In: Journal of Electronic Imaging, Volume 25, Issue 4, 7 July 2016 Ober-Gecks2016

# Efficient GPU-based Voxel Carving for Surveillance

**Antje Ober-Gecks**[a,*], **Marius Zwicker**[a], **Dominik Henrich**[a]
[a]University of Bayreuth, Institute for Computer Science, Chair for Applied Computer Science III, Bayreuth, Germany, 95440

**Abstract.** In this paper a GPU-based implementation of a space carving method for the reconstruction of the photo hull is presented. In particular, the generalized voxel coloring with item buffer (GVC-IB) approach is transferred to the GPU. The fast computation on the GPU is realized by an incrementally calculated standard deviation within the likelihood ratio test, which is applied as color consistency criterion. A fast and efficient computation of complete voxel-pixel projections is provided using volume rendering methods. This generates a speedup of the iterative carving procedure while considering all given pixel color information. Different volume rendering methods such as texture mapping and raycasting are examined. The termination of the voxel carving procedure is controlled through an anytime concept. The photo hull algorithm is examined for its applicability to real-world surveillance scenarios as online reconstruction method. For this reason, additionally a GPU-based redesign of a visual hull algorithm is provided, that utilizes geometric knowledge about known static occluders of the scene in order to create a conservative and complete visual hull that includes all given objects. This visual hull approximation serves as input for the photo hull algorithm.

**Keywords:** voxel carving, space carving, photo hull, conservative visual hull, occlusions, online reconstruction, GPU, surveillance.

*Antje Ober-Gecks, antje.ober-gecks@uni-bayreuth.de

## 1 Introduction

The capturing of real-world objects with the aid of sensors and the creation of computer-based 3D models is a well-established task and becomes increasingly more important with a growing range of applications. The main purposes of modeling 3D objects' geometries and photo-realistic surface properties are the visualization of information, the support in computer-aided 3D planning and designing as well as the acquisition of knowledge from the environment. The manual creation of such models (by hand) is expensive and time-consuming, thus an automatic capturing often is desired. Our application of interest is the reconstruction of (industrial) work cells. We want to determine the occupied space for robot tasks like path planning and collision detection. Also we aim at tracking humans in order to robustly capture their localization and pose in the surveillance volume, e.g. to derive knowledge about their behavior that might be helpful in human-robot cooperations. Current

sensors can only capture objects partially. Thus, several views of an object are required to gain sufficient information about its geometry and surface properties to be reconstructed. 3D reconstruction algorithms process captured sensor data and perform a sensor data fusion. There are two principles to implement this. One way is to use a single sensor and produce several images subsequently from different views of the object. The other way is to use a network of several sensors that work in parallel and capture data simultaneously, from different views and positions in space. The latter is more adequate for online reconstruction purposes with strong limitations to the run-time. The choice of sensor technology is a matter of the application as well as advantages and disadvantages concerning resolution, cost, measurement principle (passive or active) and other properties. For instance, active sensors such as laser scanners, sonar, structured light (e.g. the original Kinect sensor[1]), time-of-flight cameras or PMD cameras have a susceptibility towards reflective (metallic) surfaces. Other factors such as interference, absorption, sensing range, energy consumption, etc. also have to be considered. Thus passive sensors might still be preferred, as in our work. Our system consists of multiple calibrated and synchronized color cameras, which monitor a common surveillance volume. The calibration[2] is easily accomplished by capturing light points at different locations in the dark surveillance volume that were used as input for the calibration method to produce a linear system that is over determined. An approximative solution (of extrinsic and intrinsic parameters) is provided by an iterative error minimizing process. A mapping to our known world coordinate system is conducted with known points of a special registration object. This also produces a correct scaling parameter for the camera positions.

The surveillance volume contains known obstacles such as tables and racks, as shown in Figure (Fig.) 1, which are modeled as triangle meshes. Humans moving inside the surveillance volume are a priori unknown and called objects. The goal is a 3D online reconstruction of the scene, espe-

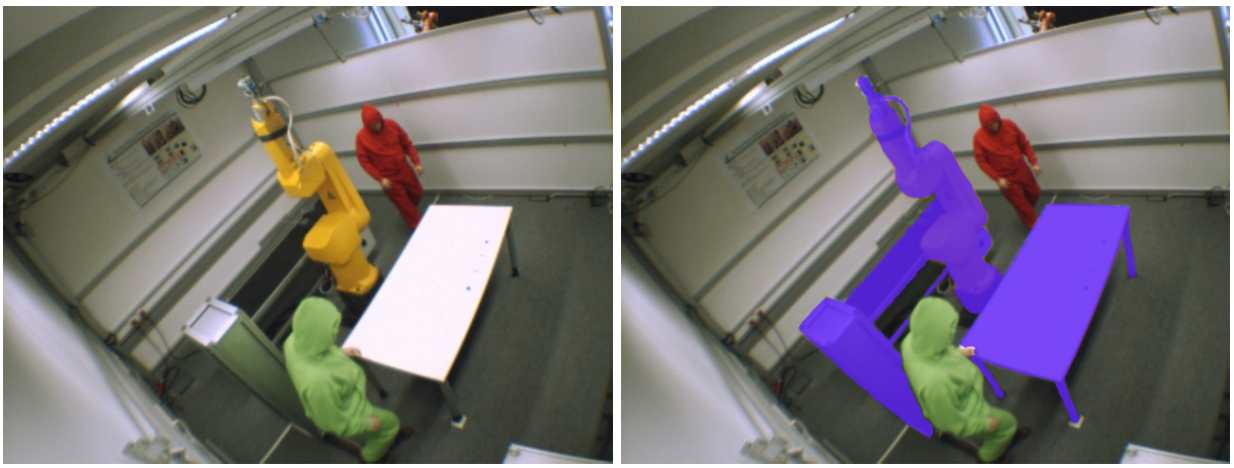

**Fig 1** Work cell with unknown objects (humans) and known static obstacles (colored violet) that cause occlusions.

cially of the objects, to obtain a good approximation of the objects' geometries as well as a good colorization of their surfaces. In this work, we provide two algorithms for the accelerated computation of a visual hull and a photo hull under consideration of occlusions caused by obstacles. A shorter form of these algorithms has been published lately,[3] but this work provides an improved and more precise presentation with additional images.

### *1.1 Overview*

We present two GPU algorithms for an accelerated reconstruction of a photo hull and a visual hull that incorporates all available pixel information to gain high exactness in computation with little loss of information. For this reason more quality can be expected concerning the reconstruction results than with several other existing methods. Moreover, knowledge about obstacles is included, so that the algorithms can be applied as online reconstruction method for surveillance scenes with occlusions. The visual hull serves as input for the photo hull. It provides an initial upper bound of the reconstruction volume and thus enables a better run-time for computing the photo hull. At every time step $t$, the visual hull is reconstructed from silhouette images that result from segmenting all camera images with a background subtraction method. A segmentation also is required for the

photo hull reconstruction as the equally colored walls of the work cell (see Fig. 1) lead to a premature termination of the algorithm otherwise (due to undesirable color consistencies). We apply a specific exact and conservative concept for a visual hull that handles the occlusions of the obstacles from the scene so that the reconstruction contains the objects completely. We published this concept in a former work.[4] In Section (Sec.) 3.1.4 our new algorithm of that visual hull is provided for a faster computation on the GPU.

For reconstructing an online photo hull, in Sec. 3.2 we present an adapted algorithm of the so called generalized voxel coloring with item buffer (GVC-IB).[5] To enable the transfer of the GVC-IB method to the GPU convenient modifications are applied, e.g. the incorporation of an incremental computation of the color consistency criterion as well as the usage of the anytime concept[6] in order to enforce a termination of the algorithm after a defined computation time $T$. One main issue that demands the utility of the GPU for both reconstruction methods is the fast and efficient computation of the complete voxel-pixel projections. With a higher number of cameras at high resolution, the online computation of the projections is too slow and an offline computation cannot be stored in the host's memory (at present). Therefore, different rendering techniques (texture mapping, raycasting) are examined. Furthermore, the computation time for the complete processing pipeline is analyzed as well as the quality of the resulting photo hull. The experimental evaluation, described in Sec. 4, is conducted with image sequences from a real-world work cell and from a simulation. Our algorithms are provided in Sec. 3. The results are summarized in Sec. 5. The following Sec. 2 gives an overview of related work.

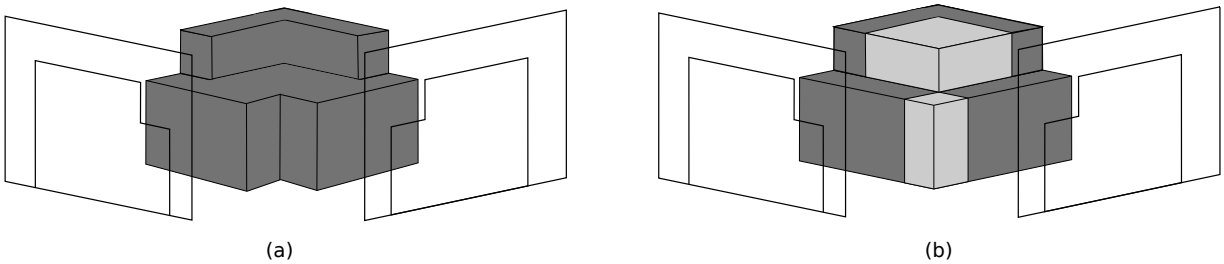


**Fig 2** Shape-from-Silhouette principle[7] (visual hull): (a) Shown is a non-convex object (dark grey) in 3D space and two silhouette images from the object with different viewpoints. (b) The reconstruction resulting from a back-projection of the silhouettes does not provide all details of the original shape. Additional cuboids (light grey) are reconstructed.

## 2 State of the Art

### *2.1 Visual Hull*

Shape-from-Silhouette methods (SfS) are commonly used for the geometric approximation of objects in 3D space. These methods use silhouette images with different views as input and accomplish for each image a back-projection of the silhouette resulting in a 3D visual cone. Each visual cone has an unlimited depth and provides only the knowledge that the object of interest must be located somewhere inside (given a complete silhouette). The geometric approximation of the object is obtained by intersecting the visual cones of several camera views (called volume intersection).

The idea of the SfS approach was already published in 1974.[8] Laurentini introduced the term visual hull.[9] Overall, the SfS approaches of the visual hull are easy and fast to compute, but do have their limitations, as shown in Fig. 2. Not all non-convex objects can be reconstructed with sufficient quality even with a high amount of cameras. Details about the reconstructability of different object geometries can be found in the works of Laurentini.[7,10] For the computation of the visual hull, different representations and data structures are applied. Surface-based methods, e.g. polyhedrons have been investigated[11–13] as well as volume-based methods like voxels[14,15] and conexels.[16] Polyhedrons, e.g. triangle meshes, require a more complex contour extraction step


**Fig 3** Influence of occluders on background subtraction: (a) Reference image of scene with occluders. (b) Image of the scene with occluders and object of interest (human). (c) Output from a background subtraction method: binary silhouette image. As parts of the human are occluded, the silhouette is not complete and will lead to an erroneous reconstruction.

for the images, e.g. with help of search algorithms. In comparison, volume-based methods use segmentation algorithms for the automatic creation of binary silhouette images, where all silhouette pixels can be back-projected into 3D space separately, exhibiting implicit parallelism. For this reason and a variety of other advantages like easy access, storage and management we use voxel data in our reconstruction algorithms.

Typically, background subtraction algorithms (also known as change detection[17]) are applied for the segmentation of (moving) objects in sequences of images (see Figure 3). The principle of such methods is, that for each camera one or several images are taken in order to create a representative reference image or background model from the scene, that does not contain objects of interest. In the monitoring mode of the system all new camera images are compared to that reference. When unknown objects occur in the scene their appearance usually differs from the reference, so that a segmentation of the objects can be accomplished. Problematic for the reconstruction of the visual hull are erroneous silhouette images, which can easily occur with a background subtraction method. Noise and shadows might lead to larger silhouettes and therewith to more coarse visual hulls. On the other hand, clipped silhouettes can arise from similar appearances of objects and

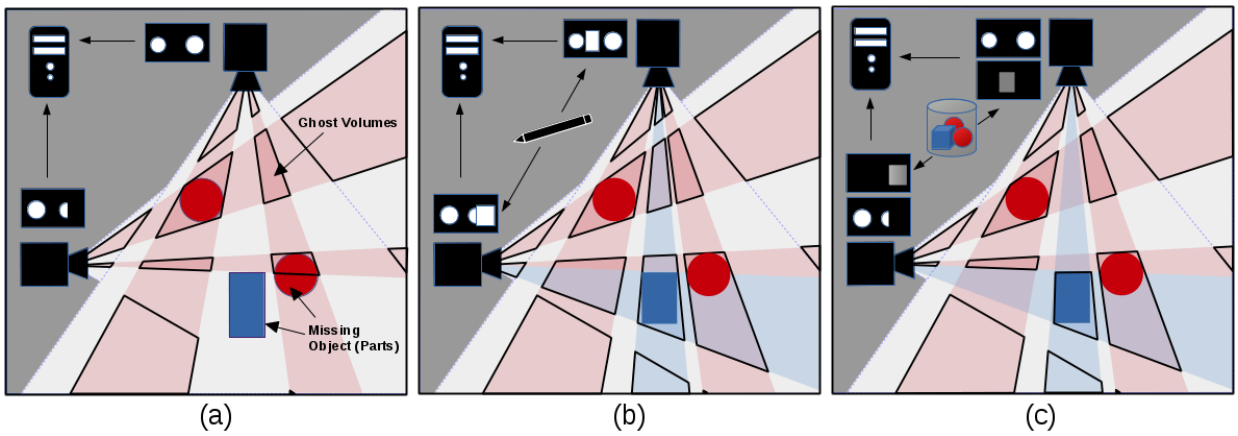


**Fig 4** Concepts of occlusion handling for the visual hull: (a) Incomplete visual hull caused by clipped silhouette images from background subtraction due to occluding obstacles. (b) Occlusion Masks:[15] Silhouette information of known static occluders is added to the output of the background subtraction. All parts of the objects are located within the visual hull, but also more additional empty volumes are reconstructed (called pseudo objects or ghost volumes). (c) Less ghost volumes are reconstructed by using 3D geometry information of the static occluders in order to create depth images that are combined with the silhouette images from the background subtraction.

backgrounds in the images or due to occlusions in the scene. The latter is caused by static obstacles present during reference creation. Thus, objects of interest might be partially or completely occluded for some or all the cameras, which leads to clipped silhouette images as shown in Fig. 3 and for this reason to incomplete visual hulls, that might not fully include the objects of interest (see Fig. 4 (a)).

