Efficient Visual Hull Computation for Real-Time 3D Reconstruction using CUDA

Alexander Ladikos Selim Benhimane Nassir Navab

Department of Computer Science, Technische Universität München
Boltzmannstr. 3 , 85748 Garching, Germany
ladikos@in.tum.de, benhiman@in.tum.de, navab@in.tum.de

Abstract

In this paper we present two efficient GPU-based visual hull computation algorithms. We compare them in terms of performance using image sets of varying size and different voxel resolutions. In addition, we present a real-time 3D reconstruction system which uses the proposed GPU-based reconstruction method to achieve real-time performance (30 fps) using 16 cameras and 4 PCs.

1. Introduction

The visual hull [16], defined as the shape which is maximally consistent with the silhouette projections of an object, lies at the heart of almost all 3D reconstruction systems aimed at real-time performance. Other 3D reconstruction methods (see [25] for a recent overview) which are targeted at producing high quality results are too computationally expensive, since they require costly optimizations over the shape of the object. The visual hull on the other hand is straightforward to compute from a set of calibrated silhouette images. The downside is that it does not recover concavities which are not seen in the silhouettes. However, this is sufficient for many application areas (for instance body pose estimation [7]) since the visual hull still captures the essence of most shapes. In addition, once the reconstruction is textured the difference to the true reconstruction is even less apparent.

The visual hull is computed by building the generalized cones created by the viewing rays emanating from the camera center and passing through the silhouette contours. Having computed the generalized cones for every silhouette image, they are intersected in 3D producing the visual hull. There are two different approaches for performing this computation, namely the polyhedral and the volumetric approach.

In the polyhedral approach geometric properties are used to compute a mesh representation of the visual hull [21, 9, 17]. In [9], for instance, the silhouette contours are first discretized into line segments. Then the silhouette vertices are backprojected to 3D viewing lines which are restricted to so called viewing edges lying on the visual hull by clipping the lines based on their projections in the other silhouette images. The initial viewing edges are then connected by considering the intersection of the backprojections of the silhouette line segments, so that a complete mesh is obtained. The accuracy of the reconstructed visual hull depends on the discretization of the silhouette contours. The computational cost increases with finer silhouette discretization and an increasing number of images. One shortcoming of this approach is that it is susceptible to errors in the silhouette images and calibration errors, since then lines might not intersect exactly. To achieve real-time performance a method for distributing the computations over multiple processors has been proposed [10].

In the volumetric approach the visual hull is computed using a voxel representation [24, 29]. The reconstruction volume is discretized into voxels and each voxel is projected into the silhouette images. If the projection of a voxel lies inside a silhouette in all images it is marked as occupied, otherwise it is marked as empty. This method is very robust, since it does not rely on the silhouette contours but on the occupied/non-occupied image regions. Depending on the application it might also be more useful to obtain a voxel representation of the visual hull, for instance to compute the volume of the intersection between an object and a region of space. In addition, the accuracy of the reconstruction can be easily limited to the desired accuracy, which is much harder in the case of the geometric approach, since the discretization is done in image space and not in the volume. To speed this approach up octree representations of space are used. In addition, the computations can be distributed over multiple processors [27].

In [6] a method is proposed which combines both the polyhedral and the volumetric approach. There are also other approaches [22, 18, 19, 20] which do not compute the visual hull explicitly, but only render images of the visual

hull from a given viewpoint.

In the last decade many systems for 3D reconstruction mostly using the visual hull have been proposed. One early system which was using stereo instead of the visual hull was the CMU dome [23]. However, the computations were performed offline. Cheung *et al*. [7] present a voxel-based system which achieves a frame rate of 16 fps using 5 cameras at a voxel resolution of 64^3. Borovikov *et al*. [4, 5] propose a system using a voxel-based reconstruction method. They report frame rates of about 10 fps using 14 cameras at a volume resolution of 64^3. Arita *et al*. [3] present a system with a resolution of 100^3 and 14 fps. Wu *et al*. [30, 31] propose a system which is using a plane intersection test for the reconstruction. They report values of 12 fps with 9 cameras.

More recently there have been several systems which are capable of running at frame rates of 30 fps. Franco *et al*. [10] parallelize the polyhedral approach over a cluster of 8 dual Xeon 2.66 GHz PCs to achieve frame rates of 30 fps using four cameras. Allard *et al*. [1, 2] also use the polyhedral reconstruction algorithm in a system with 11 dual-Xeon 2.6 GHz PCs. They report frame rates of 30 fps using 6 cameras. Soares *et al*. [27] compute the visual hull of an 8 image data set at octree level 8 in 16.82 ms using a PC with 8 AMD Opteron dual core 2.2 GHz CPUs. Hasenfratz *et al*. [13, 14] also report frame rates of 30 fps with four cameras using a voxel-based method (resolution 64^3) implemented on the GPU of a SGI Onyx IR-3000 with 8 R12000 processors.

