Efficient GPU Photo Hull Reconstruction for Surveillance

Antje Ober-Gecks
Angewandte Informatik III
Universität Bayreuth
Germany
antje.ober-gecks@uni-bayreuth.de

Marius Zwicker
Angewandte Informatik III
Universität Bayreuth
Germany
marius.zwicker@mlba-team.de

Dominik Henrich
Angewandte Informatik III
Universität Bayreuth
Germany
dominik.henrich@uni-bayreuth.de

ABSTRACT

We present a GPU-based implementation of the photo hull using the *generalized voxel coloring with item buffer* (GVC-IB) approach with the aim of reconstructing humans online in surveillance scenarios. To allow for a fast computation on the GPU, an incremental calculated standard deviation is used in the *likelihood ratio test* that is applied as color consistency criterion. Concerning the necessary fast and efficient computation of complete voxel-pixel projections, a number of volume rendering methods is examined, such as texture mapping and raycasting. The termination of the voxel carving during photo hull computation is realized by integrating the anytime concept. The input of the voxel carving is a visual hull reconstruction. An algorithm that provides an exact and conservative occlusion handling is redesigned for a fast computation on the GPU.

Keywords

voxel coloring, voxel carving, space carving, photo hull, conservative visual hull, occlusions, obstacles, reconstruction

1. INTRODUCTION

A common goal is the precise localization of humans in complex environments such as smart homes and industrial areas in order to guarantee the safety of humans, e.g. in human-robot-cooperation. Active sensors such as laser scanner, sonar, or recently the depth camera Kinect are used to approximate the localization. However, color cameras still might be preferred due to their passive information extraction from the scene as well as to their low cost.

Our system consists of multiple calibrated and synchronized color cameras, which monitor a common surveillance volume. This volume contains known *obstacles* such as tables and racks (Figure 1). The goal is to determine the subvolume occupied by humans, called *objects*. Therefore methods are employed that reconstruct online a 3D model of the surveillance volume for further analysis. The result is typically represented by voxel data. Although voxels are affected by discretization errors, they have a variety of advantages like easy access, storage, and management as well as independent processability and little numerical problems (e.g. in comparison to polyhedrons). A commonly used method is the reconstruction of the *visual hull* (shape from silhouette) [9]. This method uses segmented images to reconstruct the original geometry of objects by simply back-projecting all segmented silhouette pixels and intersecting the resulting cones. A disadavantage of the visual hull is that only geometrical information is provided. Though a variety of coloring methods for postprocessing exist, the best photo realistic colorization can be expected by an alternative reconstruction method, namely the *color reconstruction* [16]. This method also is known as *space carving* [7] (or more general *voxel carving*). The reconstruction result is called *photo hull*. The color information is integrated into the reconstruction process in due consideration of actual voxel visibilities in the images. Therewith the color information of the images is directly assigned to different voxels in 3D space. The photo hull has to fulfill the criteria of *photo integrity*, which means the reconstruction result has to be such that the projection into the camera images reproduces the original input images. Adversely, long time the photo hull was improper to process image sequences of real-time scenarios due to frame rates of several minutes, e.g. in [18]. However, modern graphics hardware promises new possibilities for accelerating the photo hull computation as provided by [10].

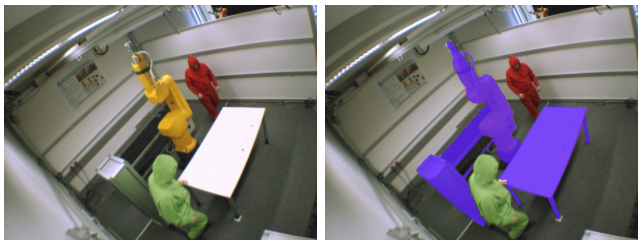

Figure 1: Work cell with unknown *objects* (humans) and known static *obstacles* (colored violet).