Different approaches exist to handle the discussed occlusion problem. One method[15] uses occlusion masks (binary masks in the image space of the static occluders), which are added to the output silhouettes from the background subtraction as shown in Fig. 4 (b). During reconstruction the pixels from the silhouette and the occlusion mask are back-projected such that all resulting pixel cones form the visual cone of one camera view. Binary occlusion masks can be created manually or with help of an iterative learning method.[18,19] In previous works we generated occlusion masks from dynamic known obstacles like a robot, by rendering the robots geometry online into the camera images with help of the known robot configuration. This can be used for image-based

collision detection[20] or distance computation[21,22] in surveillance scenes.

Occlusion masks ensure that all objects in the scene are located inside the visual hull (given ideal silhouettes for the visible part of the objects), but they also lead to additional empty volumes that are reconstructed as well, shown in Fig. 4 (b). Such volumes are called pseudo objects or ghost volumes in literature. We published another solution.[4] Geometric models of all obstacles in the scene are used to create for each camera a depth image, which determines the free space up to the surfaces of the obstacles similar to a range sensor. This enables a better geometric approximation than with the occlusion masks, as shown in Fig. 4 (c). More other solutions exist, e.g. Reference (Ref.)[23] However, we apply our approach from Ref.,[4] which incorporates the maximum knowledge about the geometry of known obstacles and offers at the same time a conservative reconstruction of the visual hull.

### *2.2 Photo Hull*

In comparison to the SfS methods (visual hull), the reconstruction of a photo hull is based on an alternative reconstruction principle, namely the color reconstruction.[24] This principle also is known as space carving[14] or voxel carving. The color information of the images controls the reconstruction process and is directly assigned to the voxels in 3D space, which results in a colored voxel representation of the object's geometry. The process of reconstruction starts with a fully occupied voxel space. Visible voxels are carved iteratively until the object's surface is reached. A voxel is visible in a pixel if no other occupied voxel is in front of that voxel from the camera's point of view. The visibility of a voxel might change with every carved voxel in the voxelspace and thus has to be determined again in each iteration (visibility test). The decision for carving a visible voxel is based on the concept of color consistency. A voxel is consistent if the respecting pixels of

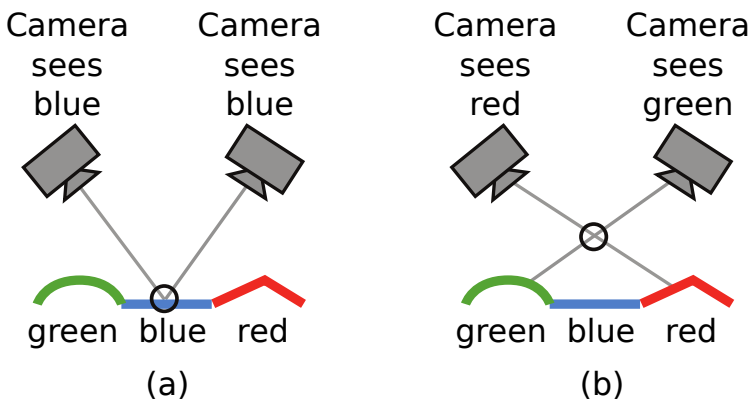


**Fig 5** Principle of the color consistency test shown for two cameras and a voxel (small black circle):[25] (a) The two cameras capture the same color (blue). Thus it is assumed that the voxel must lie on the surface of the reconstructed object. The voxel is not carved. (b) The two cameras capture different colors in the voxel (red and green). It is assumed that the voxel is not part of the object's surface. The voxel is carved (marked transparent).

all cameras for which it is visible have the same color, otherwise it is carved. The color consistency test is shown in Fig. 5. An overview of different consistency criteria, e.g. the maximum norm, can be found in Ref.[26] The reconstruction result is supposed to reproduce the original images when projected into the cameras (criteria of photo integrity). The photo hull might produce a tighter (and thus possibly better) reconstruction of the object's geometry than the visual hull (shown in Fig. 6). A drawback is that the photo integrity can be fulfilled for different object geometries at the same time, which results in ambiguities in the reconstruction, shown in Fig. 6 (a) and (b). More cameras might help in solving this ambiguity problem, but it also strongly depends on the color properties of the object as well as the applied color consistency criterion and the related thresholds. Concerning the lighting condition of the scene, typically a Lambertian reflection is assumed (diffuse lighting), which means that the color of a surface appears similar from different viewpoints (approaches exist that relax this assumption).

The bottleneck of voxel carving is the required visibility test in each iteration, which is time consuming, so that long time voxel carving was deemed inapplicable in real-time scenarios due to frame rates of several minutes.[26] However, modern graphics hardware promises new possibilities

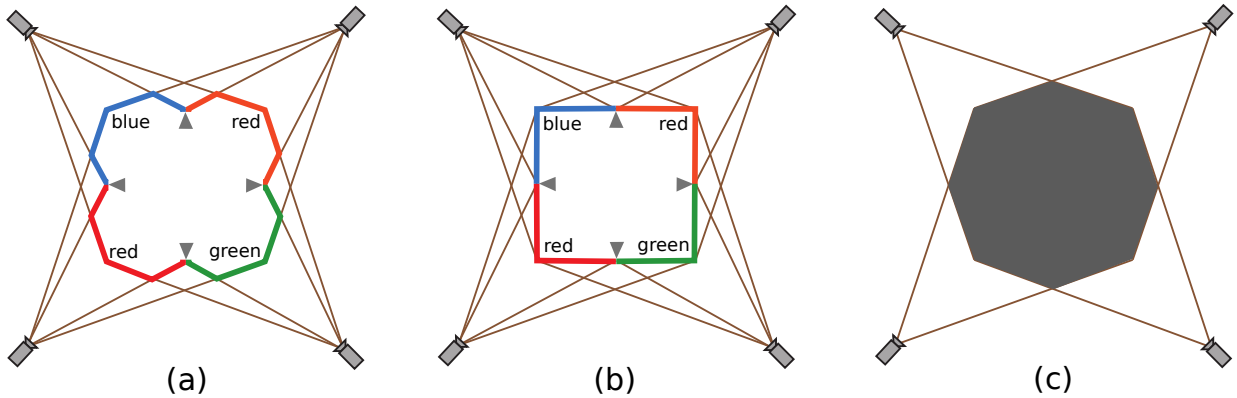


**Fig 6** Reconstruction of a photo hull and ambiguity:[14] The photo integrity can be fulfilled for different objects. Both photo hulls from (a) and (b) generate the same images and lead to an ambiguity. Nevertheless, the photo hull might provide a better geometrical approximation than the visual hull, shown in (c).

for accelerating the photo hull computation as provided by Ref.[27] and focused in this paper. Different concepts of voxel carving distinguish in the determination of the visibility. A naive and simple algorithm with high computation cost is proposed in Ref.[25] This cost can strongly be reduced by introducing a limitation to the camera positions, called ordinal visibility constraint,[24] shown in Fig. 7 (a). All cameras have to be placed behind a parting plane so that the voxels can be processed in a fixed order in one iteration. Disadvantageous is the incomplete reconstruction due to insufficient perspectives. In comparison, algorithms of the partial visibility space carving (PVSC) and full visibility space carving (FVSC)[26] apply a sweep subsequently along all three coordinate axes of the voxelspace in positive and negative directions, beginning from the external borders, as shown in Fig. 7 (b). Only the active cameras of the current sweep plane are incorporated in the consistency test. An exact computation with arbitrary camera placement is provided with the generalized voxel coloring (GVC).[5] The algorithm is shown in Algorithm 3.7. The visibility of the voxels is managed with help of a surface voxel list (SVL), shown in Fig. 8. The GVC-IB approach projects in each iteration all voxels of the SVL into the cameras and saves the closest and thus visible voxel for each pixel in an item buffer (IB). The consistency of each voxel from the SVL is determined by

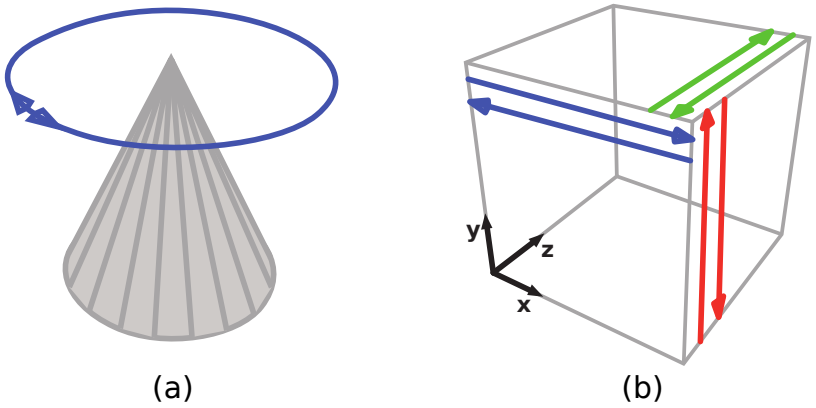


**Fig 7** Voxel visibility: (a) Camera configurations with top down view rotating around the object fulfill the Ordinal-Visibility-Constraint.[24] No explicit visibility test is required and the voxelspace can be processed in one iteration. (b) The visibility of the voxels is determined in the PVSC and FVSC algorithms by sweeping the voxelspace along its axes in six directions in each iteration.

the use of the assigned pixels from the item buffer. Whenever a voxel from the SVL is carved, it is replaced by its uncarved neighbors. This is repeated until all voxels in the SVL are consistent. A disadvantage is that the voxels of the SVL permanently have to be projected into the cameras. An improvement of the computation time can be achieved by using sorted linked lists as in the generalized voxel coloring - layered depth images method (GVC-LDI),[5] but the algorithmic complexity and the memory usage are increasing.

**(Hardware) Acceleration:** In Ref.[28] the use of texture mapping for fast voxel-pixel projections is suggested. Also, a coarse-to-fine approach e.g. with octree structures as well as temporal coherence for image sequences is recommended. Plenty approaches have been developed to accelerate the processing of the visual hull and the photo hull that often follow these fundamental ideas (partially also the presented references). For instance, a current octree-based visual hull is proposed in Ref.[29] In Ref.[30] a texture mapping is applied in combination with an octree and a multiple-sweep-space-carving similar to the PVSC approach for creating a photo hull. Another photo hull approach uses raycasting to reduce the computation time of voxel-pixel projections.[31] 20 to 30 cameras were used and computation times of 700 seconds could be reached with the for-

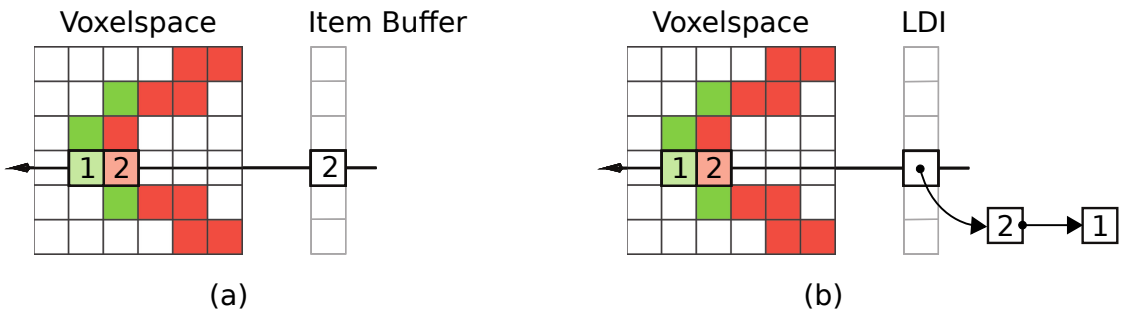


**Fig 8** Voxel visibility:[5] Illustration of the data structures for the surface visibility list applied in the GVC algorithms (a) Item-Buffer and (b) Layered-Depth-Images.

mer hardware. Many other approaches provide solutions that focus on the rendering of new virtual perspectives (e.g. by applying the ordinal visibility constraint) without the execution of an explicit reconstruction (which is our concern). One method that comes close to our work is provided in Ref.[27] A modern graphics board is used to compute segmented images, the visual hull (via vertex shader), and the photo hull (via fragment shader). A multi-sweep approach that employs a raycasting step with early-ray-termination as well as an heuristic approach is applied for determining the visibility of the voxels, whereas only the voxel center is projected to the images. For a voxelspace of size $94 \times 94 \times 113$ and eight FireWire cameras with a resolution of $1024 \times 768$ a frame rate of 33 fps is achieved.