One thing all these systems have in common is that they are using very powerful and expensive hardware to achieve real-time frame rates. We on the other hand would like to achieve equivalent results using more readily available hardware. Specifically we are using only four PCs with 2.6 GHz Quad-Core CPUs and Geforce 8800 GTX graphics adapters to obtain 30 fps using 16 cameras. We achieve this by performing the visual hull computation on the GPU. Previous GPU-based methods make use of hardware texturing. Hasenfratz *et al*. [13, 14] use the GPU to map the silhouette images as textures onto slices through the reconstruction volume and intersect them using image blending. Hornung *et al*. [15] suggest to project every voxel into the images using a fragment shader and texture mip-mapping. We propose a method making use of CUDA [8] to perform the reconstruction by using kernels which compute the projection of every voxel and accumulate the occupancy information. We present two implementations of this approach and compare them in terms of performance. We also show results obtained with our real-time 3D reconstruction system which uses our GPU-based visual hull computation algorithm.

2. GPU-based Visual Hull Computation

The voxel-based visual hull computation is a perfect example of a highly parallel algorithm. Therefore, it is well suited to be implemented on the GPU. However, there are still different design choices for performing the computations. We evaluated two different approaches. In the first approach we precompute the bounding boxes of the voxel projections in each image and store them in a lookup table which is used during the visual hull computation. In the second approach we downsample the images so that a voxel approximately projects into one pixel. This allows us to only project the voxel center point to obtain the corresponding pixel in the image.

The first approach is most useful when the camera configuration is static which is the case for a 3D reconstruction system. If on the other hand one just wants to compute the visual hull once for a given camera configuration, the overhead in terms of memory and time that the precomputation requires, make this approach less attractive. The second approach does not perform any precomputations and is therefore suited for both cases.

We implemented both an octree version and a non-octree version of each approach. Using an octree representation of space speeds up the computations, but limits the size of the reconstruction volume to cubes with power of 2 side lengths. The non-octree version also allows to compute the visual hull on non-cubic volumes but requires a longer computation time.

2.1. Precomputation-based Algorithm

In this algorithm, called GPU1, we first compute the bounding boxes of the voxel projections in each image and store them in a lookup table. This is done in an offline step. The memory required by the lookup table is proportional to the number of images and the voxel resolution used. For the octree version we compute a lookup table for every octree level. The lookup table is copied onto the GPU during an initialization step.

For every new set of silhouette images, we first compute the corresponding integral images. This allows us to efficiently evaluate the number of occupied pixels given a bounding box in the image by using only four memory accesses instead of looking at every pixel in the bounding box. The time required to compute the integral images is significantly shorter than the overhead of looking at every pixel in the bounding box. All the before mentioned steps are performed on the CPU. To speed them up, the computations are also parallelized.

The integral images are then copied onto the GPU using page-locked memory to increase the transfer rates. In the non-octree version, a kernel is executed for each voxel. The kernel accesses the lookup table to find the corner points of

the voxel bounding box. The corner coordinates are used as an index into the integral images to compute the number of occupied pixels. The sum of occupied pixels and the sum of total pixels is accumulated over all images and their quotient is assigned as the occupancy to the voxel. This gives smoother results than only assigning 1 and 0 for the voxel occupancy. If a bounding box is found to be empty in one image the remaining images are not checked and the voxel is assigned a value of 0.

In the octree version of the algorithm (GPU1_OT) a similar procedure is executed for each octree level. We maintain a list of active octree cells (we talk of cells instead of voxels since an octree cell consists of multiple voxels except at the highest octree level) containing the ids of the cells which have to be checked at the current octree level. We start at level 4 because the overhead of using the octree approach is higher than computing the result directly when starting at a lower level. We therefore initialize the active cell list with the id of every cell at level 4 (4096 cells). We only start as many kernels as there are active octree cells and use the id of the kernel as an index into the active octree cell list to get the cell id. The computation of the occupancy is performed the same way as in the non-octree version, except that a different lookup table is used for every level. Depending on the result of the occupancy test, we set an entry in a cell list. If the cell is either totally occupied or totally empty, we set the entry to 0, indicating that we are not considering the cell at the next level and then fill in the corresponding voxels in the volume. In case of partial occupancy we mark the eight sub-cells corresponding to this cell on the next octree level in the cell list. After the kernel has finished executing we use the CUDPP library [26] and a small additional kernel to compact the cell list to only contain the ids of the active cells, which are then used as the input to the kernel on the next level. The resulting voxel volume is copied off the GPU using page-locked memory.