In this paper we present a redraft of the *generalized voxel coloring with item buffer* (GVC-IB) [3] for the usage as GPU online reconstruction method, shown in Section 3.5. In order to achieve better runtimes, the input of the voxel carving is a reconstructed visual hull that processes the silhouette images of a background subtraction method at every time step t. Furthermore, a segmentation of the objects is required for our work cell as the equally colored walls lead to a premature

.

termination of the voxel carving otherwise. For this purpose we selected a state of the art visual hull algorithm that can handle occluding obstacles in the scenes and that realizes an exact and conservative reconstruction [6]. We provide a GPU-based adaptation of this method for a faster computation, provided in Section 3.4. 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. Therefore, different rendering techniques (texture mapping, raycasting) are applied. An experimental evaluation of these approaches is described in Section 4. Our contribution is summarized in Section 5. The following Section 2 gives an overview of related work.

2. STATE OF THE ART

Visual Hull: As already stated, we require a visual hull algorithm that can handle occluding obstacles in the scene. Occlusions are an issue that might lead to clipped silhouette images in the background subtraction step and thus result in an incomplete reconstruction. We found different approaches that handle occlusions. One method uses manually created segmentations of the obstacles in the images (occlusion masks) additionally to the silhouettes of the background subtraction [8]. The reconstruction is the intersection of all backprojected pixel cones. A better approximation of the objects and obstacles volume can be expected with the solution of [14] and our solution in [6]: For each camera a depth map is employed that determines the free space up to the surfaces of the obstacles similar to a range sensor.

Photo Hull: The process of reconstructing a photo hull starts with a fully occupied voxel space. Visible voxels are carved iteratively until the objects in the scene are approximated. A voxel is visible in a pixel if no other voxel is in front of that voxel. The visibility of a voxel might change with every carved voxel in the voxelspace and thus has to be determined again in each iteration. The decision for carving a visible voxel is based on the concept of color consistency. A voxel is consistent if the respecting pixels of all cameras for which it is visible (a subset of all projection pixels) have the same color, otherwise it is carved. An overview of different consistency criteria, e.g. the maximum norm, can be found in [18].

Different concepts of voxel carving distinguish in the determination of the visibility. A naive and simple algorithm with high computation cost is proposed in [17]. This cost can strongly be reduced by introducing a limitation to the camera positions, called *ordinal visibility constraint* [16]. All cameras have to be placed behind a parting plane so that the voxels can be processed in a fix order in one iteration. Disadvantageous is the incomplete reconstruction from one perspective. In comparison, algorithms of the *partial visibility space carving* (PVSC) and *full visibility space carving* (FVSC) [18] apply a sweep subsequently along all three coordinate axes of the voxelspace in positive and negative directions, beginning from the external borders. Only the active cameras of the current sweep plane are incorporated in the consistency test. All of the mentioned approaches appoximate the ordinal visibility constraint but do not provide an actual visibility test. An exact computation with arbitrary camera placement is provided with the *generalized voxel coloring* (GVC) [3]. The visibility of the voxels is managed with help of a *surface voxel list* (SVL). 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 the use of the assigned pixels from the item buffer. Whenever a voxel from the SVL is carved, it is replaced by its 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) [3], whereas the algorithmic complexity and the memory usage are increasing.

(Hardware) Acceleration: In [12] 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 [19]. In [13] 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 enhance the computation time of voxel-pixel projections [2]. 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 [10]. 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 1024×768 a frame rate of 33 fps is achieved.

3. OUR APPROACH

So far, to our knowledge, the existing (accelerated) approaches lack at minimum in one of the following aspects: (a) Explicit consideration of occluding obstacles in the process of reconstructing the visual hull. (b) Projection of the complete voxel volumes to the images, instead of using the voxel center [10] or another simplification [8]. (c) Computation of exact voxel visibilities for the use of arbitrary camera placements in voxel carving.