So far, to our knowledge, the existing (accelerated) approaches lack at minimum in one of the following aspects: 1. Explicit consideration of occluding obstacles. This is required when applying a background subtraction for scenes with obstacles. 2. Projection of the complete voxel volumes to the images, instead of using the voxel center[27] or another simplification.[15] This is required to gain a higher quality of the reconstruction. 3. Computation of exact voxel visibilities. This is required for the use of arbitrary camera placements in voxel carving. In our approach (Sec. 3) we combine existing concepts (partially from own former works) and provide GPU algorithms that consider all of these aspects.

## 3 Our Approach

In this Section we describe our GPU algorithms for a visual hull and a photo hull which overcome the mentioned drawbacks of the former approaches. For the explicit consideration of occluding obstacles we adapted our special visual hull algorithm[4] and redesigned it for hardware acceleration on the GPU, described in Sec. 3.1.3. We previously[4] used a look-up table holding the voxel-pixel correspondences of complete voxel volumes similar to Ref.[15] In comparison, we apply a GPU-supported rendering method (e.g. raycasting) in this work in order to be able to process higher resolutions of the voxelspace and the images. For reconstructing a photo hull (GVC-IB voxel carving[5]) on the GPU, we apply a rendering in combination with a transfer function in order to compute exact voxel visibilities. As consistency criterion we use the likelihood ratio test (LRT),[24] because we achieved the best reconstruction quality with it, compared to the ASDT, histogram and maximum norm,[26] in a benchmark test. We make possible the computation of the LRT on the GPU by applying an incremental computation of the standard deviation. This permits (in combination with the rendering) a fast parallelization of the incremental photo hull on the GPU. Furthermore, we integrate the anytime concept[6] to allow the termination of the algorithm after a defined maximum computation time $T$.

For hardware acceleration we use the OpenGL standard. More details on our approach and the results can be found in Ref.[32] A pseudocode of our reconstruction pipeline is shown in Algorithm (Alg.) 3.1. All pseudocodes are self-explanatory, so that only few references to those are given in the text. For better understanding we start with the CPU algorithms before presenting the GPU-based implementations of the visual hull and the photo hull. An experimental evaluation of those as well as a comparison of different rendering techniques is provided in the subsequent Sec. 4.

**Algorithm 3.1** Main Function for an accelerated reconstruction of a photo hull

**procedure** MAINFUNCTION($V$, $V_{\text{occ}}$, $V_{\text{free}}$, $C$, $O$, $\tau$, $T$)
    COPYTOGPU($V$, $C$, $O$, $\tau$, $T$)
    $I_{\text{dep}} \leftarrow$ CREATEDEPTHIMAGESFROMOBSTACLES($C$, $O$) ▷ project obstacles into cameras
    $I_{\text{col}}^{\text{ref}} \leftarrow$ CAPTUREIMAGES($C$) ▷ capture set of reference images on host
    **while** SYSTEMSTOP $= false$ **do** ▷ do for every time frame
        $I_{\text{col}} \leftarrow$ CAPTUREIMAGES($C$) ▷ capture current camera images on host
        $I_{\text{bin}} \leftarrow$ BACKGROUNDSUBTRACTION($I_{\text{col}}$, $I_{\text{col}}^{\text{ref}}$) ▷ compute silhouette images on host
        COPYTOGPU($I_{\text{col}}$, $I_{\text{bin}}$)
        $V_{\text{VH}} \leftarrow$ VISUALHULLCONSERVATIVE($V$, $V_{\text{occ}}$, $V_{\text{free}}$, $C$, $I_{\text{bin}}$, $I_{\text{dep}}$) ▷ visual hull
        $V_{\text{PH}} \leftarrow$ PARALLELGVC-IB($V_{\text{VH}}$, $C$, $I_{\text{dep}}$, $\tau$, $T$,) ▷ refine to photo hull
        COPYTOHOST($V_{\text{PH}}$) ▷ pull back from GPU to host
        use reconstructed photo hull in further processing steps
    **end while**
**end procedure**

### *3.1 Visual Hull*

#### *3.1.1 Standard Visual Hull*

In general, our reconstruction runs at discrete steps $t$ over a time interval $t_{\text{end}} - t_{\text{start}}$. However, our computations are not incremental over time. Thus, we improve clarity of the following discussion by only considering a single step $t$. There is no loss of generality, and an additional time index intuitively integrates into subsequent formulas. Our surveillance volume, a subvolume $[0, 1]^3 \subset \mathbb{R}^3$, is discretized by dividing each dimension into a number of equally spaced intervals. This results in a grid, called voxelspace, where each grid cell (a cube or a cuboid) is a voxel. We define $V = \{v_1, \ldots, v_{|V|}\}$ as a finite set of $|V| \in \mathbb{N}$ voxels $v_i$. Let $\lambda : V \to \mathbb{R}^3$ be a function that returns the center point $r_{v_i} = \lambda(v_i)$ of each voxel. For monitoring the surveillance volume, let $C = \{c_1, \ldots, c_{|C|}\}$ be a finite set of $|C| \in \mathbb{N}$ calibrated and synchronized color cameras $c_j$. The function $\omega : c \to \mathbb{R}^3$ returns the position of a camera's focal point $r_{c_j} = \omega(c_j)$. We assume a pinhole projection and ignore effects of distortion in the following explanations, though we considered such effects in processing the images from our real-world scenario. Moreover, for a

better illustration of our algorithms we omit some implementation details and assume that each voxel of the voxelspace can be seen from all cameras. Each camera contains a number of sensor elements, called pixels, defined as a finite set $P_{c_j} = \{p_1, \ldots, p_{|P_{c_j}|}\}$ of $|P_{c_j}| \in \mathbb{N}$ pixels $p_k \in P_{c_j}$, where the set of all pixels in all cameras is referred to as $P = \cup_{c_j \in C} P_{c_j}$. At each time step a camera $c_j$ provides for each pixel discrete data as a map $image_{\{\text{col,bin,dep}\},c_j} : P_{c_j} \rightarrow D_{\{\text{col,bin,dep}\}}$, where $D_{\text{col}} = [1, 2^m]^3 \in \mathbb{N}^3$ (color image with $m$ bit color resolution in three channels), $D_{\text{bin}} = \{0, 1\}$ (binary image) and $D_{\text{dep}} = \mathbb{R}$ (depth image).

Given the color images $I_{\text{col}} = \{image_{\text{col},c_1}, \ldots, image_{\text{col},c_{|C|}}\}$ of all cameras of the multi-camera system, a background subtraction is applied that returns a number of segmented binary silhouette images $I_{\text{bin}} = \{image_{\text{bin},c_1}, \ldots, image_{\text{bin},c_{|C|}}\}$. The background subtraction method requires a limited number of reference color images, that are captured at a certain time $t^{\text{ref}}$, while no objects of interest are present in the scene. We refer to this step using a set of images $I^{\text{ref}}_{\text{col}}$. We use a background subtraction method from the OpenCV library[33] and accomplish the segmentation on the host. This can be replaced by any other method (which might be more complex and possibly adaptive) that is suitable for the reconstruction conditions of the specific environment, e.g. with changing lighting conditions. After background subtraction, the resulting value of each $p_k \in P_{c_j}$ is given either as $image_{\text{bin},c_j}(p_k) = 0$ if $p_k$ is classified as background or $image_{\text{bin},c_j}(p_k) = 1$ if $p_k$ is classified as foreground. The segmented binary silhouette images are used as input for reconstructing a visual hull. The visual hull (VH) is a set of voxels with $V_{\text{VH}} \subset V$ that is reconstructed from a priori unknown humans (objects) in the surveillance volume. Let $\Phi_{v_i,c_j} \subset P_{c_j}$ be the set of projection pixels of a voxel $v_i$ in camera $c_j$, and $\Phi_{v_i} = \cup_{c_j \in C}(\Phi_{v_i,c_j})$. The visual hull

is composed of all those voxels $v_i \in V$ for which the Formula (Form.) (1) holds.

$$\forall c_j \in C : \forall p_k \in \Phi_{v_i,c_j} : image_{\text{bin},c_j}(p_k) = 1 \tag{1}$$

Given the pixel sets $\Phi_{v_i,c_j}$ for all voxels, the visual hull can be determined by set operators in terms of boolean expressions. A simple algorithm for computing the standard visual hull is shown in Alg. 3.2. A voxel that is carved is discarded. It is empty (free) and not visible anymore.

**Algorithm 3.2** Algorithm for a standard visual hull

```
1:  procedure StandardVisualHull(V, C, I_bin)
2:      V_VH ← V                                   ▷ initialize visual hull with full voxelspace
3:      for all v_i ∈ V_VH do                                              ▷ for each voxel
4:          for all c_j ∈ C do                                             ▷ for each camera
5:              Φ_{v_i,c_j} ← ProjectVoxelToCam(v_i)
6:              for all p_k ∈ Φ_{v_i,c_j} do                  ▷ for each projection pixel of voxel
7:                  if GetImageValue(p_k, I_bin, c_j) = 0 then                ▷ if pixel is background
8:                      V_VH ← V_VH \ {v}                                     ▷ carve voxel
9:                  end if
10:             end for
11:         end for
12:     end for
13:     return V_VH                                          ▷ return set of uncarved voxels
14: end procedure
```

### *3.1.2 Visual Hull with Occluding Obstacles*

For the standard visual hull, projections of objects are assumed to be completely included in the silhouettes of the $|C|$ segmented images $I_{\text{bin},c_j}$. This is true (given ideal silhouette images) if the capturing of reference images was accomplished for an empty surveillance volume. As opposed to this, static obstacles like tables that are present in advance are not part of the silhouettes and thus are not reconstructed. Moreover they might occlude partially or completely the objects so that clipped silhouettes arise. In result Form. (1) is false negative and it is not guaranteed that the remaining

visual hull contains the complete objects. We provide a solution of this problem in.[4] Knowledge of a priori known static and dynamic occluding obstacles is integrated into the reconstruction process. In this paper we focus on known static obstacles only. Given $O = \{o_1, \ldots, o_{|O|}\}$ as a finite set of $|O| \in \mathbb{N}$ a priori known obstacles $o_q$ in the surveillance volume, we assume that all those obstacles are geometrically modeled (in our work as triangle meshes). With help of those models, we create depth images $I_{\text{dep}} = \{image_{\text{dep},c_1}, \ldots, image_{\text{dep},c_{|C|}}\}$ in an offline step on the GPU for all cameras (with the same pixel resolution as the other images). A depth image encodes the maximum free range of each pixel $p_k$ in $P_{c_j}$. If the pixel $p_k$ sees no obstacle, the assigned depth pixel value is infinite. In the reconstruction process all pixels of a projected voxel are tested for visibility given their depth values. Only pixels having a free sight to the voxel are permitted to contribute to the decision of carving (occupation).

Given the position $r_{c_j} \in \mathbb{R}^3$ of the camera $c_j$, the center of a voxel $r_{v_i} \in \mathbb{R}^3$ and a constant $d_v \in \mathbb{R}$ describing half of the length of the diagonal of a voxel (to be conservative), the Form. (1) can be extended such that, for all voxels of the visual hull $v_i \in V_{\text{VH}}$ holds:

$$\forall p_k \in \Phi_{v_i,c_j} : (image_{\text{bin},c_j}(p_k) = 1) \vee (|r_{v_i} - r_{c_j}| + d_v \geq image_{\text{dep},c_j}(p_k)) \tag{2}$$

Accordingly, all objects and all obstacles are part of the visual hull. The magnitude of the difference of the position vectors is calculated via $L_2$-Norm and thus corresponds to the distance between both positions. The algorithm in Fig. 3.2 needs to be adjusted so that Line 7 implements:

$$(|r_{v_i} - r_{c_j}| + d_v < image_{\text{dep},c_j}(p_k)) \wedge (image_{\text{bin},c_j}(p_k) = 0) \tag{3}$$

### 3.1.3 Conservative Visual Hull with Occluding Obstacles

The Formulas (1) and (2) do not account for discretization errors of the voxelspace approximation, which have an impact on the boundary of the visual hull. Only one projection pixel $p_k \in \Phi_{v_i,c_j}$ of a camera with value $image_{\text{bin},c_j}(p_k) = 0$ (background) suffices to carve a voxel, though all other projection pixels might have the value $image_{\text{bin},c_j}(p_k) = 1$ (foreground). In result, the boundary of the visual hull is formed by voxels that completely project to silhouette pixels in all cameras. However, to ensure that all parts of objects and obstacles are enclosed by the visual hull, the following conservative formulation is required. A voxel is only carved if all projection pixels $\Phi_{v_i,c_j}$ of at least one camera are classified as background, whereas no occluded or foreground pixels are allowed (Form. (4)).

$$\exists c_j \in C : \forall p_k \in \Phi_{v_i,c_j} : (image_{\text{bin},c_j}(p_k) = 0) \wedge (|r_{v_i} - r_{c_j}| + d_v < image_{\text{dep},c_j}(p_k)) \quad (4)$$

On the other hand, one foreground or one occluded pixel in each camera is sufficient to keep the voxel in the reconstruction (Form. (5)).

$$\forall c_j \in C : \exists p_k \in \Phi_{v_i,c_j} : (image_{\text{bin},c_j}(p_k) = 1) \vee (|r_{v_i} - r_{c_j}| + d_v \geq image_{\text{dep},c_j}(p_k)) \quad (5)$$

A respective algorithm is shown in Alg. 3.3.

### 3.1.4 Conservative Visual Hull with Occluding Obstacles on the GPU

After having shown the conservative visual hull with occluding obstacles we present our corresponding implementation on the GPU, shown in Algorithms 3.4 and 3.5.