2.2. Direct Algorithm

The direct algorithm (GPU2) does not perform any precomputations. Instead we downsample the silhouette images so that every voxel approximately projects into a single pixel. This allows us to only compute the projection of the voxel center point and to perform a single memory access in the silhouette images. In the non-octree version we downsample the image to match the voxel resolution using a Gaussian smoothing followed by a downscaling. The kernel is then executed for every voxel. It derives the 3D position of its corresponding voxel from its id and projects the voxel center point into the image. The occupancy value of the voxel is computed as the minimum over the values at all voxel projections. If a value is 0 in one image the remaining images are not checked anymore and the voxel occupancy is set to 0.

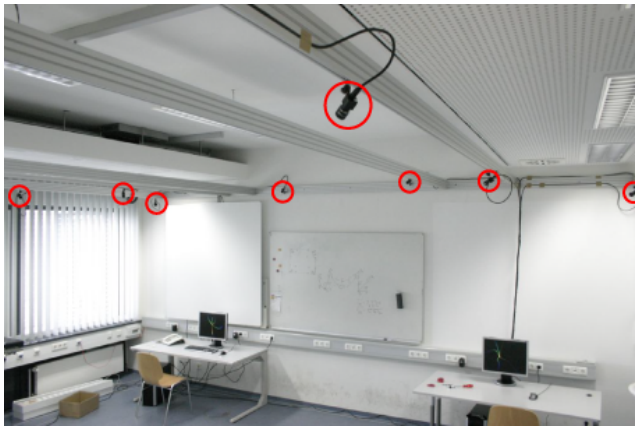

Figure 1. Lab setup for our real-time 3D reconstruction system. The cameras are marked with red circles (not all cameras are seen in this image).

In the octree version (GPU2_OT) the silhouette images are downsampled in a Gaussian pyramid, so that one image is obtained for every octree level. The kernel is then executed for every level using only the active octree cells which are determined in the same way as in the precomputation-based algorithm. The resulting voxel volume is copied off the GPU using page-locked memory.

3. Real-time 3D Reconstruction System

3.1. System Architecture

3.1.1 Hardware

Our system consists of 4 PCs used for the reconstruction, 1 PC used for visualization and 16 cameras mounted on the ceiling (see figure 1). The cameras have an IEEE 1394b interface and provide color images at a resolution of 1024x768 and a frame rate of 30 Hz. To cover a big working volume we use wide angle lenses with a focal length of 5 mm. The cameras are externally triggered to achieve synchronous image acquisition. Groups of four cameras are connected to one PC using two IEEE 1394b adapter cards. There are four PCs (slaves) dedicated to capturing the images and computing the visual hull and one PC (master) dedicated to visualizing the result and controlling the acquisition parameters. The four PCs used for image acquisition and reconstruction are equipped with an Intel 2.6 GHz Quad-Core CPU (Q6700), 2 GB of main memory and a NVIDIA 8800 GTX graphics board with 768 MB of memory. The master PC uses an Intel 3.0 GHz Dual-Core CPU (E6850), 2 GB of main memory and a NVIDIA 8800 GTS graphics board with 640 MB of memory. The PCs are connected through a Gigabit Ethernet network.

3.1.2 Software

To achieve real-time performance the reconstruction process (running on the slave PCs) is implemented as a four stage pipeline consisting of image acquisition, silhouette extraction, visual hull computation and volume encoding and transmission. Each pipeline step is realized as a thread and will be described in detail in the following sections. On the master PC the processing is also distributed into several steps. There is a separate thread for handling network communication, compositing the partial reconstructions and visualizing the result. In addition, an application specific stage can be introduced to perform additional postprocessing on the volume. Figure 2 gives an overview of the processing steps in the system.

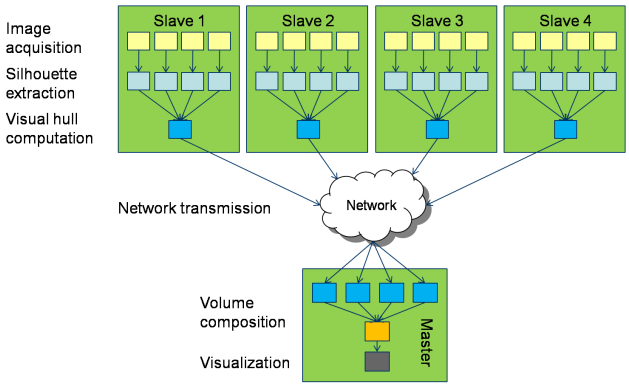

Figure 2. Each workstation acquires images from its locally attached cameras and computes the visual hull using the GPU. The resulting voxel volumes are sent to the master PC which combines them and visualizes the result.

