PARALLEL HIGH RESOLUTION REAL-TIME VISUAL HULL ON GPU

Wolfgang Waizenegger, Ingo Feldmann, Peter Eisert, Peter Kauff

Fraunhofer Institute for Telecommunications, Heinrich-Hertz-Institute, Berlin, Germany
Email: {Wolfgang.Waizenegger, feldmann, eisert, kauff}@hhi.fraunhofer.de

ABSTRACT

In this paper we present an efficient high resolution Image Based Visual Hull (IBVH) algorithm that entirely runs in real-time on a single consumer graphics card. The target application is a real-time 3D video conferencing system. One major contribution of this paper is a novel caching strategy for the reduction of line segment intersection tests. In contrast to existing approaches, it additionally allows us to pre-select a close estimation of the set of relevant pixels in the desired view. Based on this, we obtain a significant computational speedup, especially for high resolutions. Further, we propose an efficient way to use IBVH for the generation of voxel equivalent 3D models. We compare our techniques in terms of resolution and runtime with state of the art real-time multi GPU voxel based approaches. Our experiments show that we achieve a speed-up by a factor of five and more for high resolutions.

Index Terms— Image based Visual Hull, CUDA, Real-Time, 3D Reconstruction

1. INTRODUCTION

The target application of this paper are immersive 3D videoconferencing systems. In difference to traditional high-end commercial 2D solutions, such as Cisco's TelePresence, Polycom's TPX, and HP's HALO, in a 3D tele-conferencing system the conferees are cut out of the real scene and virtually placed into a 3D shared table environment together with the remote conferees. From a research point of view the major challenge of such systems is the generation of real-time high quality and high resolution depth maps or 3D models of the captured scene and the conferees [1]. Recent approaches usually use highly optimized disparity estimation techniques based on stereo block matching [2]. Nevertheless, in the past years an increasing number of approaches can be found which solve the problem of real-time 3D scene reconstruction with volume based Visual Hull (VH) techniques. The use of VH methods is a common approach if a fast 3D reconstruction is desired. Lacking the ability of recovering concavities the VH redress this issue with a rather low computational complexity. Other 3D reconstruction methods focused on high quality and accurateness suffer from a huge processing time in the range of minutes or even hours for each frame because a global optimization or costly iterative filter procedure has to be done [3, 4, 5].

There are two main approaches for VH computation. In the *polyhedral approach* a mesh representation of the VH is computed by exploiting geometric properties, e.g. [6]. The *volumetric approach* achieves VH generation by using a voxel representation of a volume and discarding the voxels whose projections are not part of all silhouettes. To achieve high performance with voxel based methods some authors use expensive multiprocessor machines or distributed computing solutions [7, 8]. In order to avoid expensive hardware

Fig. 1. Experimental real-time 3D videoconferencing system setup based on Image Based Visual Hull.

Ladikos et al. [9] proposed a distributed system with 4 conventional PCs each of which carrying a Geforce 8800 GTX graphics card for VH computation using CUDA [10]. It achieves 30fps with 16 cameras and a volumetric resolution of 128^3.

The main contribution of this paper is a novel fast real-time Image-Based Visual Hull approach for an experimental 3D video conferencing system as illustrated in fig. 1. In difference to Ladikos et al. our algorithms run on a single Geforce GTX 280 graphics card in CUDA on a conventional PC. Compared to other existing techniques our algorithm offers a significant computational speedup. Our method implements a special variation of an Image Based Visual Hull that was initially designed for depth map rendering. We will show, that it has also a close relationship to voxel based methods and is capable to generate a *voxel equivalent* 3D model representation.

In the following, we will first give a brief overview concerning IBVH techniques. A general algorithmic description and parallelization specific issues are discussed afterwards. In section 4 we will outline the details of our novel caching strategy. Afterwards, we discuss the speedup gained through our novel pixel pre-selection technique during the computation of the required 3D intervals. Subsequently, we discuss how IBVH techniques can be used to generate voxel equivalent 3D models by exploiting a close relationship to voxel based Visual Hulls. Finally, the experimental section proofs the efficiency of our algorithm. We will discuss the results of our method on the Middlebury test dataset [3] and on our experimental 3D videoconferencing setup as well.

