: The multiple textures for the source images are pasted on top of each other into one large single texture, where the v coordinate is translated depending on the current camera. The same is done with the textures for the mappings. That way, texture images have to be selected just once before rendering all of the geometry.

5. RESULTS

In our method, besides the negligible amount of geometry, data transfer from memory to the graphics card is limited to six very efficient texture loads per frame, one for each camera image. This amounts to 1.8MB of raw data in the case of 320 x 240 RGBA source images. It might be further reduced by first selecting the cameras which are able to contribute to the scene depending on the current viewpoint. However, it does not appear too much in view of the AGP 4x peak bandwidth, which lies in regions of 1GB per second. Instead, retrieving the data from hard drive fast enough is the bottleneck and requires a sophisticated compression scheme.

After all data has been transferred, our implementation is able to render a $64 \times 64 \times 64$ voxel volume from an arbitrary viewpoint at 30 fps. For each voxel, the contributions from the two nearest cameras are blended. The time to render each frame scales linearly with the number of visible voxels and almost linearly with the number of source cameras - note that the projected coordinates of each voxel have to be computed only once, regardless

(a) *Source images.* These are four of the six silhouettes used to construct the visual hull and texture the billboards. The cameras are spaced roughly 60 degrees apart.

(b) *Rendered view.* The novel viewpoint lies directly in between two of the source cameras in Fig. 4(a).

Figure 4. Input silhouette images and result of the rendering algorithm.

of the number of cameras. The program runs on a 1.8GHz Pentium IV Xeon with a GeForce 4 Ti 4600 graphics card. If no dependent texturing is available on a system, the mapping from 3D-space to the camera images can also be computed for each vertex by the CPU or a vertex program, decreasing performance slightly. The only real drawback of the algorithm is the limited resolution of the texture color channels: Since image coordinates are currently converted into 8-bit values, the discretized range of Π consists of only 256×256 distinct points. Higher image resolution gives more detail, nevertheless, because coordinates in-between arise from bilinear interpolation on the graphics card. Furthermore, already announced graphics hardware will support textures with higher dynamic range, probably up to floating-point accuracy, thus nullifying the problem.

Fig. 4(b) displays a rendered image of the person from a novel viewpoint directly in between two of the source cameras having an angular distance of about 60 degrees. Note that despite the low resolution of $64 \times 64 \times 64$ voxels, much finer details are visible. There are also few noticable transitions between voxels textured by different cameras, and the overall sharpness compared in view of sharpness and resolution of the source images in Fig. 4(a) is good.

It is important to note that although we use a voxel model of the visual hull to approximate the geometry, the actual rendering algorithm is not limited to this representation. Instead, it can use a polygonal model of the visual hull using exactly the same texture setup, or in fact almost any other geometry proxy.

6. CONCLUSIONS

The algorithm we presented employs a voxel-based representation of the visual hull as a geometry proxy for fast image-based rendering. By splatting small texture patches onto rectangular billboards, intricate details in the source images are preserved even in the case of coarse geometry data. The rendering algorithm can also handle other approximations to the scene geometry besides the visual hull. Real-time frame-rates of 30 Hz are achieved on current off-the-shelf hardware for $64 \times 64 \times 64$ voxel volumes.

Our ultimate goal are applications in free-viewpoint television. The next step in this direction are novel compression methods which are especially tuned to the data required for the real-time rendering. We are also working on a portable vision studio to record real-world outdoor scenes.

REFERENCES

1. M. O. de Beeck and A. Redert, "Three dimensional video for the home," *Proc. EUROIMAGE International Conference on Augmented, Virtual Environments and Three-Dimensional Imaging (ICAV3D'01),* Mykonos, Greece , pp. 188–191, May 2001.
2. B. Wilburn, M. Smulski, K. Lee, and M. Horowitz, "The light field video camera," *SPIE Proc. Media Processors 2002* **4674**, Jan. 2002.
3. S. Würmlin, E. Lamboray, O. Staadt, and M. Gross, "3D video recorder," in *Proceedings of Pacific Graphics*, pp. 10–22, 2002.
4. M. Levoy and P. Hanrahan, "Light field rendering," *Proc. ACM Conference on Computer Graphics (SIGGRAPH'96),* New Orleans, USA , pp. 31–42, Aug. 1996.
5. J.-X. Chai, X. Tong, S.-C. Chan, and H.-Y. Shum, "Plenoptic sampling," *Proc. ACM Conference on Computer Graphics (SIGGRAPH-2000),* New Orleans, USA , pp. 307–318, July 2000.
6. W. Matusik, C. Buehler, R. Raskar, S. Gortler, and L. McMillan, "Image-based visual hulls," in *Proceedings of ACM SIGGRAPH*, pp. 369–374, 2000.
7. S. Gortler, R. Grzeszczuk, R. Szeliski, and M. Cohen, "The lumigraph," *Proc. ACM Conference on Computer Graphics (SIGGRAPH'96),* New Orleans, USA , pp. 43–54, Aug. 1996.
8. D. Wood, D. Azuma, K. Aldinger, B. Curless, T. Duchamp, D. Salesin, and W. Stuetzle, "Surface light fields for 3D photography," *Proc. ACM Conference on Computer Graphics (SIGGRAPH-2000),* New Orleans, USA , pp. 287–296, July 2000.
9. S. Vedula, S. Baker, and T. Kanade, "Spatio-temporal view interpolation," Tech. Rep. CMU-RI-TR-01-35, Carnegie Mellon University, Sept. 2001.
10. S. Yamazaki, R. Sagawa, H. Kawasaki, K. Ikeuchi, and M. Sakauchi, "Microfacet billboarding," in *Proceedings of the 13th Eurographics Workshop on Rendering*, pp. 175–186, 2002.
11. A. Laurentini, "The visual hull concept for silhouette-based image understanding," *IEEE Transactions on Pattern Analysis and Machine Recognition* **16**, pp. 150–162, Feb. 1994.
12. K. Cheung, T. Kanade, J.-Y. Bouget, and M. Holler, "A real time system for robust 3D voxel reconstruction of human motions," in *Proc. of CVPR*, **2**, pp. 714–720, June 2000.
13. R. Tsai, "An efficient and accurate camera calibration technique for 3D machine vision," in *Proc. of CVPR*, **2**, pp. 364–374, June 1986.
14. C. Theobalt, M. Magnor, P. Schueler, and H.-P. Seidel, "Combining 2D feature tracking and volume reconstruction for online video-based human motion capture," in *Proceedings of Pacific Graphics 2002*, pp. 96–103, 2002.