In our approach we consider all of these aspects. For aspect (a) we adapted our algorithm of [6] and redesigned it for hardware acceleration on the GPU, described in section 3.3. To our knowledge it is the most general approach with conservative volume approximation of objects and obstacles in the presence of occlusions. Aspect (b) also is covered in [6] by using a look-up table that holds the voxel-pixel correspondences similar to [8]. In order to realize the processing of higher resolutions of the voxelspace and the images, we replace this method by a GPU-supported rendering method (raycasting). Finally, aspect (c) is realized by a redraft of the GVC-IB voxel carving [3] (In [2] the usage of graphics boards for acceleration of the GVC approaches also is suggested). We apply the *likelihood ratio test* (LRT) as pro-

posed in [16] as consistency criterion. In pretests the LRT achieved the best reconstruction quality (compared to the ASDT, histogram and maximum norm [18]). An incremental computation of the standard deviation in the LRT permits a fast parallelization of the incremental photo hull on the GPU. The integration of the anytime concept [4] allows for the termination of the algorithm after a defined maximum computation time. For hardware acceleration we appreciated using an open standard and decided to apply OpenGL due to the rendering of plenty of images to get the voxel-pixel projections in the cameras. More details to our approach and the results can be found in [20].

In this Section we describe our GPU implementations of the visual hull und the photo hull, starting with each CPU equivalent in advance, for better understanding.

3.1 Standard Visual Hull

Given is our system with multiple calibrated and synchronized color cameras C_i at positions $\mathbf{p}_{C,i}$ with the images $P_{C,i}$, $i = 1, ..., n$. A visual hull $\mathrm{VH}(P_{C,1}, ..., P_{C,n})$ is reconstructed from the a priori unknown humans, called *objects*, represented as a set of voxels $x \in \mathrm{VH}(P_{C,1}, ..., P_{C,n}) \subset \mathbb{R}^3$. At first, the camera images are processed by a background subtraction method. The resulting segmented n silhouette images $M_{C,i}$ are used as input for the visual hull.

The value $\mathrm{color}(p, M_{C,i})$ of a pixel p in the silhouette images indicates the foreground ($= 1$) respectively the background ($= 0$). Let $\pi_{x,i}$ be the set of pixels, which show the projection of a voxel x in Camera i. The voxels of the visual hull are defined by all those voxels $x \in \mathrm{VH}(P_{C,1}, ..., P_{C,n})$, which hold for the Formula (1).

$$(\mathrm{color}(p, M_{C,i}) = 1)\ \forall p \in \pi_{x,i}, \forall i \in \{1, ..., n\} \quad (1)$$

Given the pixel sets $\pi_{x,i}$ for all voxels, the visual hull can be determined by set operators in terms of boolean expressions. A simple algorithm of the standard visual hull given $\pi_{x,i}$ is shown in Figure 2. A voxel marked "UNCARVED" is visible and occupied. A voxel that is carved is discarded. It is empty and not visible anymore.

```
1 def visualHull(V)
2   V.set_all( UNCARVED )
3   V.each do |x| # for each voxel
4     π_x.each do |p| # for each pixel
5       if( color(p,M_C,i) == 0 )
6         carve x
7     end
8   end
9 end
```

Figure 2: Algorithm of the standard visual hull

3.2 Visual Hull with Occluding Obstacles

For the standard visual hull, projections of objects are assumed to be completely included in the silhouettes of the n segmented images $M_{C,i}$. Using an online surveillance system with ideal background subtraction method, this is true for all objects that enter an empty surveillance volume after capturing the reference images. 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 targeted objects such that clipped silhouettes arise. In result Formula (1) is wrongly not true and it is not guaranteed that the remaining visual hull contains the complete objects. We provide a solution of this problem in [6]. 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. The idea of [6] is now briefly described.

The obstacles are geometrically modeled, e.g. as triangle meshes. With help of those, depth images are created for all cameras with the same resolution as the other images. Each *depth pixel* contains the distance to the closest surface of the obstacles. 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. Lets define the function $\mathrm{depth}(p)$ that returns the maximum free range of a pixel p. Given the position $\mathbf{p}_{C,i}$ of the camera i, the center of a voxel $\mathbf{p}_x$ and a constant d_v describing half of the length of the diagonal of a voxel (to be conservative), the Formula (1) can be extended such that, for all voxels of the visual hull $x \in \mathrm{VH}(P_{C,1}, ..., P_{C,n})$ holds:

$$(\mathrm{color}(p, M_{C,i}) = 1) \vee (|\mathbf{p}_x - \mathbf{p}_{C,i}| + d_v \geq \mathrm{depth}(p)) \quad \forall p \in \pi_{x,i}, \forall i \in \{1, ..., n\} \quad (2)$$