2. IMAGE BASED VISUAL HULLS

Image based Visual Hulls (IBVH) were initially proposed by Matusik et al. [11]. Instead of computing an explicit 3D model, *reference views* are used to render only the depth map for a *desired view*. Therefore, the generalized cones induced by the viewing rays emanating from each camera center and passing through the silhouette borders are regarded. The Visual Hull can be acquired by intersecting all generalized cones. Consequently, the depth map for the desired view can readily be assembled by intersecting the viewing rays of the desired view with the intersection of all generalized cones. Viewing ray intersection with the intersection of all generalized cones is equivalent to the intersection of the 3D intervals originated by the intersection of the viewing ray with each generalized cone. An important cornerstone for the efficiency of IBVH techniques is to take advantage of epipolar transfers in order to perform the intersection operations with the generalized cones in image space. Due to this and the possibility of utilizing a very effective cache data structure for minimizing the amount of intersection tests, IBVH is computationally efficient compared to voxel based approaches where the projections of large amounts of voxels to the image planes is required in order to check if they belong to the silhouettes. IBVH techniques are in general well suited for parallelization as the computation for every pixel in the desired view can be done independently.

3. GENERAL ALGORITHM

The proposed general algorithmic chain is illustrated in fig. 2. Note, that all modules are implemented on the GPU. The first algorithmic step is the computation of the polygonal representation of the silhouette borders. Therefore, for the extraction of the line segments of the silhouette images, we use marching squares, which is the 2D version of the well known marching cubes algorithm [12]. Based on this, the generation of our cache structure is performed. It is mainly used for the caching of line segments in order to reduce the intersection tests with epipolar lines originated by pixels of the desired image. Furthermore, the cache structure can be additionally used to pre-select relevant pixels of the desired image. Subsequent to the intersection computation in image space the results are lifted to 3D. This leads to a set of 3D intervals for each pixel of the desired image and each silhouette image. The intersection of all interval sets attached to a pixel of the desired image afterwards provides the intersections of the viewing rays with the intersection of all generalized cones. Finally, a depth map or alternatively a voxel equivalent representation is generated. All modules will be discussed in more detail in the following sections.

4. COMPUTATION OF THE LINE SEGMENT CACHE

In the following we present a novel technique for caching the line segments of the polygonal representations of the silhouette borders. This is done by exploiting an angular relationship between the epipoles and the line segments as described below. Preparative we describe how to represent line segments from silhouette images.

During line segment extraction, our marching squares implementation stores each line segment as a tuple $\boldsymbol{L} = (\boldsymbol{l}, \boldsymbol{\alpha})$ where $\boldsymbol{l}$ denominates a line in homogenous coordinates and $\boldsymbol{\alpha} = (\alpha_1, \alpha_2)$, $\alpha_1 < \alpha_2$ are angular values that restrict this line to an interval. In order to exploit the properties of epipolar geometry for the caching structure, $\boldsymbol{\alpha}$ is computed against the epipole of the line segment's silhouette image. For an illustration please refer to fig. 3. Clearly we cannot serve the case where the epipole resides at infinity with this representation, but this is only a theoretical issue and not relevant in almost all practical setups. Furthermore, the loss of generality is well refunded with a significant algorithmic speedup.

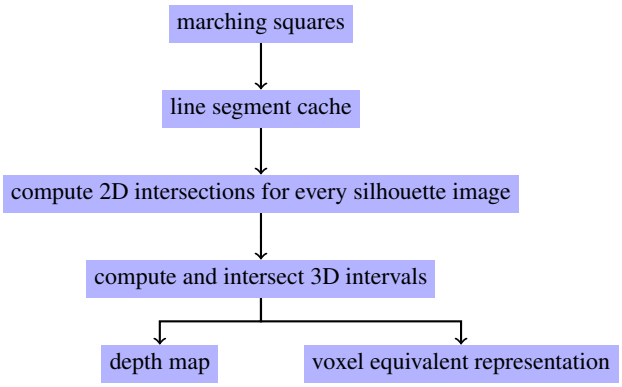

Fig. 2. Algorithm chain for our image based visual hull implementation on the GPU.