3.2. Calibration

In order to perform the reconstruction the cameras have to be calibrated. The calibration is performed using the multi-camera calibration method proposed by Svoboda *et al*. [28] because it is easy to use and yields good calibration results. The method relies on point correspondences between the cameras created by means of a point light source such as an LED. First, the lighting in the room is dimmed, so that it becomes easier to extract the point created by the LED in the camera images. By moving the light source through the reconstruction volume a large number of correspondences is created which is then used in a factorization-based algorithm to determine the camera intrinsic and extrinsic parameters. This requires synchronized cameras to make certain that the point seen in each image is created by the same physical point. The method is robust to occlusions of the points in some cameras. The computed camera coordinate system is registered to the room coordinate system by using a calibration target at a known position in the room.

3.3. Reconstruction

3.3.1 Silhouette Extraction

The silhouettes are computed using a robust background subtraction algorithm [11] working on color images. Before the system is used background images are acquired. During runtime the images are first corrected for illumination changes using a color mapping table which is built using the color distributions of corresponding non-foreground regions in the current image and the background image. After applying this mapping to the background image, a thresholding is applied to extract the foreground pixels in the current image. Small holes in the segmentation are filled using morphological operations.

3.3.2 Handling Static Occluders

One problem which has to be addressed during silhouette extraction is the presence of static occluders in the scene. Static occluders are objects inside the working volume which cannot be removed, such as tables mounted to the floor. Hence static occluders are also present in the background images. The assumption during background subtraction, however, is that all foreground objects are located in front of the background. This is not the case in the presence of an occluder because a foreground object could move behind the occluder and effectively disappear from the silhouette image. This will result in the partial or complete removal of the object from the reconstruction. To overcome this problem, the areas in the silhouette images corresponding to the static occluder have to be disregarded during the visual hull computation. We achieve this goal by building a 3D representation of the object and projecting it into the cameras or by manually segmenting the object in the reference images. This gives us a mask for every camera in which the static occluder is marked as foreground. This mask is then added (logical OR) to the silhouette images computed during runtime. A similar approach was suggested in [12].

This method is also useful when it is not convenient to remove a static occluder from the working volume every time background images are taken. For instance it could be used to reconstruct a table which has been removed from the volume when the background images were taken. Now, every time something in the background changes it is possible to just take new background images including the table and use the static occluder mask to integrate the table into the scene without having to move it out of the scene first.

3.3.3 Visual Hull Computation

Using the silhouette images the object shape is reconstructed using the GPU-based visual hull algorithm (GPU2_OT) described in section 2. In order to increase the

working volume we also reconstruct regions which are only seen by at least four cameras instead of only using the overlapping region of all 16 cameras. This allows us to avoid the use of extreme wide angle lenses for covering a big area, which also results in a higher spatial resolution of the camera images. To reconstruct the non-overlapping regions, one has to consider the handling of voxels which project outside of the image in other cameras. The traditional approach is to just mark these voxel as empty. Instead we do not consider the contribution of the images in which the voxels are not visible, thereby also reconstructing regions only seen by a few cameras. To avoid the introduction of artifacts due to a too low number of cameras, we only use regions which are seen by at least four cameras. This is implicitly accomplished in our system by performing an unconstrained reconstruction on the slave PCs which also reconstructs the regions seen by only one camera. On the master PC the local reconstructions are combined using a logical AND operator, which will remove any regions which have not been observed by at least one camera at each of the four slave PCs.

3.4. Visualization

For visualization the voxel representation is converted to a mesh representation using a CUDA-based marching cubes implementation. A CPU-based implementation was not able to generate meshes at the desired frame rate.

4. Results

4.1. Evaluation of the Visual Hull Algorithms

To evaluate the performance of the visual hull computation algorithms, we tested them on three data sets at different voxel resolutions.

The first data set is used to assess the performance for a scene typically encountered in a real-time 3D reconstruction system. This data set was captured with our system and shows a person walking through the room. This is a very typical scenario for a real-time 3D reconstruction system. We first computed the visual hull using all 16 images (resolution 1024x768). The runtimes are shown in figure 3. It can be seen that even at a resolution of 128^3 we still achieve real-time performance using the direct algorithm (GPU2_OT) which consistently provides the best reconstruction times at all resolutions. Also note that the precomputation-based algorithms (GPU1, GPU1_OT) fail to run at a resolution of 256^3 because the lookup table size exceeds the available storage on the GPU (768MB).