**Algorithm 3.3** Algorithm for a conservative visual hull with occlusion handling

**procedure** CONSERVATIVEVISUALHULL($V$, $C$, $I_{\text{bin}}$, $I_{\text{dep}}$)
  $V_{\text{VH}} \leftarrow V$ ▷ initialize visual hull with all voxels from voxelspace
  **for all** $v_i \in V_{\text{VH}}$ **do** ▷ for each voxel
    **for all** $c_j \in C$ **do** ▷ for each camera
      *flag_foreground* $\leftarrow$ *nil* ▷ a flag with three values is required
      *flag_occludedPixelExist* $\leftarrow$ *false*
      $\Phi_{v_i,c_j} \leftarrow$ PROJECTVOXELTOCAM($v_i$)
      **for all** $p_k \in \Phi_{v_i,c_j}$ **do** ▷ for each projection pixel of the voxel
        **if** $|r_{v_i} - r_{c_j}| + d_v <$ GETIMAGEVALUE($p_k$, $I_{\text{dep}}$, $c_j$) **then** ▷ if voxel is visible
          **if** GETIMAGEVALUE($p_k$, $I_{\text{bin}}$, $c_j$) $= 1$ **then** ▷ if pixel is foreground
            *flag_foreground* $\leftarrow$ *true*
          **end if**
          **if** *flag_foreground* $=$ *nil* **then** ▷ if first projection pixel in $c_j$ is background
            *flag_foreground* $\leftarrow$ *false*
          **end if**
        **else** ▷ voxel is not visible in pixel
          *flag_occludedPixelExist* $\leftarrow$ *true*
        **end if**
      **end for**
      **if** (*flag_foreground* $=$ *false*) $\wedge$ (*flag_occludedPixelExist* $=$ *false*) **then**
        $V_{\text{VH}} \leftarrow V_{\text{VH}} \setminus \{v_i\}$ ▷ carve voxel
      **end if**
    **end for**
  **end for**
  **return** $V_{\text{VH}}$
**end procedure**

All segmented images $I_{\text{bin}}$ and other required data are transferred to the GPU. Each camera is assigned an ID, given with $id : C \rightarrow \{1, \ldots, |C|\}$ where $id(c_j) = j$. The computation takes place sequentially for one camera after the other, sorted by ascending $id(c_j)$, but parallel for all pixels. The process executed for each pixel and each voxel is shown in Alg. 3.5. Each pixel handles all voxels that are included by its backprojection cone in a sequential way. This is accomplished by an iterative rendering.

**Algorithm 3.4** Algorithm for a conservative visual hull with occlusion handling on the GPU

1: **procedure** VISUALHULLCONSERVATIVE($V$, $V_{\text{occ}}$, $V_{\text{free}}$, $C$, $I_{\text{bin}}$, $I_{\text{dep}}$)
2: **for** from $l \leftarrow$ ID(c)$= 1$ to ID(c)$= |C|$ step 1 **do** ▷ sequentially with ascending camera id
3: **parallel for all** $p_k \in P_{c_j}$ **do** ▷ parallel for all pixels
4: $v_i \leftarrow$ GETFIRSTVOXELPROJECTINGTOPIXEL($p_k$, $V$) ▷ render closest voxel
5: **while** $v_i \in V$ **do**
6: $v_{\text{occ}} \leftarrow$ GETVOXELOCC($v_i$), $v_{\text{free}} \leftarrow$ GETVOXELFREE($v_i$)
7: PROCESSVOXELINPIXEL($v_{\text{occ}}, v_{\text{free}}, p_k, c_j, I_{\text{bin}}, I_{\text{dep}}$)
8: $v_i \leftarrow$ GETNEXTVOXELPROJECTINGTOPIXEL($v_i$, $p_k$, $V$)
9: **end while**
10: **end parallel for**
11: **end for**
12: $V_{\text{VH}} \leftarrow$ ANALYZEVOXELVALUES($V_{\text{occ}}$,$V_{\text{free}}$) ▷ gather all occupied voxels
13: **return** $V_{\text{VH}}$
14: **end procedure**

**Algorithm 3.5** Algorithm for processing each pixel in parallel

1: **procedure** PROCESSVOXELINPIXEL($v_{\text{occ}}, v_{\text{free}}, p_k, c_j, I_{\text{bin}}, I_{\text{dep}}$)
2: **if** VALUE($v_{\text{occ}}$) $\geq$ VALUE($v_{\text{free}}$) $\vee$ VALUE($v_{\text{free}}$) $=$ ID($c_j$) **then**
3: **if** (GETIMAGEVALUE($p_k$, $I_{\text{bin}}, c_j$) $= 1 \vee$ ▷ if pixel is foreground
4: $|r_{v_i} - r_{c_j}| + d_v \geq$ GETIMAGEVALUE($p_k$, $I_{\text{dep}}, c_j$) ) **then** ▷ if voxel is occluded
5: VALUE($v_{\text{occ}}$) $=$ ID($c_j$)
6: **else** ▷ if voxel is visible and pixel is background
7: VALUE($v_{\text{free}}$) $=$ ID($c_j$)
8: **end if**
9: **end if**
10: **end procedure**

We create two equal sized voxelspaces $V_{\text{free}}$ and $V_{\text{occ}}$ that have the properties of the original voxelspace. Each voxel is initialized with 0 and holds a single camera ID after processing. This ID is obtained by the functions $value_z : V \rightarrow \{0\} \cup \{1, \ldots, |C|\}$ over the iteration steps $z$, where value $value_0(v_i) = 0$ and value of iteration $z + 1$ derives as shown in pseudocode Alg. 3.5. Basically, every voxel gets for each projection pixel an entry of at least one camera ID in either $V_{\text{free}}$ or $V_{\text{occ}}$. The pixel-parallelism is ensured by the special construction of the condition in Alg. 3.5, Line 2. Possible mixes of read and write operations on the voxel space do not matter, because after the processing of a camera (the outer loop), a memory barrier is employed, that ensures

| $value(v_{occ}) < value(v_{free})$ |
|---|
| Camera $id(c_j) = \|C\|$ or a previous processed camera contains only free pixels. One camera that sees the voxel completely as free is sufficient to carve the voxel. Thus the voxel is **free**. |
| $value(v_{occ}) \geq value(v_{free})$ |
| At least one occupied or occluded pixel was recorded for the last camera $id(c_j) = \|C\|$. This indicates that no previous camera sees the voxel completely as free. The voxel is **occupied**. |

**Table 1** Final values of each voxel in $V_{\text{free}}$ and $V_{\text{occ}}$ and the resulting voxel occupation.

synchronization. All pixels that are part of an object's silhouette or that are not able to see the voxel due to occlusions generate an entry in $V_{\text{occ}}$ (Alg. 3.5, Line 5). All the other projection pixels generate an entry in $V_{\text{free}}$. To carve a voxel (mark as free), it sufficies that the voxel is complety background and not occluded in all projection pixels of one camera. The condition of Line 2 in Alg. 3.5 is constructed so that it is never true for the following cameras, after being fulfilled once for a camera. The final visual hull corresponds to all voxels classified as occupied. The classification of each voxel can be captured by comparing its entrys of $V_{\text{occ}}$ and $V_{\text{free}}$ in a postprocessing step, as shown in Table 1 (see also Alg. 3.5, Line 12).

Hitherto, objects and occluding obstacles are part of the visual hull. However, currently we consider scenarios with static obstacles, whereby parts of the visual hull remain constant for every time step $t$ of the image sequence. For the subsequent computation of the photo hull we decided to ignore the constant parts (which might be computed once in an offline step) and concentrate on the reconstruction of the changing environment (humans). The static parts can be added to the online reconstruction result in a postprocessing step. For this reason a modified algorithm is shown in Alg. 3.6. The difference is that occluded pixels are now ignored completely (Lines 2 and 3). The resulting values of the two voxelspaces $V_{\text{free}}$ or $V_{\text{occ}}$ have to be interpreted as shown in Table 2.

**Algorithm 3.6** Algorithm for a conservative visual hull with occlusion handling on the GPU. Static Obstacles are not reconstructed.

```
1:  procedure PROCESSVOXELINPIXEL(v_i, p_k, c_j, I_bin, I_dep)
2:      if |r_{v_i} − r_{c_j}| + d_v ≥ GETIMAGEVALUE(p_k, I_dep, c_j) then
3:          return
4:      end if
5:      if VALUE(v_occ) ≥ VALUE(v_free) ∨ VALUE(v_free) = ID(c_j) then
6:          if GETIMAGEVALUE(p_k, I_bin) = 1 then
7:              VALUE(v_occ) = ID(c_j)
8:          else
9:              VALUE(v_free) = ID(c_j)
10:         end if
11:     end if
12: end procedure
```

| |
|---|
| $value(v_{occ}) < value(v_{free})$ |
| Camera $id(c_j) = \|C\|$ or a previous camera contains only free pixels besides possibly occluded pixels. At least one camera sees the voxel completely as free. This is sufficient to carve the voxel. Thus the voxel is **free**. |
| $(value(v_{occ}) \geq value(v_{free})) \wedge (value(v_{occ}) > 0)$ |
| At least one occupied pixel was recorded for the last camera $id(c_j) = \|C\|$. This indicates that no previous camera sees the voxel as free. The voxel is **occupied**. |
| $(value(v_{occ}) = 0) \wedge (value(v_{free}) = 0)$ |
| The voxel is occluded in all corresponding projection pixels of all cameras. The voxel is **free**. |

**Table 2** Final values of each voxel in $V_{\text{free}}$ and $V_{\text{occ}}$ and the resulting voxel occupation. Static obstacles are not reconstructed.

### *3.2 Photo Hull*

Our GPU algorithm for reconstructing a photo hull can be found in Alg. 3.8. It is derived from the generalized voxel coloring with item buffer (GVC-IB approach) of Ref.,[5] which will be presented before discussing the details of our adaptation.

The algorithm of the GVC-IB approach is shown in Alg. 3.7 and works as follows. At the beginning of each iteration the surface voxels are determined and stored in the dynamic surface visibility list (SVL) as shown in Fig. 8 (a), Sec.2.2. In the first iteration the SVL contains all outer voxels of the voxelspace or the outer voxels of the visual hull if used as input data.

**Algorithm 3.7** Algorithm of the GVC-IB[5] with likelihood ratio test (LRT) as consistency criterion

**procedure** GVC-IB($V$, $I_{\text{col}}$, $\tau$)
  $V_{\text{PH}} \leftarrow V$
  $L \leftarrow$ DETERMINESVL($V_{\text{PH}}$) ▷ determine initial surface visibility list (SVL)
  **repeat**
    **for all** $v_i \in L$ **do** ▷ for all voxels of the SVL
      $\Psi_{v_i} \leftarrow$ VISIBILITY($v_i$) ▷ gather projection pixels that view a voxel
      $\mu_{v_i} \leftarrow$ COMPUTEMEANCOLOR($\Psi_{v_i}$, $I_{\text{col}}$)
      $val \leftarrow$ COMPUTELRT($\mu_{v_i}$, $\Psi_{v_i}$) ▷ compute value for color consistency test
      **if** $val < \tau$ **then** ▷ color is consistent
        COLOR($v_i$) $\leftarrow \mu_{v_i}$ ▷ maintain voxel and add pixel color to voxel
      **else** ▷ color is not consistent
        $V_{\text{PH}} \leftarrow V_{\text{PH}} \setminus \{v_i\}$ ▷ carve voxel
        $L \leftarrow$ UPDATESVL($L$) ▷ remove voxel from SVL and add its neighbours
      **end if**
    **end for**
  **until** no voxel is carved
  **return** $V_{\text{PH}}$
**end procedure**

For each surface voxel from the list, the color consistency test is conducted to decide whether the voxel is located on an object's surface. This is assumed to be true if all pixels in which the voxel is visible have the same color. The voxel remains uncarved in this case (classified as occupied), otherwise the voxel is carved and replaced in the SVL by the next potential surface voxels. The pixels for that a voxel is visible are determined via a visibility test (in this work an adaptation of a SVL and image item-buffers, as shown in Fig. 8). As implementation of a visibility test we define the function $visibility : V \rightarrow 2^P$ where $visibility(v_i)$ of a voxel $v_i$ is the set $\Psi_{v_i} \subset \Phi_{v_i}$ of all pixels that have a free sight to that voxel and thus capture the voxel's color.

The SVL of the GVC-IB is applied in order to reduce the amount of voxel projections. However, an implementation of such a dynamic list on the GPU is hard to realize and allows elements only to be inserted or deleted in a sequential way. Thus, we replace the SVL and utilize an efficient GPU rendering technique, discussed in Sec. 4.1. Therewith in each iteration all the currently

occupied voxels $V_{\text{PH}}$ of the voxelspace $V$ are projected into the cameras, while applying a visibility transfer function to each voxel (Form. (6)), similar to the volume rendering used in other application fields.