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. The algorithm in Figure 2 can be adjusted by replacing Line 5 with:

```
1 if( (|p_x − p_C,i| + d_v < depth(p)) &&
2     (color(p,M_C,i) == 0) )
```

3.3 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 $p \in \pi_{x,i}$ of a camera with value $\mathrm{color}(p, M_{C,i}) = 0$ (free) suffices to carve a voxel, though all other projection pixels might have the value $\mathrm{color}(p, M_{C,i}) = 1$. 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 provided:

$$\exists i \in \{1, ..., n\}, \forall p \in \pi_{x,i}: \quad (\mathrm{color}(p, M_{C,i}) = 0) \wedge (|\mathbf{p}_x - \mathbf{p}_{C,i}| + d_v < \mathrm{depth}(p)) \quad (3)$$

A voxel is only carved if all projection pixels $\pi_{x,i}$ of at least one camera are completely free, whereas no occluded or occupied pixel is allowed (Formula (3)). On the other hand, one occupied or one occluded pixel in each camera is sufficient to keep the voxel in the reconstruction (Formula (4)).

$$\exists p \in \pi_{x,i}, \forall i \in \{1, ..., n\}: \quad (\mathrm{color}(p, M_{C,i}) = 1) \vee (|\mathbf{p}_x - \mathbf{p}_{C,i}| + d_v \geq \mathrm{depth}(p)) \quad (4)$$

An according algorithm is shown in Figure 3.

```
1  def visualHullConservative(V)
2    V.set_all( UNCARVED )
3    V.each do |x|
4      Cameras.each do |C_i|
5        occupied = nil
6        occludedPixelExist = false
7        π_{x,i}.each do |p|
8          if( |p_x − p_{C,i}| + d_v < depth(p) )
9            if( color(p,M_{C,i}) == 1 )
10             occupied = true
11           else if( occupied == nil )
12             occupied = false
13         else
14           occludedPixelExist = true;
15       end
16       if( (occupied == false) &&
17           (occludedPixelExist == false) )
18         carve x
19     end
20   end
21 end
```

Figure 3: Algorithm for a conservative visual hull with occlusion handling.

3.4 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. All images are transferred to the GPU and pixels are processed in parallel.

```
1  def processVoxelInPixel(x_free,x_occ,p,M_{C,i})
2    if( (color(x_occ) >= color(x_free)) ∨
3        (color(x_free) == i) )
4      if( (color(p,M_{C,i}) != 0) ∨
5          (|p_x − p_{C,i}| + d_v >= depth(p)) )
6        color(x_occ) = i
7      else
8        color(x_free) = i
9  end
```

Figure 4: Algorithm for a conservative visual hull with occlusion handling on the GPU.

We create two equal sized voxelspaces V_{free} and V_{occ} that have the properties of the original voxelspace. Each voxel holds a single number out of $\mathbb{N}$ that is initialized with 0. Each camera is assigned an ID $i \in \mathbb{N}$. The computation takes place sequentially for one camera after the other, sorted by ascending i, but parallel for all pixels. Each pixel handles all voxels that are included by its backprojection cone in a sequential way. The process executed for each pixel and each voxel is shown in Figure 4. Basically every voxel gets for each projection pixel an entry of the current camera ID in either V_{free} or V_{occ}. The pixel-parallelism is ensured by the special construction of the condition in Figure 4, 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 synchronization. Once a voxel is detected as free

$\text{color}(x_{occ}) < \text{color}(x_{free})$
Camera n or a previous camera ($< n$) contains only free pixels. One camera that sees the voxel completely as free is sufficient to carve the voxel. Thus the voxel is **free**.
$\text{color}(x_{occ}) \geq \text{color}(x_{free})$
At least one occupied or occluded pixel was recorded for the last camera n. This indicates that no previous camera ($< n$) sees the voxel completely as free. The voxel is **occupied**.