To test the scalability of the algorithms, we ran them on the same data set using different numbers of images. Figure 4 shows the resulting runtimes, plotted over the number of input images. The first plot shows the results of the non-octree algorithms. It can be seen that the precomputation-based algorithm (GPU1) has a complexity linear in the number of images, while the direct algorithm (GPU2) has a complexity which first increases steeply with the number of images and then only grows very slowly. The highly linear behavior of the precomputation-based method can be explained by the the computation of the integral images which is linear in the number of images. In the direct algorithm this step is not necessary, so that it only grows much slower (the down-sampling of the images is very fast). It should also be noted that the probability of finding a totally empty voxel projection early is increasing with a higher number of images, so that it is not unexpected to see only minor changes in the runtime as the number of images increases. An interesting observation is that the non-octree algorithms can provide real-time results. In particular the GPU1 method can be used with four images and a resolution of 128^3 to obtain reconstruction times of about 30 ms. Since no octrees are used, this means that it is also possible to reconstruct non-cubic volumes with the same performance as long as they have a similar number of voxels. In this case it is advisable to use the precomputation-based algorithm (GPU1) since it is faster at higher resolutions than the direct algorithm when using a reasonable number of images.

The second plot shows the performance of the octree-based algorithms. We can observe the same behavior as for the non-octree algorithms, albeit at a higher performance level. The direct algorithm (GPU2_OT) exhibits a slightly higher growth rate using the octree than without using it.

Method	64^3	128^3	256^3
GPU1	46.40 ms	64.44 ms	x
GPU2	18.60 ms	113.88 ms	870.06 ms
GPU1_OT	45.46 ms	51.94 ms	x
GPU2_OT	**15.24 ms**	**25.91 ms**	**73.53 ms**

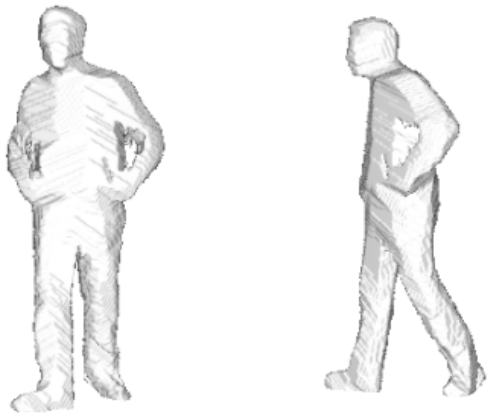

Figure 3. Runtimes on the person dataset consisting of 16 images (1024x768). The precomputation-based algorithms cannot run at level 256^3 due to the size of the lookup table.

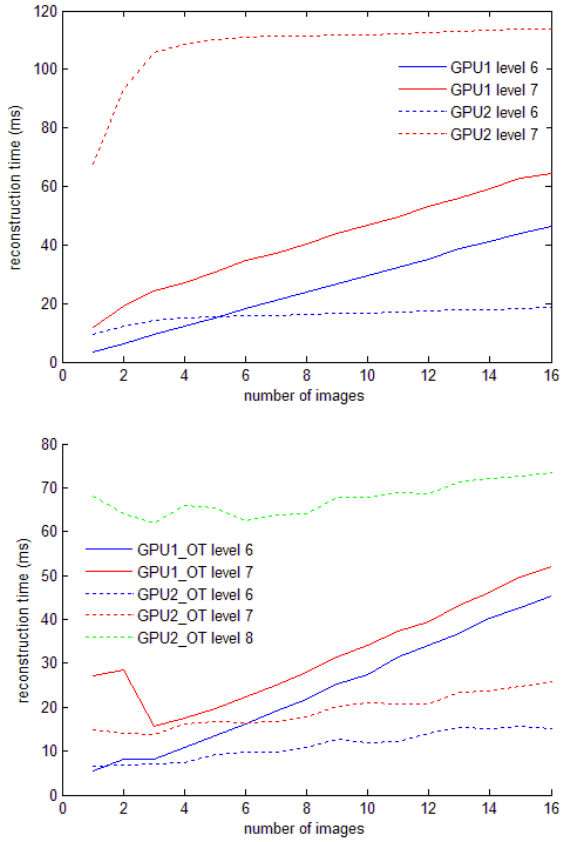

Figure 4. Runtimes on the person data set over number of images used. The image resolution used was 1024x768.