Let $idx : V \rightarrow \mathbb{N}^3$ be a function that returns a tupel of indices $(idx(v_i)^{(1)}, idx(v_i)^{(2)}, idx(v_i)^{(3)})$ for the grid position of a voxel $v_i$ and let $image_{\text{vis},c_j} : P_{c_j} \rightarrow \mathbb{N}^3 \times \{0,1\}$ be a map of a pixel to virtual pixel data for each camera $c_j$. Then, we have the visibility images (VI) of all cameras of the multi-camera system as $I_{\text{vis}} = \{image_{\text{vis},c_1}, \ldots, image_{\text{vis},c_{|C|}}\}$. The value of each pixel in a visibility image is given by a function $transfer_{p_k} : V \rightarrow \mathbb{N}^3 \times \{0,1\}$ that maps a voxel's indices to the color channels of the image and adds an occupation value of $\{0,1\}$, indicating that a voxel is occupied $(= 1)$ or free $(= 0)$ (we store this value in the alpha channel of the image). The resulting value of each $p_k$ is given with the transfer function as following:

$$transfer_{p_k}(v_i) = \begin{cases} (idx(v_i)^{(1)}, idx(v_i)^{(2)}, idx(v_i)^{(3)}, 1) & \text{if} \quad v_i \in V_{\text{PH}} \quad \wedge \quad p_k \in \Phi_{v_i,c_j} \quad \wedge \\ & |r_{v_i} - r_{c_j}| + d_v < image_{\text{dep},c_j}(p_k) \\ (0,0,0,0) & \text{else} \end{cases} \tag{6}$$

As our visual hull contains objects but no obstacles, additionally we have to consider the occlusions in Form. (6) by applying the information of the depth images similar to the usage in the visual hull algorithm. The rendering in each pixel terminates after drawing the first visible occupied voxel. This is realized as shown for a raycasting in Alg. 3.9. After rendering in each iteration, the resulting color values of each pixel $p_k$ encode the coordinates of the closest visible voxel (Alg. 3.8, Line 8). This equals the projection of the SVL into the cameras. In our case the initial occupation of each voxel is determined by the computed visual hull from the previous step (Alg. 3.8, Line 4).

**Algorithm 3.8** Algorithm for the accelerated GVC-IB on the GPU

1: **procedure** PARALLELGVC-IB($V_{\text{VH}}$, $C$, $I_{\text{col}}$, $I_{\text{dep}}$, $\tau$, $T$)
2: $V_{\text{PH}} \leftarrow V_{\text{VH}}$
3: **for all** $v_i \in V_{\text{PH}}$ **do**
4: OCCUPATION($v_i$, *true*) ▷ initialize all input voxels as occupied
5: **end for**
6: $s \leftarrow$ TIME ▷ get current time
7: **while** TIME $-s < T$ **do** ▷ reconstruct for defined duration (anytime concept)
8: $I_{\text{vis}} \leftarrow$ RENDERVISIBILITYIMAGES($V_{\text{PH}}$, $I_{\text{dep}}$) ▷ render closest occupied voxels
9: **for all** $c_j \in C$ **do** ▷ for all cameras
10: **parallel for all** $p_k \in P_{c_j}$ **do** ▷ for all pixels in visibility image
11: $v_i \leftarrow$ GETIMAGEVALUE($p_k$, $I_{\text{vis}}$, $c_j$) ▷ get voxel that is visible in pixel
12: $\mu^n_{v_i} \leftarrow$ UPDATEMEANCOLOR($v_i$, $I_{\text{col}}$, $c_j$, $p_k$) ▷ update color mean
13: $\sigma^n_{v_i} \leftarrow$ UPDATESTANDARDDEVIATION($v_i$, $\mu^n_{v_i}$) ▷ update standard deviation
14: **if** $((n-1)/n) \cdot (n \cdot (\sigma_{v_i})^2) < \tau$ **then** ▷ if color is consistent
15: SETCOLOR($v_i$, $\mu^n_{v_i}$) ▷ set new voxel color
16: **else** ▷ color is not consistent
17: OCCUPATION($v_i$, *false*) ▷ carve voxel by setting its state to not occupied
18: **end if**
19: **end parallel for**
20: **end for**
21: **end while**
22: $V_{\text{PH}} \leftarrow$ ANALYZEVOXELVALUES($V_{\text{PH}}$) ▷ gather all occupied voxels
23: **end procedure**

Each carved voxel (Alg. 3.8, Line 17) gets an entry of being not occupied (free), so that it will not be rendered in the next iteration anymore. The result of the rendering in each iteration is stored in so called visibility images (VI). By having encoded the current visible voxel in each pixel, a parallel processing of the pixels is reasonable for the consistency test in each carving iteration. The likelihood ratio test (LRT) is applied as consistency criterion, because we achieved the best qualitative results in a benchmark test. The LRT employs the standard deviation $\sigma_{v_i} \in \mathbb{R}^3$. Uncarved (occupied) voxels are colored with the average value $\mu_{v_i} \in \mathbb{R}^3$ that are assigned to each voxel, given the projection pixels $\Psi_{v_i}$ of all cameras for that the voxel is visible. The color of each pixel $p_k \in \Psi_{v_i}$ is defined as $u_k = image_{\text{col},c_j}(p_k)$. The mean value of the pixel colors $\mu_{v_i}$ that

belong to a voxel and the associated standard deviation $\sigma_{v_i}$ is calculated incrementally to avoid dynamic data structures, too.

**Algorithm 3.9** Algorithm for rendering the visibility images with a raycasting

```
1:  procedure RENDERVISIBILITYIMAGES(V, V_VH, C, I_dep, I_vis)
2:      for all c_j ∈ C do
3:          parallel for all p_k ∈ P_{c_j} do
4:              color ← 0
5:              V_ray ← GETVOXELSALONGRAY(p_k, V)
6:              while color = 0 do                ▷ stop after rendering the first occupied voxel
7:                  v_i ← GETNEXTVOXEL(V_ray)
8:                  color ← GETVALUEFROMTRANSFERFUNCTION(v_i, p_k, c_j, V_PH, I_dep)
9:              end while
10:             SETIMAGEVALUE(p_k, I_vis, c_j, color)   ▷ pixel contains next possible surface voxel
11:         end parallel for
12:     end for
13: end procedure
```

The original definition of the mean value requires all $n = |\Psi_{v_i}|$ elements $u_k$ in advance.

$$\mu_{v_i} = \frac{1}{n} \sum_{k=1}^{n} u_k \tag{7}$$

This can be replaced by the following incremental Equation:

$$\mu_{v_i}^n = \frac{1}{n}(u_n - \mu_{v_i}^{n-1}) + \mu_{v_i}^{n-1} \tag{8}$$

The original definition of the standard deviation is:

$$\sigma_{v_i}^n = \sqrt{\frac{1}{n} \sum_{k=1}^{n} (u_k - \mu_{v_i})^2} \tag{9}$$

Its incremental equivalent can be expressed as in:[34]

$$n \cdot (\sigma_{v_i}^n)^2 = (n-1) \cdot (\sigma_{v_i}^{n-1})^2 + n \cdot (n-1) \cdot (\mu_{v_i}^n - \mu_{v_i}^{n-1})^2 \tag{10}$$

We define the functions $f : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$. As can be seen from the Formula, the computation of $n \cdot (\sigma_{v_i}^n)^2$ requires the evaluation of a monotonic expression following the form of $f(n) = f(n-1) + X$, $X > 0$. Similarly, the function $g(n) = 1 - 1/n = (n-1)/n$ is monotonic when $n > 0$. The product of these two functions $g(n) \cdot f(n)$ will be monotonic as well. A combination of Form. (10) with the definition for the consistency test $lrt : \mathbb{R} \to \mathbb{R}$, which is given as $lrt = (n-1) \cdot (\sigma_{v_i})^2$, yields a function following the form of $g(n) \cdot f(n)$. As such, $lrt$ can be computed incrementally using the following term, which is monotonously increasing in $n$:

$$lrt(n) = \left(\frac{n-1}{n}\right) \cdot \left(n \cdot (\sigma_{v_i}^n)^2\right) \tag{11}$$

Due to this adaptation, the consistency test can now be accomplished parallel for all pixels in each iteration. Whenever $lrt(n) > \tau$, the voxel can be carved immediately as it can not become smaller anymore due to its monotony. Solely the update of $\mu_{v_i}$ and $n \cdot (\sigma_{v_i})^2$ requires a common access. As a repeated carving does not influence the result, a synchronization only is required for accessing the values of the same voxel, which is realized by the use of spin-locks. Due to the high amount of pixels a parallel processing is guaranteed anyhow. A further challenge is the loop condition "until no voxel is carved" of the GVC-IB (Fig. 3.7, Line 16) because this requires a feedback of the graphics board that currently only can be realized with a trick.[35] Moreover the computation time of the photo hull varys depending on the content of the scene. Thus we require a bound for the

maximal computation time. Therefore we apply the concept of an anytime algorithm[6] to the photo hull. After initialization, such algorithms can be interrupted at any time while providing a correct result. The quality of the result improves as function of the time. We define an upper bound of processing time in which at minimum one iteration can be accomplished, so that a colored visual hull is guaranteed in worst case and a refined photo hull is provided in the best case. It should be noted that the obtainable quality of the photo hull is limited mainly due to the properties of the reconstructed objects as well as the experimental setup (e.g. number and perspectives of cameras).

## 4 Results

All experiments were accomplished with a graphics board of type AMD Radeon HD 7970 with 3GB RAM. The proprietary AMD driver "fglrx" (Catalyst) in version 12.104 was employed and the operating system was OpenSUSE 12.3. The processor was of type AMD FX-8350 Octocore, with 4 GHz clock and 32GB RAM. Due to the intensive utilization of the graphics board the number of cores is irrelevant. We chose a fix camera resolution of $640 \times 480$ pixels for all experiments.

### *4.1 Analysis of the Rendering Methods*

As we have already stated, a fast and efficient computation of the complete voxel-pixel projections that are required in the presented algorithms can be realized by employing a GPU-based rendering method. We render the voxelspace in the camera views by considering the camera properties (focal point, distortion etc.) so that each pixel encodes in its RGB values the coordinates of the visible voxel. Two groups of approaches for efficient rendering methods are available, namely texture mapping and raycasting. Texture mapping[36,37] utilizes the ability of graphic boards for surface texturing. It interprets volumes to be rendered, e.g. the voxelspace, as a pack of images (layers),

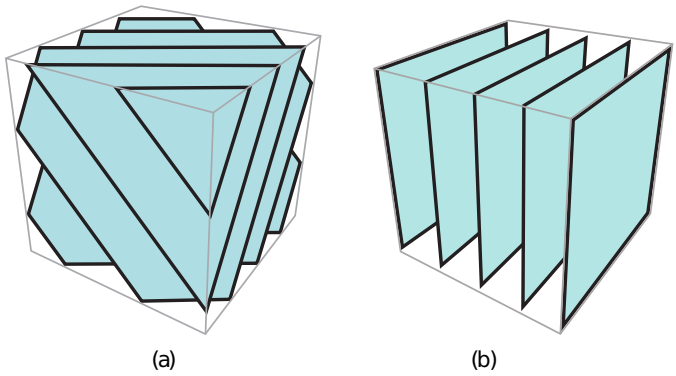


**Fig 9** Texture mapping: orientation of the layers with respect to the rendering view.[36] (a) The rendering view is aligned parallel to the layers. (b) The rendering view is rather orthogonal to the layers and produces more artefacts.

that are rendered individually. For a fast computation the complete layers are rendered instead of single voxels. The layers are projected as texture to locally shifted polygons. When activating blending or an explicit discarding (not required in this work), layers that are farther away become visible in pixels that are empty or semitransparent. For the viewer a spatial image impression is originated. The therefor used layers are called proxy geometry. The work mainly comprises mapping of textures and blending of layers. The quality strongly depends on the distance between the layers as well as the orientation of the layers with respect to the rendering view, as shown in Fig. 9. Larger distances lead to aliasing effects and wrong voxel-pixel mappings. To avoid aliasing effects we insert layers parallel to each coordinate axis as shown in Fig. 10.

In comparison, the standard raycasting determines the visible voxel in each pixel by following the back-projected ”viewing ray” until an occupied voxel is intersected (ray marching). The ray is sampled at discrete points with an equal step size. Different optimizations exist for accelerations, e.g. empty-space-skipping.[38] Aliasing effects caused by the discrete ray marching are avoided by the optimization of Amanatides.[39] The intersections of the ray with the voxels are analytical computed. The transitions of the ray from one voxel to the other is provided without a loss of processing speed. We compare the texture mapping with the standard raycasting, the Amanatides

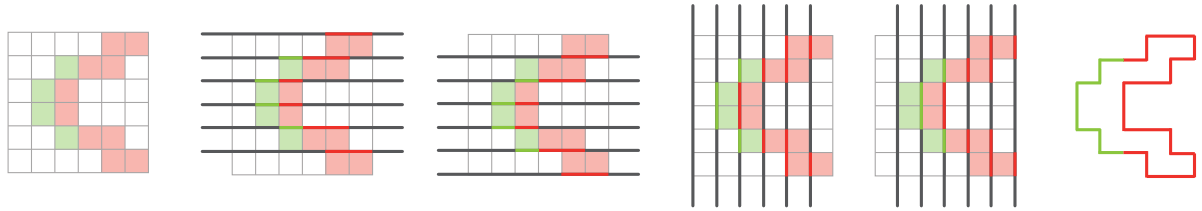

**Fig 10** Exact texture mapping for a voxelspace. From left to right: occupied voxels, rendering in directions: +y, -y, -x, +x, final image of the voxels

raycasting as well as an OpenCL compute shader implementation of the Amanatides algorithm.

For analyzing the computation time of these rendering approaches, we created a test case (independent of the reconstruction methods) in which different voxelspaces were projected to the camera images. The voxelspace resolution was varied from $100^3$ to $350^3$ in steps of $50^3$. We varied the occupation of the voxelspaces from 1 $\%$ to $100\%$ by randomly placing spheres and counting the filled voxels until the desired occupation was achieved.

Fig. 11 shows one result that is characteristic for all results. Beginning with unsegmented images (continuous lines): The texture mapping is worst in all cases. Although it employs most the given hardware functions for vertices, there is much overhead while projecting layers that are not visible into the cameras. The standard raycasting approach with fix step size as well as the optimization of Amanatides[39] produce similar results. The cost for calculating the intersection points of the viewing rays with the voxels (for reducing the number of steps) are opposed to the cost of more required iterations in the standard raycasting.