Table 1: Final values of each voxel in V_{free} and V_{occ} and the resulting voxel occupation.

in one camera, the condition is never true for the following cameras. The resulting visual hull can be readout by comparing the two voxelspaces after the processing of all pixels and voxels, as shown in Table 1.

```
1  def processVoxelInPixel(x_free,x_occ,p,M_{C,i})
2    if( |p_x − p_{C,i}| + d_v >= depth(p) )
3      return
4    if( (color(x_occ) >= color(x_free)) ∨
5        (color(x_free) == i) )
6      if( color(p,M_{C,i}) != 0 )
7        color(x_occ) = i
8      else
9        color(x_free) = i
10 end
```

Figure 5: Algorithm for a conservative visual hull with occlusion handling on the GPU. Obstacles are not part of the visual hull.

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). For this reason a modified

$\text{color}(x_{occ}) < \text{color}(x_{free})$
Camera n or a previous camera ($< n$) contains only free pixels besides possibly occluded pixels. This is sufficient to carve the voxel. Thus the voxel is **free**.
$(\text{color}(x_{occ}) \geq \text{color}(x_{free})) \wedge (\text{color}(x_{occ}) > 0)$
At least one occupied pixel was recorded for the last camera n. This indicates that no previous camera ($< n$) sees the voxel as free. The voxel is **occupied**.
$(\text{color}(x_{occ}) = 0) \wedge (\text{color}(x_{free}) = 0)$
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_{free} and V_{occ} and the resulting voxel occupation. Obstacles are not part of the visual hull.

algorithm is shown in Figure 5. The difference is that occluded pixels are now ignored completely (Lines 2 and 3). The resulting values of the two voxelspaces V_{free} or V_{occ} have to be interpreted as shown in Table 2.

3.5 Photo Hull on the GPU

```
1  def GVCIB(V,τ)
2    determine SVL
3    do
4      project Voxels of SVL, collect π_x
5      SVL.each do |x|
6        compute μ_x, LRT of π_x
7        if( LRT < τ )
8          color(x) = μ_x
9        else
10         carve x
11         add neighbours of x to SVL
12     end
13   while voxel was carved
14 end
```

Figure 6: Algorithm of the GVC-IB [3] with likelihood ratio test (LRT) as consistency criterion.

Our GPU algorithm of the photo hull can be found in Figure 7. It is similar to the GVC-IB approach of [3] in Figure 6. In the latter method, at the beginning of each iteration the surface voxels are determined and stored in the *surface visibility list* (SVL), which contains the outer voxels of the voxelspace in the first iteration (or the outer voxels of the visual hull if used as input data). For each of the surface voxels the consistency test is conducted to decide whether the voxel is really located on an objects surface. This is true if all projection pixels in which the voxel is visible have the same color. The voxel remains uncarved in this case, otherwise the voxel is carved and replaced in the SVL by the next potential surface voxels. The SVL 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 the following subsection). Therewith in each iteration all the currently occupied voxels of the voxelspace V_t are projected into the cameras, while applying a *visibility transfer function* to each voxel (Formula 5), similar to the *volume rendering* used in other application fields.

$$f(x,p) = \begin{cases} (x[0],x[1],x[2],1) & \text{if}((\text{occ}(x)>0) \wedge (\text{depth}(p) \\ & \geq |\mathbf{p}_x - \mathbf{p}_{C,i}|)),\ p \in P_{C,i} \\ (0,0,0,0) & \text{else} \end{cases} \tag{5}$$

As our visual hull contains objects but no obstacles, additionally we have to consider the occlusions in Formula 5 by applying the information of the depth maps similar to the usage in the visual hull algorithm. After rendering, the resulting values R,G,B of each pixel p encode the coordinates of the closest visible voxel ($x = \text{vis(pixel)}$ in Figure 7). 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. Each carved voxel (Figure 7, line 12) gets the entry $\text{occ}(x) = 0$, 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

```
1  def parallelGVCIB(V_t,τ,duration)
2    start = Time.now
3    do
4      render occupied voxels of V_t and
5      obtain visibility images VI
6      VI.each.pixels do |pixel|
7        x = vis(pixel)
8        update μ_x, n·(σ_x)^2
9        if( ((n−1)/n)·(n·(σ_x)^2) < τ )
10         color(x) = μ_x
11       else
12         carve x  # occ(x) = 0
13     end
14   while Time.now - start < duration
15 end
```

Figure 7: Algorithm of the GVC-IB on the GPU

reasonable for the consistency test in each carving iteration.