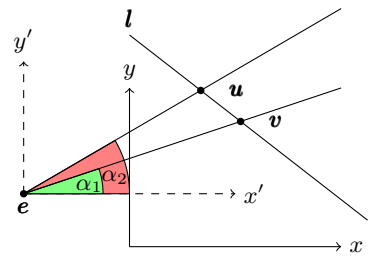

Fig. 3. Line representation with interval generating angular values. Marching Squares identified a line segment from $\boldsymbol{u}$ to $\boldsymbol{v}$. The epipole of the silhouette image is denoted with $\boldsymbol{e}$.

In the following we examine the generation of our angular line segment cache. Given the introduced line segment representation, we first envision how the cache works and how to prepare these representation for a correct and efficient caching strategy. For intersecting a viewing ray with a generalized cone in image space, we compute the epipolar line $\boldsymbol{l} \sim (l_1, l_2, l_3)$, which is the projection of this viewing ray on the silhouette image. Since the angular part of the line segment representation is computed against the epipole, the epipolar line's angle for intersection testing is consequently given through $\alpha_l = \arctan(-l_1/l_2)$. Therefore $\boldsymbol{l}$ intersects all line segments whose angular values fulfill $\alpha_1 \leq \alpha_l \leq \alpha_2$ or $\alpha_1 \leq \alpha_l - \text{sign}(\alpha_l) \cdot \pi \leq \alpha_2$. For this reason, we perform a mapping on each α tuple of the line segments to receive a compact cache structure with a maximal angular range of $[0, \ldots, \pi]$.

The cache assembly proceeds as follows. For each silhouette image our angular line segment cache provides N bins, where N should be selected proportionally to the number of line segments. By quantizing the maximal angular range of the polygonal silhouette representations we assign an angular value to each cache bin. Depending on its angular range each line segment is placed in one or more bins of the cache structure. For an illustration please refer to fig. 4.

In order to perform a fast parallel cache generation we tackle the whole cache structure computation on the GPU using a divide and

conquer strategy. Parallelization is done on image level and on line segment level as well. We compute the cache for each silhouette image in parallel and simultaneously split the amount of line segments into subsets. This leads to a parallel assembly of a separate cache for each subset. The recombination of these *subcaches* provides the desired result.

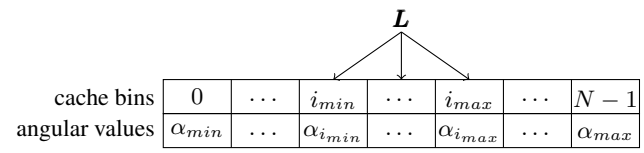

Fig. 4. The angular values assigned to the cache bins are quantized according to the maximal angular range of the silhouette mask α_{min} and α_{max}. The line segment $\boldsymbol{L}$ is assigned to every cache bin i, $i_{min} \leq i \leq i_{max}$ within its angular range.

5. COMPUTATION OF 3D INTERVALS

In this section, we illustrate how the 3D intervals are computed that emerge through the intersection of the viewing rays of the desired image with the intersection of all generalized cones. All calculations are entirely performed on the GPU and can be divided into three steps. These are the pre-selection of relevant pixels in the desired image by angular constraints, the intersection of each viewing ray with every generalized cone in image space, and finally, lifting the 2D intersections back to 3D and intersecting the resulting 3D intervals. Since the latter two steps are exhaustively covered by Matusik et al. [11, 13] we mainly discuss the pixel pre-selection.

Due to our novel cache data structure, we can utilize the angular range of the line segment cache for pixel pre-selection. After performing epipolar transfer for a pixel $\boldsymbol{x}_D$ of the desired image to a silhouette image I, we can compute the angular value $\alpha^I_{\boldsymbol{x}_D}$ of the resulting epipolar line. The angular value is mapped to the maximal cache range of $[0, \ldots, \pi]$ and $\boldsymbol{x}_D$ is only considered for further computations if $\alpha^I_{\min} \leq \alpha^I_{\boldsymbol{x}_D} \leq \alpha^I_{\max}$ holds for all silhouette images. This gives a close estimation to the final set of silhouette pixels in the desired image and accelerates computation significantly.

Subsequent to 2D line intersection and 3D interval intersection it is straight forward to render a depth map for the desired view. We only have to collect the 3D interval points with the smallest z-value of each interval set attached to a pixel of the desired image and place them to their positions in the depth map. Of course it is also possible to use the whole set of 3D intervals to plot a 3D point cloud which represents the visual hull, but this is discussed in more detail in the context of using cameras at infinity and voxel equivalent representations in the next section.