This is due to the computation of the image pyramid in the octree version. The fact that for a low number of images the runtimes of the algorithms are sometimes decreasing when using more images can be explained by the fact, that by using more images the visual hull will be more constrained and hence smaller. This allows the octree methods to stop checking some octree cells at an early octree level, which increases the performance. After a certain number of images this effect disappears, because adding new images only changes the visual hull slightly. When using the octree version of the algorithms the direct method should be used (GPU2_OT), because it shows a consistently better performance than the precomputation-based method.

The second and third data set are taken from the Middlebury multi-view evaluation [25]. We used them to test, how the algorithms perform with large numbers of images and complex scenes. The results obtained with the dinoRing data set consisting of 48 images (resolution 640x480) are shown in figure 5. The memory requirements of the precomputation-based algorithms were so high that they could only run at the lowest resolution. The direct octree algorithm on the other hand computes the visual hull quite

Method	64^3	128^3	256^3
GPU1	95.26 ms	x	x
GPU2	52.76 ms	417.79 ms	3033.89 ms
GPU1_OT	63.95 ms	x	x
GPU2_OT	**39.94 ms**	**99.89 ms**	**296.71 ms**

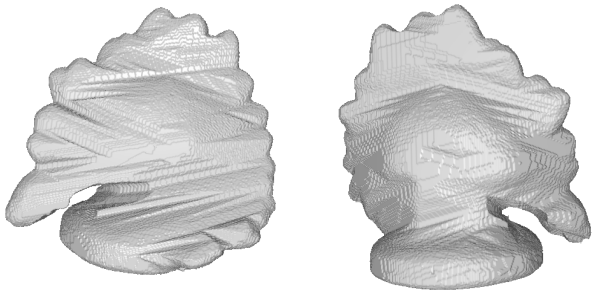

Figure 5. Runtimes on the dinoRing dataset consisting of 48 images (640x480). The precomputation-based algorithms can only run at 64^3 due to the size of the lookup table.

fast, at a resolution of 64^3 even at 25 fps. The results obtained with the templeRing data set consisting of 47 images (resolution 640x480) are shown in figure 6. This data set was used to test how the performance changes with increasing scene complexity. While the non-octree methods show a similar performance as on the dinoRing data set, the octree methods have an increased runtime. This is expected, because the probability of being able to terminate the computations on a low octree level is lower, when the scene is more complex. It is clear from these experiments that for large data sets and high resolutions the direct octree method (GPU2_OT) provides the best results.

One also has to consider the precomputation times when using the precomputation-based algorithms. They usually lie in the range of a few seconds for moderately sized data sets. This means that if one wants to only compute the visual hull once with a given camera configuration, it is better to use the direct method. In a real-time system where the camera configuration does not change this is not an issue.

4.2. Results with the Real-Time System

To test the quality of the reconstruction achieved by our real-time 3D reconstruction system and the static occluder handling, we set up an application, in which we want to avoid collisions between objects in the working volume and a device. This application is for instance of interest when working with fast-moving robots, where it is dangerous to enter the working area. The device used is a medical X-ray device. We first took the device out of the working area and took background images. Then we reconstructed it using our system and used the resulting model as a static occluder. We then restarted the system taking new background images containing the static occluder. We defined the area extending 20 cm outside of the bounding box of the device as its