The *likelihood ratio test* (LRT) is applied as consistency criterion, which employs the standard deviation σ_x. Uncarved voxels are colored with the average value μ_x of all colors that are assigned to each voxel. The mean value of the pixel colors μ_x and the associated standard deviation σ_x is calculated incrementally to avoid dynamic data structures, too. The original definition of the mean value requires all n elements v_i in advance.

$$\mu_x = \frac{1}{n}\sum_{i=1}^{n} v_i \tag{6}$$

This can be replaced by the following incremental Formula:

$$\mu_x^n = \frac{1}{n}(v_n - \mu_x^{n-1}) + \mu_x^{n-1} \tag{7}$$

The original definition of the standard deviation is:

$$\sigma_x^n = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(v_i - \mu_x)^2} \tag{8}$$

Its incremental equivalent can be expressed as in [5]:

$$n\cdot(\sigma_x^n)^2 = (n-1)\cdot(\sigma_x^{n-1})^2 + n\cdot(n-1)\cdot(\mu_x^n - \mu_x^{n-1})^2 \tag{9}$$

As can be seen from the Formula, the computation of $n \cdot (\sigma_x^n)^2$ 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 Formula (9) with the definition for the consistency test LRT, which is given as $\text{LRT} = (n-1)\cdot(\sigma_x)^2$, yields a function following the form of $g(n) \cdot f(n)$. As such, the LRT can be computed incrementally using the following term, which is monotonously increasing in n:

$$\text{LRT}(n) = \left[\frac{n-1}{n}\right] \cdot \left[n\cdot(\sigma_x^n)^2\right] \tag{10}$$

Due to this adaptation, the consistency test can now be accomplished parallel for all pixels in each iteration. Whenever $LRT > \tau$, the voxel can be carved immediately as it can not

become smaller anymore due to its monotony. Solely the update of μ_x and $n \cdot (\sigma_x)^2$ 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 "`while voxel was carved`" of the GVC-IB (Figure 6, Line 13) because this requires a feedback of the graphics board that currently only can be realized with a trick [11]. 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 proposed by [4] 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. amount 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, clocked with 4 GHz und access of 32GB RAM. We chose a fix camera resolution of 640 × 480 pixels for all experiments.

4.1 Analysis of the Rendering Methods

As we have already stated, we utilize the GPU for a fast and efficient computation of the complete voxel-pixel projections that are required in both reconstruction methods. We render the voxelspace in the camera views 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 interprets volumes, e.g. the voxelspace, as a pack of images (layers), which are rendered individually. To avoid aliasing effects we insert layers parallel to each coordinate axis as shown in Figure 8.

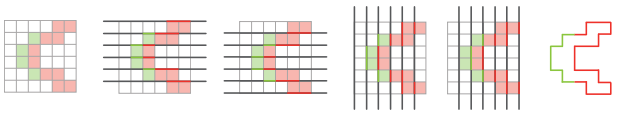

Figure 8: Exact texture mapping. From left to right: occupied voxels, rendering in directions: +y, -y, -x, +x, final image of the voxels

In comparison, the standard raycasting determines the visible voxel in each pixel by following the backprojected "viewing ray" until an occupied voxel is intersected (*ray marching*). The ray is sampled at discrete points with an equal step size. The optimization of Amanatides [1] avoids aliasing effects by computing analytical the intersections of the ray with the voxels. We compare the texture mapping with the standard raycasting, the Amanatides raycasting as well as a 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.

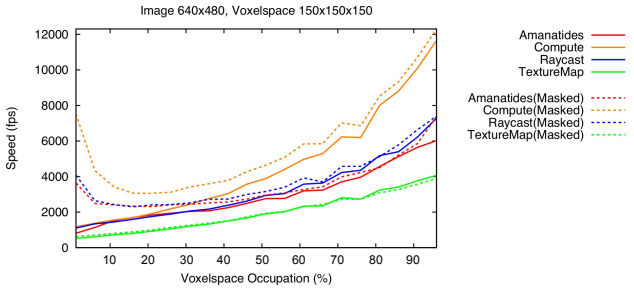

Figure 9: Computation times of different rendering methods for a varying occupation of the voxelspace.