Obviously, the frame rates increase with higher voxel occupations for all methods. This is caused by in average shorter ray marching distances up to the intersection with an occupied voxel. The frame rates significantly improved when using the compute shader implementation of the Amanatides method. This means that the usage of the full OpenGL pipeline in the vertex- and fragment-shader based raycasting methods is not reasonable due to the arising overhead.

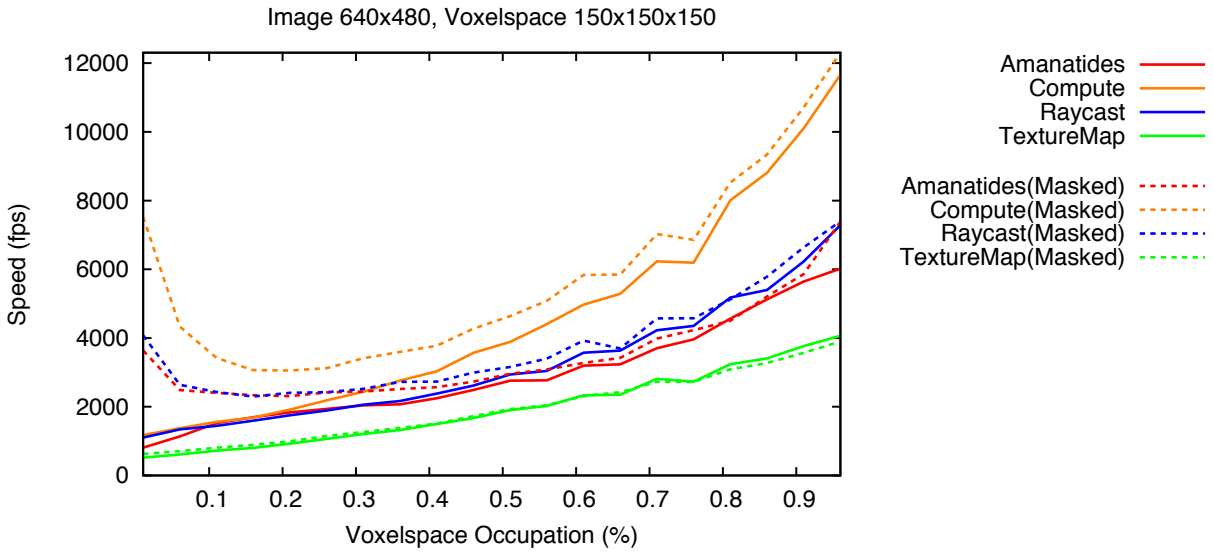


**Fig 11** Computation times of different rendering methods for a varying occupation of the voxelspace.

For each method and parameter combination the influence of segmented silhouette images is examined (dotted lines) as such are used for the reconstruction of the visual hull. The silhouette creation time is not included in the measurements. As expected, all raycasting methods benefit from the resulting skipping of pixels that are located outside the silhouette. This leads additionally to a decreasing computation time for little voxelspace occupations with almost no pixels to process. The efficiency of the texture mapping cannot be increased, because all layers have to be rendered regardless.

We evaluated the quality of the rendering methods by comparing the results with a naive rendering that renders for each single voxel 12 triangles in the image. Although this method is less efficient than the others, it can be handled as ground truth, since it does not apply numerical approximations and returns results of best quality, which we also verified on the CPU for voxelspaces of low resolution. We measured the absolute number of wrong pixels and the average deviation for the actual voxel positions encoded in the pixel from the target voxels. The Manhattan distance is caculated between each pair of coordinates from actual voxel and target voxel.

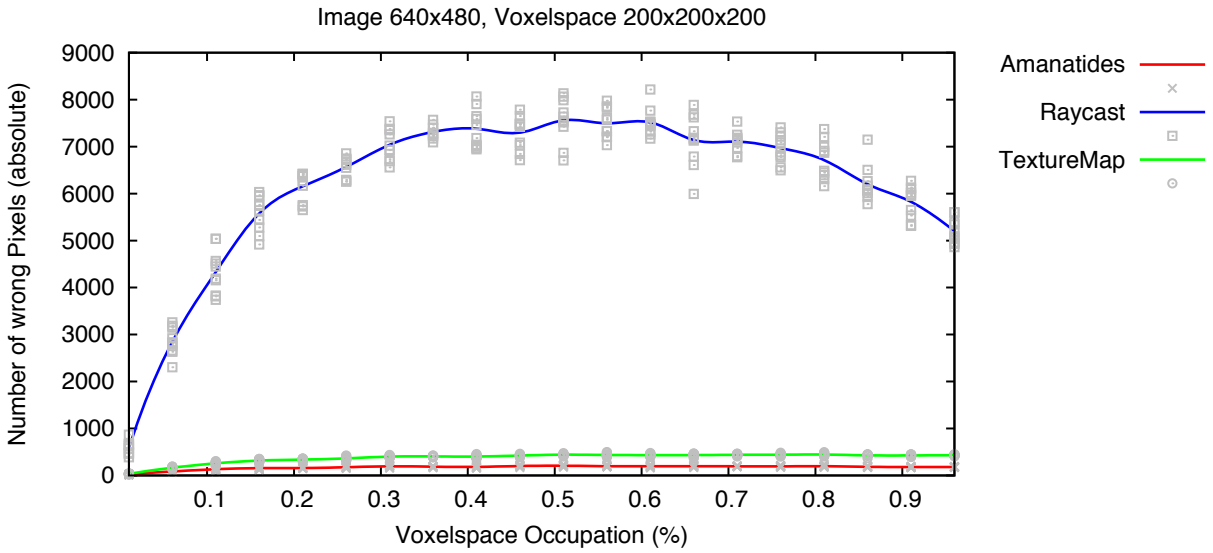


**Fig 12** Absolute number of wrong pixels compared to a naive rendering.

An example measurement for a voxelspace of size $200^3$ is shown in Fig. 12 and 13. As the compute shader implementation of the Amanatides does not yield a difference in the quality, the graph is identical to Amanatides and thus not shown in the Figures. Also the usage of segmented silhouette images has no influence on the quality and is not examined in this experiment.

The standard raycasting method produces the worst results. The voxel rendering is imprecise for 2.4 percent of all pixels. Even a ray marching step width of $1/5$th of the shortest voxel edge results in numerous wrong assignments of voxels to pixels with an average of 8 voxels (Fig. 13). This can influence the carving decision in the photo hull algorithm as the photo consistency test might be accomplished with wrong colors for these voxels. The Amanatides rendering achieves the best results with just 0.08 $\%$ wrong pixels (Fig. 12). The error can be neglected. An improvement of more than factor 10 in comparison to the raycasting can be expected. Also, the average deviation of 2...3 voxels is significantly lower for the Amanatides algorithm (Fig. 13) as for the raycasting. The texture mapping is a little worse than the Amanatides algorithms in the overall number of wrong assignments (not visible by eye), because it is completely conducted on the graphics board,

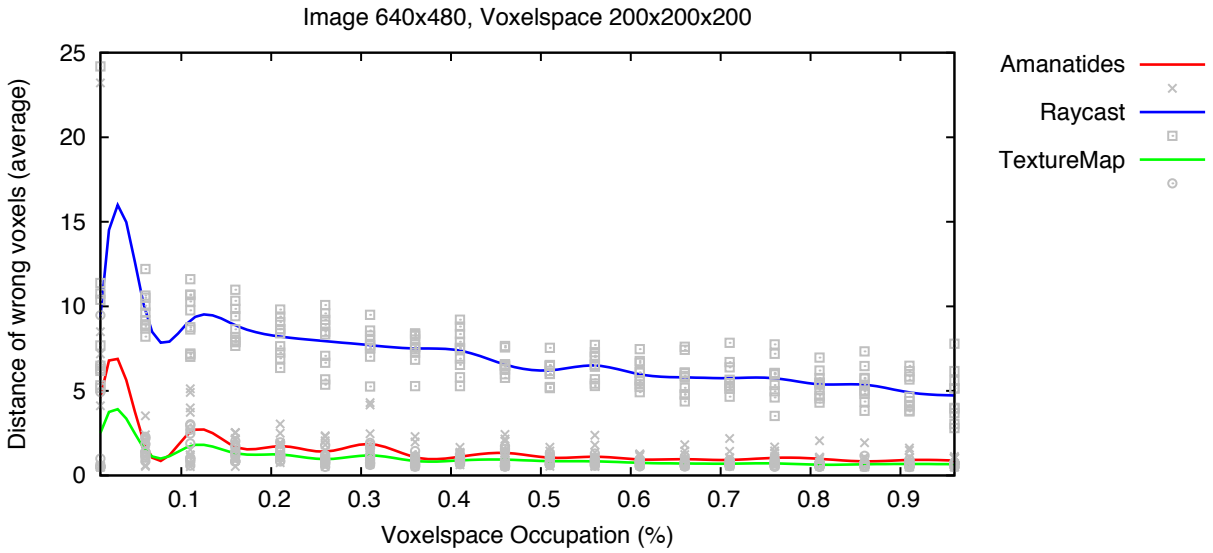


**Fig 13** Average Manhattan distance between rendered actual voxels and target voxels.

which is optimized in speed and not in accuracy. The average deviation is similar to the Amanatides algorithms.

In summary, the compute shader implementation of the Amanatides method achieved the best results in both computation time (rendering of 2000 images per second) and quality. Hence, this rendering method was applied in the following experiments.

### *4.2 Analysis of the Photo Hull*

The complete reconstruction pipeline was tested with image sequences of a real work cell as shown with seven calibrated[2] cameras of resolution $640 \times 480$ of type "Unibrain Fire-i400" and aperture angle of 120 degrees. Each camera is connected with a separate front-end processor that captures the frames and transfers the images via gigabit network to a main processor, which accomplishes the reconstruction. Image sequences with up to 4260 frames have been examined. All frames were used to produce the average results shown in the following Figures. Two or three persons with colored overalls (for better background subtraction) are moving in the cell that contains several

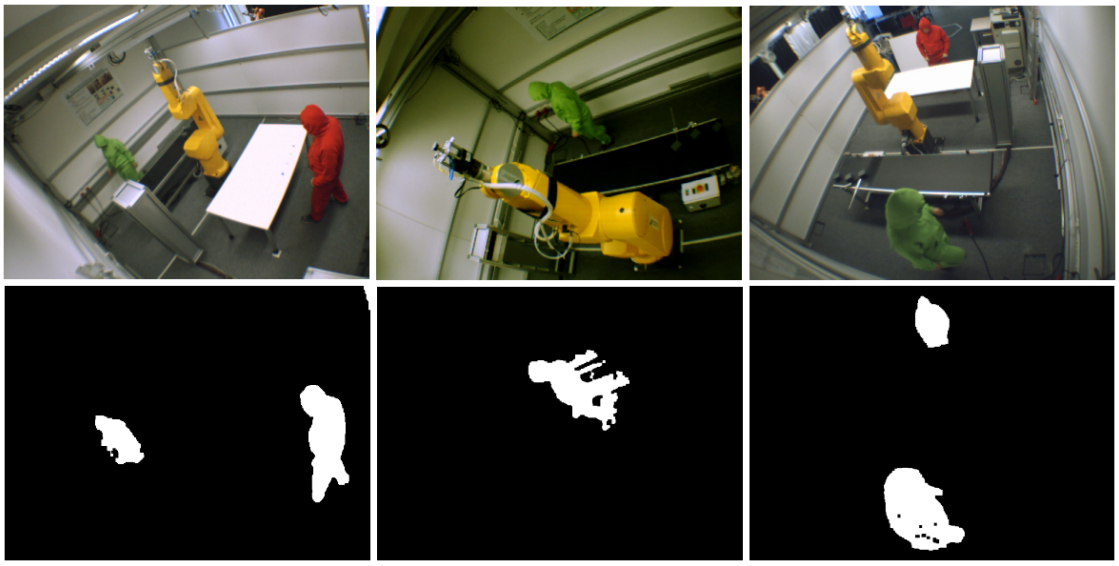

**Fig 14** Color images and silhouette images from background subtraction. Shown is our work cell with three different camera views. The results of the background subtraction are not always ideal.

occluding static obstacles, e.g. a robot, as shown in Fig. 14, which are modeled in 3D with triangle meshes and used to create depth images that are integrated in the reconstruction process for occlusion handling. The spatial extent of the voxelspace is chosen such that the work cell is filled completely, but walls, ceiling and cameras are not touched and thus not modeled. Additionally we examined the reconstruction results with a perfect ground truth and undistorted images given by a simulation of the work cell with one person moving around. First, we determined a fix upper bound of $800\ ms$ for computing the photo hull in some tests. That bound was selected because after elapsing that time, only little amount of voxels were carved, so that a further improvement of the reconstruction quality was negligible. Afterwards we examined the number of iterations that could be accomplished for the given limitation in processing time by varying the voxelspace resolutions ($100^3$ up to $300^3$) with a step size of $50^3$ and the number of cameras. The results are shown in Fig. 15.