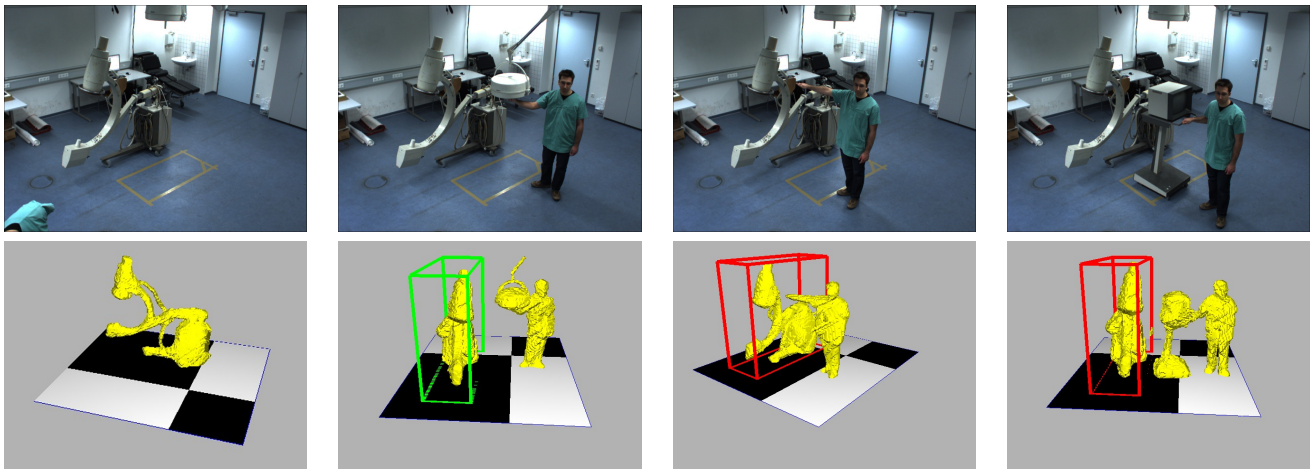

Figure 7. Results of the collision avoidance application. Each column shows one of the input images in the upper row and the reconstruction in the bottom row. The first column shows the reconstructed device. The second column shows the device in a safe state (green bounding box), while the third and fourth column contain an object in the working volume (red bounding box).

Method	64^3	128^3	256^3
GPU1	97.42 ms	x	x
GPU2	**51.61 ms**	372.95 ms	3022.10 ms
GPU1_OT	62.83 ms	x	x
GPU2_OT	56.48 ms	**170.91 ms**	**516.80 ms**

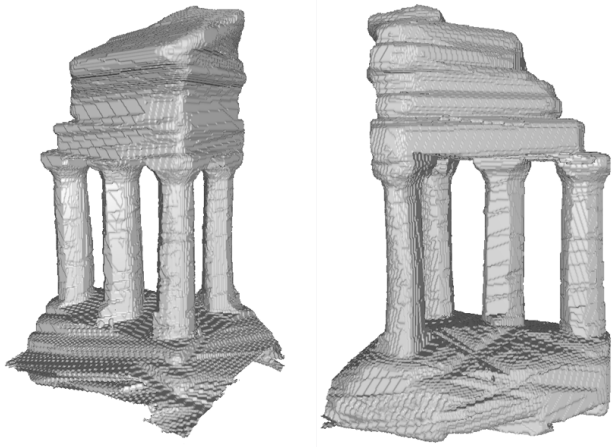

Figure 6. Runtimes on the templeRing dataset consisting of 47 images (640x480). The precomputation based algorithms can only run at 64^3 due to the size of the lookup table.

working volume. We then proceeded to place different objects in the vicinity of the device. When a intersection with the device working volume was detected, the bounding box was painted red. Figure 7 shows some reconstruction results and some configurations with objects inside and outside the working volume.

To speed up the segmentation the input images were downsampled to a resolution of 512x384. The volume resolution used was 128^3. The runtimes of the different steps of the system as described in section 3 are as follows: segmentation 15 ms, reconstruction 7-15 ms and volume encoding and transmission 10 ms. The volume is encoded on the CPU using run-length encoding. The visualization runs easily at 30 fps. The total latency of the system is approximately 120 ms. We compared the performance of both octree algorithms and found, that at the resolution and the volume size used, both had a worst-case performance of about 15 ms depending on the scene complexity with the GPU2_OT method being faster on average.

5. Conclusion

We presented two GPU-based approaches for efficient visual hull computation. One approach uses a precomputed lookup table, while the other directly computes the voxel projections. Both methods were implemented in an octree and a non-octree version on the GPU using CUDA. We compared the performance of both methods on different data sets. Our results indicate that for small non-cubic resolutions the precomputation-based non-octree algorithm (GPU1) should be used, while for high resolution volumes and large numbers of images the direct octree algorithm (GPU2_OT) performs better. In addition, we presented a real-time 3D reconstruction system using the proposed octree visual hull computation algorithm (GPU2_OT). The system uses 16 cameras with only four PCs for the reconstruction and runs at a frame rate of 30 fps at a resolution of 128^3. We also showed results obtained with this real-time 3D reconstruction system in a collision avoidance application.

Acknowledgement: This research is funded by Siemens Medical Solutions, Forchheim, Germany. We would also like to thank Dr. Thomas Redel and Dr. Klaus Klingenbeck-Regn for their support.

References

[1] J. Allard, J.-S. Franco, C. Ménier, E. Boyer, and B. Raffin. The grimage platform: A mixed reality environment for interactions. In *IEEE International Conference on Computer Vision Systems*, 2006.

[2] J. Allard, C. Menier, B. Raffin, E. Boyer, and F. Faure. Grimage: Markerless 3d interactions. In *SIGGRAPH - Emerging Technologies*, 2007.

[3] D. Arita and R. Taniguchi. Rpv-ii: A stream-based real-time parallel vision system and its application to real-time volume reconstruction. In *IEEE International Conference on Computer Vision Systems*, 2001.

[4] E. Borovikov and L. Davis. A distributed system for real-time volume reconstruction. In *IEEE International Workshop on Computer Architectures for Machine Perception*, 2000.