Figure 9 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 and the optimization of Amanatides [1] produce similar results. The cost for calculating the intersection points of the viewing rays with the voxels (for reducing the amount 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.

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 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 amount 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 actual voxel and target voxel. An example measurement of the abso-

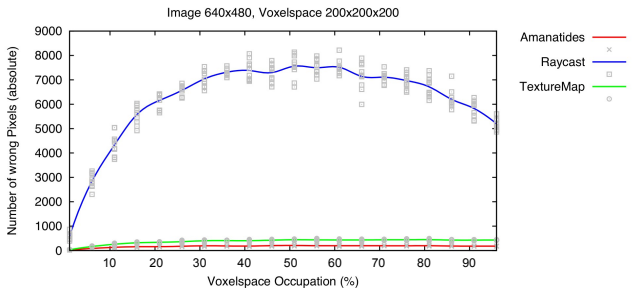

Figure 10: Absolute amount of wrong pixels compared to the naive rendering.

lute amount of wrong pixels for a voxelspace of size 200^3 is shown in Figure 10. The standard raycasting produces the worst results. The Amanatides rendering achieves the best results as well as its alternative implementation of the compute shader (graph is identical to Amanatides). With just 0.08 % wrong pixels the error can be neglected. Also, the average deviation of 2...3 voxels is significant lower for the Amanatides algorithms as for the raycasting (not shown). The texture mapping is a little worse than the Amanatides algorithms (not visible by eye), because it is completely conducted on the graphics board, which is optimized in speed and not in accuracy.

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. 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 with seven cameras of resolution 640×480. Two or three persons with colored overalls (for better background subtraction) are moving in the cell that contains several occluding static obstacles as shown in Figure 12. Furthermore a simulation of the same work cell with one person is used in order to have perfect ground truth data and undistorted images.

First, for computing the photo hull with a fix upper bound of 800 ms we examined the number of iterations by varying the voxelspace resolutions (100^3 up to 300^3) and the amount of cameras (4 up to 7). The results are shown in Figure 11. The presented values are average measurements of all image sequences. Surprisingly, the influence of the amount 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 amount of cameras. A large part of the computation time (up to 50% for 300^3 voxels) is required for resetting the voxelspace data structures. After each iteration the counter of each voxel that is used for the incremental computation of the LRT has te be reset. After each frame additionally the voxel data structures V_{free} and V_{occ} have to be zeroed. The OpenGL standard of version 4.2

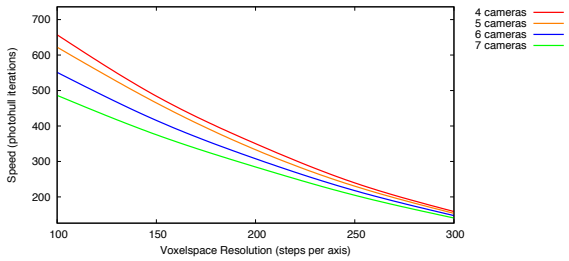

Figure 11: Iterations of the photo hull for varying amount of cameras and voxelspace resolutions.

does not provide a more efficient approach, but an outlook for improvement is given with OpenGL version 4.4 [15].

Another experiment aimed in the applicability of the approach to our real-world work cell of 40 m^3 with 7 cameras and a voxel resolution of 200^3. First, we determined the optimal threshold of $\tau = 7.5$ for the LRT consistency test given 800 ms processing time 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 the first 200 frames for each sequence.

Sequence	**VH**	**PH**		**Total**	
Persons		**Iter.**	**Total**	**Time**	**fps**
Sim 1	22 ms	8.2 ms	200 ms	222 ms	4,5
Real 2	29 ms	5.1 ms	200 ms	229 ms	4,3
Real 3	28 ms	8.1 ms	200 ms	228 ms	4,4

Table 3: Computation Times for the Visual Hull (VH) and the Photo Hull (PH).