By increasing the number of cameras by 75 percent (4 up to 7), the number of iterations reduced from 650 to 500, which are 23 percent less iterations. On the other hand, an enhancement

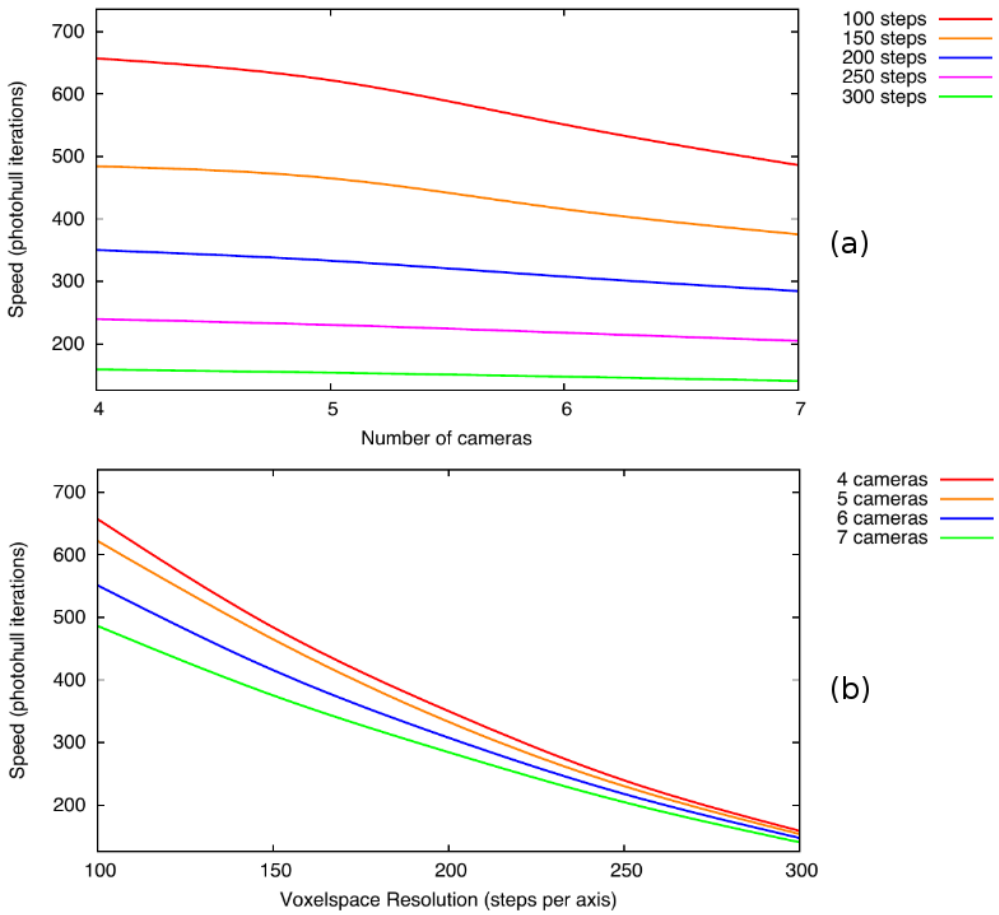


**Fig 15** Iterations of the photo hull for: (a) a varying number of cameras and (b) different voxelspace resolutions.

of the voxelspace resolution from $200^3$ to $250^3$ voxels results in a reduction of the number of iterations from 350 to 250 (28 percent). Surprisingly, the influence of the number of cameras is similar to the resolution of the voxelspace for the selected parameters, though the calculation of the photo consistency is done iteratively per image and should mainly depend on the number of cameras. A large part of the computation time (up to $50\%$ for $300^3$ voxels) is required for "preparation" and "resetting" the data structures. This is shown in Fig. 16 for the chosen $800\ ms$ processing time. After each frame, the voxel data structures $V_{free}$ and $V_{occ}$ have to be zeroed. Also, a reset is accomplished at the end of each iteration for the counter that is used for the incremental computation of the consistency test (LRT). This is accomplished for every voxel, so that the number of cameras is irrelevant, whereas an increase of the duration with the voxelspace resolution can be noticed from

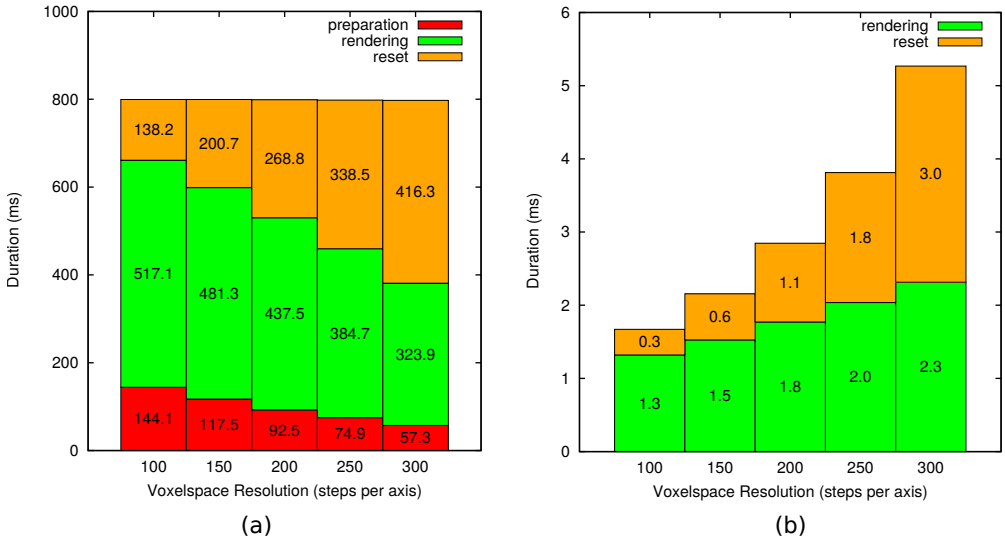


**Fig 16** Duration of the single computation steps of the photo hull given different voxelspace resolutions: (a) Average results over all cameras and all iterations. (b) Average results over all cameras for a single iteration.

Fig. 16. The used OpenGL standard of version 4.2 did not enable a better way for handling the preparation and reset accesses, but newer versions of OpenGL (version 4.4[40] or 4.5) should provide a more efficient handling. The computation time for rendering and updating the consistency of the single voxels also increases as expected with the voxelspace resolution (see "rendering" in Fig. 16). This correlation can be explained by the usage of the Amanatides raycasting as the step size for determination of the intersection points of viewing rays and voxels is adapted to the voxel size and thus to the voxelspace resolution.

Another experiment aimed in the applicability of the approach to our real-world work cell of $40\ m^3$ with $7$ cameras and a selected voxelspace resolution of $200^3$, with voxel side lengths of $21.5$, $19$ and $12\ mm$ each. We wanted to find out the highest frame rate with acceptable result. First, we determined the optimal threshold of $\tau = 7.5$ for the LRT consistency test given $800\ ms$ processing time (see Fig. 17). to get the best quality. Afterwards we reduced the computation time. As most voxels are carved within the first $200\ ms$, the slight refinements afterwards can be omitted. The achievable computation times are shown in Table 3. The values were averaged over


**Fig 17** Influence of the threshold for the LRT consistency test on the reconstruction quality. (a) Threshold is too low. The reconstruction is incomplete. (b) Good Threshold. All important details are maintained. (c) Threshold is too high. More Voxel are kept than necessary.

the first 200 frames for each sequence. A reconstruction performance of more than four frames per seconds could be reached for our scenario. For our sequences, the number of persons did not lead to a significant difference in computation time.

| **Sequence** | **Persons** | **Visual Hull** | **Photo Hull** | | **Total** | |
|---|---|---|---|---|---|---|
| | | | **Iteration** | **Total** | **Time** | **fps** |
| **Simulation** | 1 | 22 ms | 8.2 ms | 200 ms | 222 ms | 4,5 |
| **Real-world 1** | 2 | 29 ms | 5.1 ms | 200 ms | 229 ms | 4,3 |
| **Real-world 2** | 3 | 28 ms | 8.1 ms | 200 ms | 228 ms | 4,4 |

**Table 3** Computation times for the visual hull and the photo hull averaged over the first 200 frames for each sequence. The total processing time of the photo hull was limited to 200 $ms$. An online reconstruction of more than 4 frames per second (fps) can be expected for the given scenario.

Finally, we examined the quality of the photo hull. The photo integrity of Seitz and Dyer,[24] which demands that the projection of the reconstruction into the cameras reproduces the original images is mainly fulfilled, as shown in Fig. 18 and 19. Occluders are partially reconstructed, which might be caused due to discretization errors, an imprecise camera calibration, imprecise modeled obstacles or just artefacts from background subtraction. These effects occur less in the simulation sequence, shown in Fig. 19.

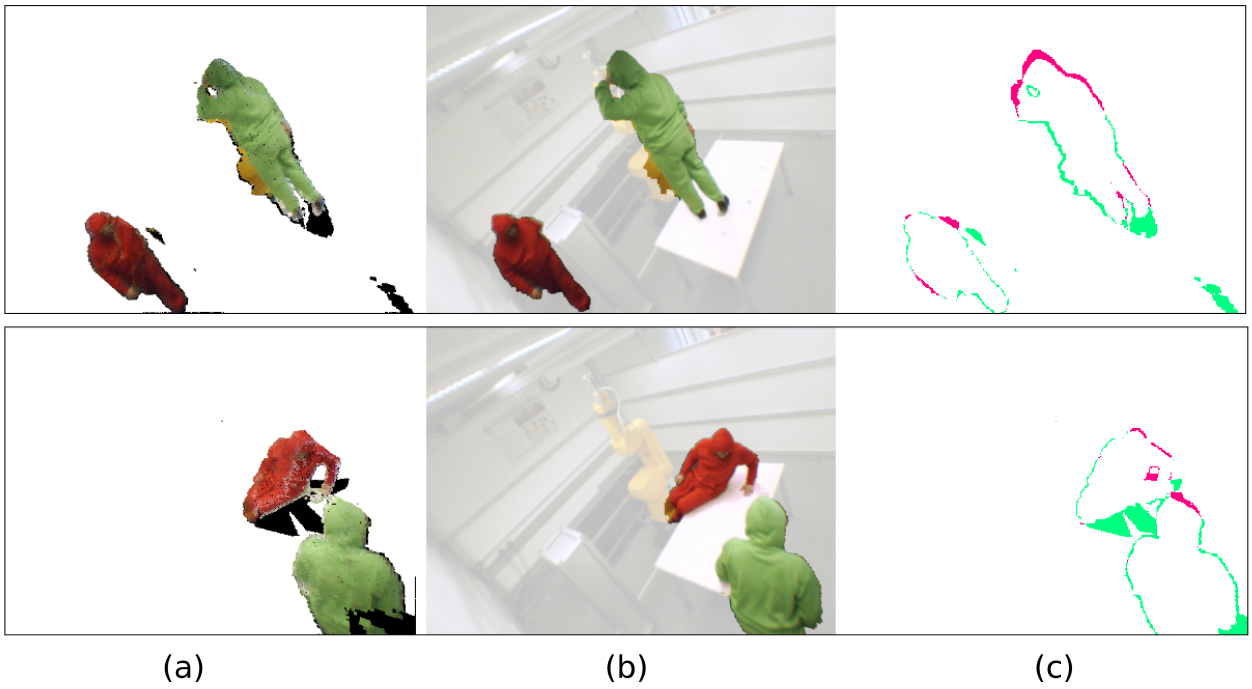


**Fig 18** Photo hull reconstruction of the real work cell: (a) Reconstruction results. (b) Input camera images with silhouettes from background subtraction. (c) Analysis of the objects' contours. Red pixels mark areas that are missing in the reconstruction. Green pixels mark areas that are reconstructed falsely. Artefacts occur due to discretization errors, an imprecise camera calibration and other distortions.

A further demand that is required for a high quality reconstruction of the photo hull,[24] is the broad viewpoint coverage of the camera sensors. Since we only apply 7 cameras, as expected, the reconstruction results suffer on the cell borders and other areas, where only little sensoric covering exists. This is shown in Fig. 20 (bottom). Nevertheless, the results in the center of the work cell are sufficient for our purpose, as shown in 20 (top).

## 5 Conclusions

We examined a photo hull algorithm for the online reconstruction of humans in environments that are characterized by the presence of occluding obstacles as well as a rather little amount of incorporated cameras. A new GPU implementation of the GVC-IB approach[5] is presented. Therefore an incremental calculation of the LRT consistency criterion is integrated. The termination of the

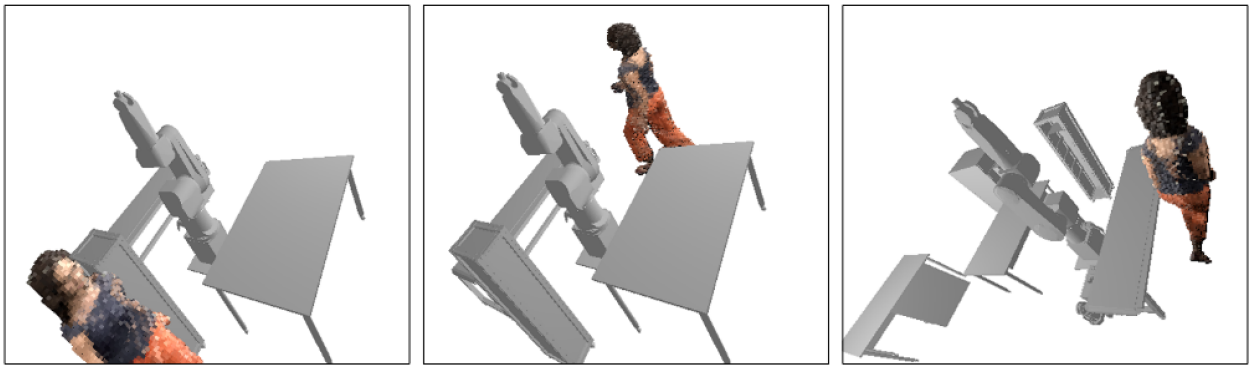

**Fig 19** Photo hull reconstruction of the simulation sequence. Less distortions influence the reconstruction result.

algorithm is managed with the anytime concept of Ref.[6] The input of the photo hull is a visual hull reconstruction. For the GPU we redesigned our visual hull algorithm, presented in Ref.,[4] that can handle conservatively known occluding obstacles. We analyzed different rendering techniques (texture mapping and raycasting approaches) for computing the required voxel-pixel projections. Our compute shader implementation of the Amanatides algorithm[39] yielded the best results concerning computation time and quality. Thus, the OpenGL pipeline is partially dispensable. A comparative implementation in OpenCL would be interesting. We examined the computation times depending on variations of the voxelspace resolution and amount of cameras.