[5] E. Borovikov, A. Sussman, and L. Davis. A high performance multi-perspective vision studio. In *ACM International Conference on Supercomputing*, 2003.

[6] E. Boyer and J. S. Franco. A hybrid approach for computing visual hulls of complex objects. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2003.

[7] G. Cheung, T. Kanade, J. Y. Bouguet, and M. Holler. A real-time system for robust 3d voxel reconstruction of human motions. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2000.

[8] CUDA. http://www.nvidia.com/cuda.

[9] J. S. Franco and E. Boyer. Exact polyhedral visual hulls. In *British Machine Vision Conference*, 2003.

[10] J. S. Franco, C. Menier, E. Boyer, and B. Raffin. A distributed approach to real time 3d modeling. In *Proceedings of the 2004 Conference on Computer Vision and Pattern Recognition Workshops*, 2004.

[11] S. Fukui, Y. Iwahori, H. Itoh, H. Kawanaka, and R. Woodham. Robust background subtraction for quick illumination changes. In *PSIVT*, 2006.

[12] L. Guan, S. Sinha, J.-S. Franco, and M. Pollefeys. Visual hull construction in the presence of partial occlusion. In *3DPVT '06: Proceedings of the Third International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT'06)*, 2006.

[13] J.-M. Hasenfratz, M. Lapierre, J.-D. Gascuel, and E. Boyer. Real-time capture, reconstruction and insertion into virtual world of human actors. In *Vision, Video and Graphics*, pages 49–56, 2003.

[14] J.-M. Hasenfratz, M. Lapierre, and F. Sillion. A real-time system for full body interaction with virtual worlds. *Eurographics Symposium on Virtual Environments*, pages 147–156, 2004.

[15] A. Hornung and L. Kobbelt. Robust and efficient photo-consistency estimation for volumetric 3d reconstruction. In *European Conference on Computer Vision*, 2006.

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

[17] S. Lazebnik, Y. Furukawa, and J. Ponce. Projective visual hulls. *International Journal of Computer Vision*, 74(2):137–165, 2007.

[18] M. Li, M. Magnor, and H. Seidel. Hardware accelerated visual hull reconstruction and rendering. In *Proc. of Graphics Interface*, 2003.

[19] M. Li, M. Magnor, and H. Seidel. Improved hardware-accelerated visual hull rendering. In *Vision, Modeling, and Visualization*, 2003.

[20] M. Li, M. Magnor, and H.-P. Seidel. A hybrid hardware-accelerated algorithm for high quality rendering of visual hulls. In *GI '04: Proceedings of Graphics Interface 2004*, pages 41–48, 2004.

[21] W. Matsuik, C. Buehler, and L. McMillan. Polyhedral visual hulls for real-time rendering. In *Eurographics Workshop on Rendering*, 2001.

[22] W. Matsuik, C. Buehler, R. Raskar, S. Gortler, and L. McMillan. Image-based visual hulls. In *SIGGRAPH*, 2000.

[23] P. J. Narayanan, P. Rander, and T. Kanade. Constructing virtual worlds using dense stereo. In *International Conference on Computer Vision*, pages 3 – 10, 1998.

[24] M. Potmesil. Generating octree models of 3d objects from their silhouettes in a sequence of images. *Comput. Vision Graph. Image Process.*, 40(1):1–29, 1987.

[25] S. Seitz, B. Curles, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2006.

[26] S. Sengupta, M. Harris, Y. Zhang, and J. D. Owens. Scan primitives for gpu computing. In *Graphics Hardware 2007*, pages 97–106. ACM, 2007.

[27] L. Soares, C. Ménier, B. Raffin, and J.-L. Roch. Parallel adaptive octree carving for real-time 3d modeling. In *IEEE Virtual Reality*, 2007.

[28] T. Svoboda, D. Martinec, and T. Pajdla. A convenient multi-camera self-calibration for virtual environments. *Presence: Teleoperators and Virtual Environments*, 14(4):407–422, 2005.

[29] R. Szeliski. Rapid octree construction from image sequences. *CVGIP: Image Understanding*, 58(1):23–32, 1993.

[30] X. Wu and T. Matsuyama. Real-time active 3d shape reconstruction for 3d video. *Proceedings of the 3rd International Symposium on Image and Signal Processing and Analysis*, 1:186–191 Vol.1, 2003.

[31] X. Wu, O. Takizawa, and T. Matsuyama. Parallel pipeline volume intersection for real-time 3d shape reconstruction on a pc cluster. In *IEEE International Conference on Computer Vision Systems*, 2006.