Finally, we examined the quality of the photo hull. The *photo integrity* of Seitz and Dyer [16], which demands that the projection of the reconstruction into the cameras reproduces the original images is sufficiently fulfilled, as shown in Figure 12. 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. The other demand of *broad viewpoint coverage* [16] is sufficiently guaranteed in the center of the cell whereas the quality of the geometric reconstruction strongly suffers at the cell borders, which mainly are monitored by one camera only.

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 [3] is presented. Therefore an incremental calculation of the LRT consistency criterion is integrated. The termination of the algorithm is managed with the anytime concept of [4]. The input of the photo hull is a visual hull reconstruction. For the GPU we redesigned our visual hull algorithm of [6] that can handle conservatively known occluding obstacles. We analyzed different rendering techniques (texture mapping and raycasting

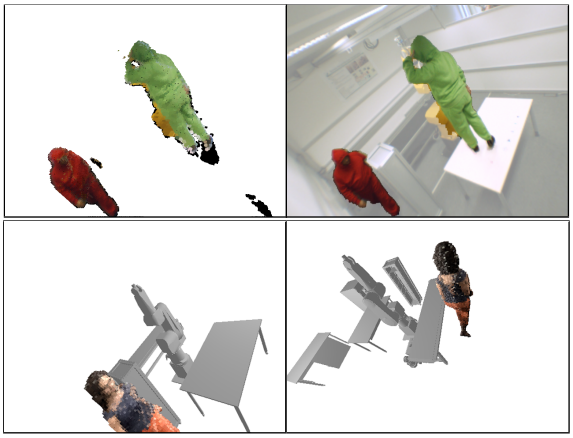

Figure 12: Results of the photo hull. Top: real-world work cell. Bottom: simulation.

approaches) for computing the required voxel-pixel projections. Our compute shader implementation of the Amanatides algorithm [1] 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. An improvement is expected with OpenGL 4.4. We figured out that 200 ms as upper bound for computing the photo hull (about 50 iterations) is sufficient for our scenario. An overall frame rate of 4,4 fps 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.

6. ACKNOWLEDGMENTS

This work has partly been supported by the *Deutsche Forschungsgemeinschaft (DFG)* under grant agreement He2696/11 SIMERO.

7. REFERENCES

[1] J. Amanatides and A. Woo. A fast voxel traversal algorithm for ray tracing. In *Proceedings of EUROGRAPHICS*, volume 87, pages 3–10, 1987.

[2] O. Batchelor, R. Mukundan, and R. Green. Ray casting for incremental voxel colouring. In *New Zealand: International Conference on Image and Vision Computing - IVCNZ05*, pages 206–211, 2005.

[3] W. B. Culbertson, T. Malzbender, and G. Slabaugh. Generalized voxel coloring. In *Vision Algorithms: Theory and Practice*, volume 1883, pages 100–115. 1999.

[4] T. Dean and M. Boddy. An analysis of time-dependent planning. In *Proceedings of the seventh national conference on artificial intelligence*, pages 49–54, 1988.

[5] T. Finch. Incremental calculation of weighted mean and variance. *University of Cambridge*, 2009.

[6] 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*, pages 784–795. Springer Verlag, Berlin, 2009.

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

[8] A. Ladikos, S. Benhimane, and N. Navab. Efficient visual hull computation for real-time 3D reconstruction using CUDA. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08.*, pages 1–8, 2008.

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

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

[11] 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.

[12] A. Prock and C. Dyer. Towards real-time voxel coloring. In *Proceedings of the DARPA Image Understanding Workshop*, pages 315–321, 1998.

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

[14] 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*, pages 372–381, 2009.

[15] M. Segal and K. Akeley. OpenGL 4.4 core profile specification, July 2013.

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

[17] 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*, pages 81–101, 2001.

[18] 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.

[19] 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.

[20] M. Zwicker. Erweiterung der Photohülle zur schnellen Onlinerekonstruktion auf moderner Grafikhardware. Master thesis, University of Bayreuth, Bayreuth, 2014.