A strong dependency on the amount of voxels was detected caused by expensive memory accesses for the voxels required during data structure preparation. This might be improved with newer OpenGL versions. We figured out that $200ms$ as upper bound for computing the photo hull (about 50 iterations) is sufficient for our scenario. An overall frame rate of $4.4fps$ can be reached. Furthermore, we examined the quality of the photo hull. The geometrical approximation of the visual hull could not be visibly improved for the given scenario with only 7 cameras and occluding obstacles. Nevertheless, the aimed colorization of the reconstruction fulfills the photo integrity assumption. In future work, we want to evaluate our GPU-based photo hull in the context of a large-scale robotics framework for real-time robotics applications.[41]

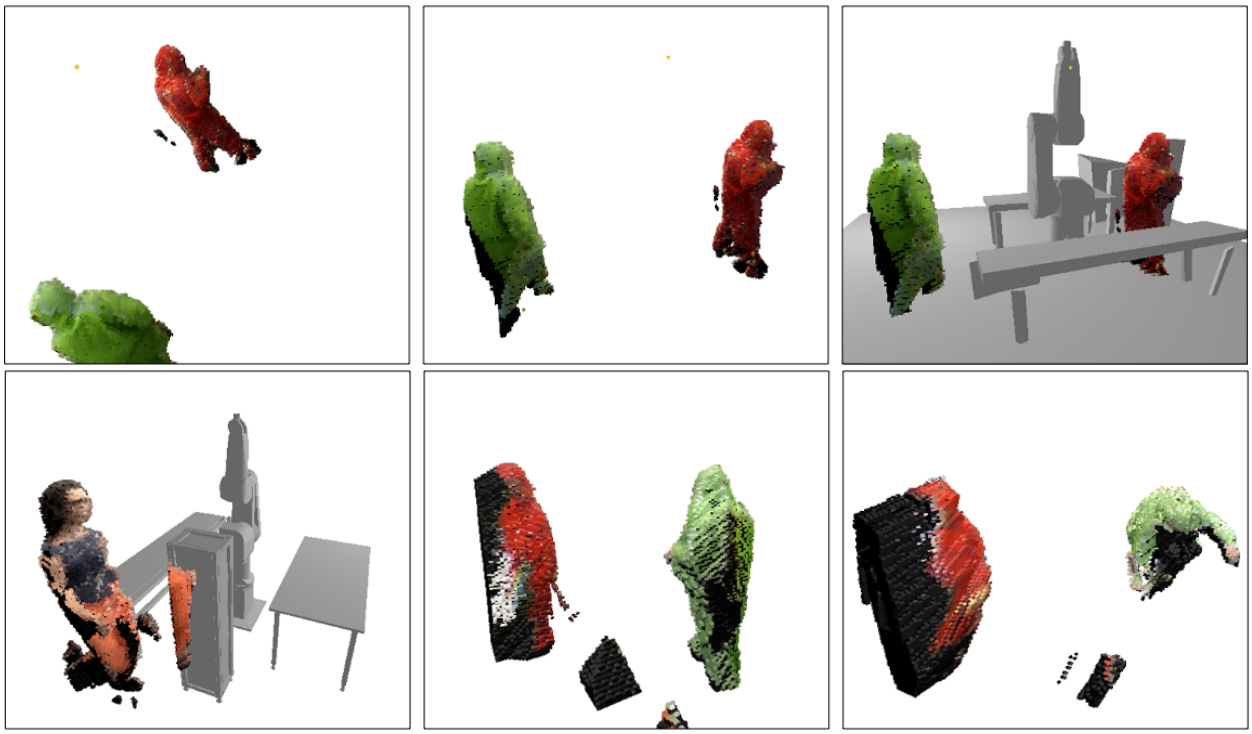

**Fig 20** Further reconstruction results of our real-world scenario from different views (top). Reconstruction artefacts at the cell borders, where the viewpoint coverage is insufficient (bottom).

*References*


1 Z. Zhang, "Microsoft kinect sensor and its effect," *IEEE Multimedia* **19**(2), 4–10 (2012).

2 T. Svoboda, D. Martinec, and T. Pajdla, "A convenient multicamera self-calibration for virtual environments," *Presence: Teleoperators and Virtual Environments* **14**(4), 407–422 (2005).

3 A. Ober-Gecks, M. Zwicker, and D. Henrich, "Efficient gpu photo hull reconstruction for surveillance," in *Proceedings of the International Conference on Distributed Smart Cameras*, *ICDSC '14*, 21:1–21:8, ACM, (New York, NY, USA) (2014).

4 S. Kuhn and D. Henrich, "Multi-view reconstruction of unknown objects in the presence of known occlusions," in *ISVC '09 Proceedings of the 5th International Symposium on Advances in Visual Computing: Part I*, 784–795, Springer Verlag, Berlin (2009).

5 W. B. Culbertson, T. Malzbender, and G. Slabaugh, "Generalized voxel coloring," in *Vision*

*Algorithms: Theory and Practice*, B. Triggs, A. Zisserman, and R. Szeliski, Eds., **1883**, 100–115, Springer Berlin Heidelberg (1999).

6 T. Dean and M. Boddy, "An analysis of time-dependent planning," in *Proceedings of the seventh national conference on artificial intelligence*, 49–54 (1988).

7 A. Laurentini, "The visual hull concept for silhouette-based image understanding," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **16**(2), 150–162 (1994).

8 B. G. Baumgart, "Geometric modeling for computer vision," AIM-249, STA -CS-74-463, CS Dept, Stanford U. (1974).

9 L. A., "The visual hull: A new tool for contour-based image understanding," *Proc. 7th Scandinavian Conf. Image Analysis* , 993–1002 (1991).

10 A. Laurentini, "How far 3d shapes can be understood from 2d silhouettes.," *IEEE Trans. Pattern Anal. Mach. Intell.* **17**(2), 188–195 (1995).

11 J.-S. Franco and E. Boyer, "Efficient polyhedral modeling from silhouettes," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **31**(3), 414–427 (2009).

12 M. Fischer and D. Henrich, "Surveillance of robots using multiple colour or depth cameras with distributed processing," *Third ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC 2009), Aug 30* (2009).

13 S. Lazebnik, Y. Furukawa, and J. Ponce, "Projective visual hulls," *Int. J. Comput. Vision* **74**, 137–165 (2007).

14 K. N. Kutulakos and S. M. Seitz, "A theory of shape by space carving," *International Journal of Computer Vision* **38**(3), 199–218 (2000).

15 A. Ladikos, S. Benhimane, and N. Navab, "Efficient visual hull computation for real-time 3d reconstruction using cuda.," in *CVPR Workshops*, 1–8, IEEE (2008).

16 J. R. Casas and J. Salvador, "Image-based multi-view scene analysis using 'conexels'," in *Proceedings of the HCSNet Workshop on Use of Vision in Human-computer Interaction - Volume 56*, *VisHCI '06*, 19–28, Australian Computer Society, Inc., (Darlinghurst, Australia, Australia) (2006).

17 R. J. Radke, S. Andra, O. Al-Kofahi, and B. Roysam, "Image change detection algorithms: A systematic survey," *IEEE Transactions on Image Processing* **14**, 294–307 (2005).

18 L. Guan, J.-S. Franco, and M. Pollefeys, "3d object reconstruction with heterogeneous sensor data," *4th International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT), Atlanta, GA, USA* **2** (2008).

19 M. A. Keck and J. W. Davis, "3d occlusion recovery using few cameras," in *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008), 24-26 June 2008, Anchorage, Alaska, USA*, (2008).

20 D. Ebert and D. Henrich, "Safe human-robot-cooperation: Image-based collision detection for industrial robots," *IEEE/RSJ International Conference on Intelligent Robots and Systems, Lausanne, September 30th-October 5th, 2002* (2002).

21 S. Kuhn, T. Gecks, and D. Henrich, "Velocity control for safe robot guidance based on fused vision and force/torque data," *IEEE Conference on Multisensor Fusion and Integration for Intelligent Systems, Heidelberg, Germany, September 03-06, 2006* (2006).

22 A. Ober and D. Henrich, "A safe fault tolerant multi-view approach for vision-based protective devices," in *Seventh IEEE International Conference on Advanced Video and Signal*

*Based Surveillance, AVSS 2010, Boston, MA, USA, August 29 - September 1, 2010*, 17–25 (2010).

23 A. Schick and R. Stiefelhagen, "Real-time GPU-based voxel carving with systematic occlusion handling," in *Pattern Recognition. 31st DAGM Symposium, Jena, Germany, September 9-11, 2009. Proceedings*, 372–381 (2009).

24 S. M. Seitz and C. R. Dyer, "Photorealistic scene reconstruction by voxel coloring," *International Journal of Computer Vision* **35**(2), 151–173 (1999).

25 G. G. Slabaugh, W. B. Culbertson, T. Malzbender, and R. W. Schafer, "A survey of methods for volumetric scene reconstruction from photographs," in *VG'01 Proceedings of the 2001 Eurographics conference on Volume Graphics*, 81–101 (2001).

26 G. G. Slabaugh, W. B. Culbertson, T. Malzbender, M. R. Stevens, and R. W. Schafer, "Methods for volumetric reconstruction of visual scenes," *International Journal of Computer Vision* **57**(3), 179–199 (2004).

27 C. Nitschke, *A Framework for Real-time 3D Reconstruction by Space Carving using Graphics Hardware*, Grin Verlag, München (2007).

28 A. Prock and C. Dyer, "Towards real-time voxel coloring," in *Proceedings of the DARPA Image Understanding Workshop*, 315–321 (1998).

29 T. Werner and D. Henrich, "Efficient and precise multi-camera reconstruction," in *Eighth ACM/IEEE International Conference on Distributed Smart Cameras - ICDSC, November 4-7, Venice*, (2014).

30 M. Sainz, Bagherzadeh, Nader, and Susin, Antonio, "Hardware accelerated voxel carving," in *1st Ibero-American Symposium in Computer Graphics (SIACG 2002)*, 289–297 (2002).

31 O. Batchelor, R. Mukundan, and R. Green, “Ray casting for incremental voxel colouring,” in *New Zealand: International Conference on Image and Vision Computing - IVCNZ05*, 206–211 (2005).

32 M. Zwicker, “Erweiterung der Photohülle zur schnellen Onlinerekonstruktion auf moderner Grafikhardware,” master thesis, University of Bayreuth, Bayreuth (2014).

33 “OpenCV open source computer vision.” http://opencv.org/. Accessed: 2016-04-21.

34 T. Finch, “Incremental calculation of weighted mean and variance,” *University of Cambridge* (2009).

35 J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell, “A survey of general-purpose computation on graphics hardware,” *Computer Graphics Forum* **26**(1), 80–113 (2007).

36 M. Meissner, U. Hoffmann, and W. Strasser, “Volume rendering using OpenGL and extensions,” in *Visualization ’99. Proceedings*, 207–526 (1999).

37 R. Fernando and NVIDIA Corporation, *GPU gems: programming techniques, tips, and tricks for real-time graphics*, Addison-Wesley, Boston MA, 5. ed. (2004).

38 J. Kruger and R. Westermann, “Acceleration techniques for GPU-based volume rendering,” *IEEE Visualization Conference* , 287–292 (2003).

39 J. Amanatides and A. Woo, “A fast voxel traversal algorithm for ray tracing,” in *Proceedings of EUROGRAPHICS*, **87**, 3–10 (1987).

40 M. Segal and K. Akeley, “OpenGL 4.3 core profile specification,” (2013).

41 T. Werner, M. Gradmann, D. Henrich, E. M. Orendt, M. Sand, and M. Spangenberg, “Enact:

An efficient and extensible entity-actor framework for modular robotics software components," *47th International Symposium on Robotics (ISR)* (2016).

# List of Figures

**List of Algorithms**



# List of Tables



**The Authors**

**Antje Ober-Gecks** received her degree as Diploma engineer in media technology from Technical University of Ilmenau (Germany) in 2007. She took her studies with emphasis on computer science and neuro-informatics. Since 2008, she is research assistent at the University of Bayreuth. She is working on her doctoral thesis with the topic of person tracking under occlusions based on 3D reconstruction data. Her interests include pattern recognition, image processing and computer graphics.

**Marius Zwicker** received the BS degree in applied computer science from the University of Bayreuth (Germany) in 2011 and the MS degree in applied computer science from the same university in 2013. He then joined Garmin Wuerzburg GmbH, the European research and development center of Garmin Ltd. His interests include computer graphics and software architecture with a focus on software defined video composition.

**Prof. Dr. Dominik Henrich** finished his Doctorate in 1994. From 1996 to 1999, he built up a research group at the University of Karlsruhe (Germany). From 1999 to 2003, he headed as professor a research group at the University of Kaiserslautern. Since 2003 he holds the chair for Robotics and Embedded Systems at the University of Bayreuth. His research interests are e.g. collision detection, motion planning, room surveillance, sensor-based manipulation and intuitive robot programming.