6. VOXEL EQUIVALENT REPRESENTATIONS

Beside the rendering of desired views or depth maps, the generation of 3D models is in the focus of visual hull computation. Both main approaches, namely the polyhedral and the volumetric approach deal with this issue. Our novel approach at this place is to use the IBVH method with a desired camera at infinity. In this way it is possible to create a *voxel equvialent* representation. For an illustration please refer to fig. 5.

In order to find the appropriate camera at infinity we compute the bounding volume for the scene viewable by the reference cameras. Based on the axis aligned bounding cuboid of this bounding volume and the desired volumetric resolution we determine the adequate camera at infinity $\boldsymbol{P}_{\inf}$. By using IBVH with $\boldsymbol{P}_{\inf}$ we receive 3D intervals with a constant discrete spacing in x- and y-direction and, up to floating point precision, exact values in z-direction. If desired, a simple way to transform this result into a common voxel representation is to round the z-values according to the voxel resolution and perform a flood fill operation on each interval. Due the loss of precision in z-direction and the additional time and memory consumption, there is no reason for switching to this representation for visualization. We rather visualize the 3D intervals directly by using cuboids.

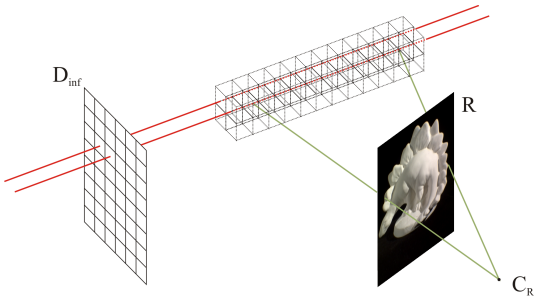

Fig. 5. A voxel equivalent visualization is generated with a desired view at infinity, $D_{\inf}$. Instead of projecting voxels to the reference image R, only 3D intervals on the parallel viewing rays of the camera at infinity are computed.

7. RESULTS

In this section we present the results in terms of runtime and resolution which are achieved with our algorithm on a single Geforce GTX 280 graphics card. For evaluation, we use the *dinoRig* dataset from the Middlebury benchmark [3] and real-world image data from our 3D videoconferencing setup. Moreover, we compare our results for the voxel equivalent representation on the Middlebury data with a recent voxel based technique proposed by Ladikos et al. [9]. The authors use four consumer PCs each of which is carrying one Geforce 8800 GTX to perform a distributed visual hull computation on the GPU. A fifth PC is used for recombination of the partial visual hull results and visualization. The Geforce 8800 GTX has a theoretical peak performance of 518 GFLOPS and a device throughput of 86.4 GB. For our experiments we have used a Geforce GTX 280 graphics card with 933 GFLOPS and a throughput of 141.7 GB. Comparing both systems on a hardware level one can see that the computational power in terms of GFLOPS and device throughput is more than twice as large for the system proposed by Ladikos.

7.1. Test Datasets

The Middlebury *dinoRig* dataset consists of 48 images with a resolution of 480×640 and the corresponding camera calibration data. For all our experiments we use silhouette images of the same size like the dataset images. The size of the desired image differs depending on the requested resolution as described in section 6. For the voxel equivalent image based visual hull calculation (VE-IBVH), we use resolutions of 64^3, 128^3, 256^3 and 512^3 voxels. An example of our result for 512^3 can be seen in fig. 6.

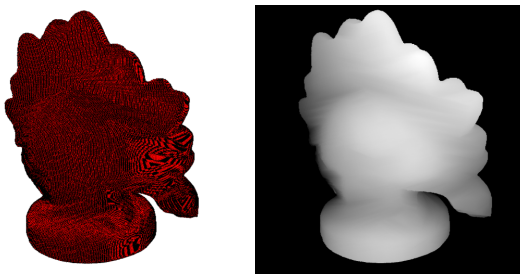

Fig. 6. VE-IBVH result for the *dinoRig* dataset, **left)** Voxel equivalent visualization with a resolution of 512^3 voxels, **right)** the corresponding depth map.

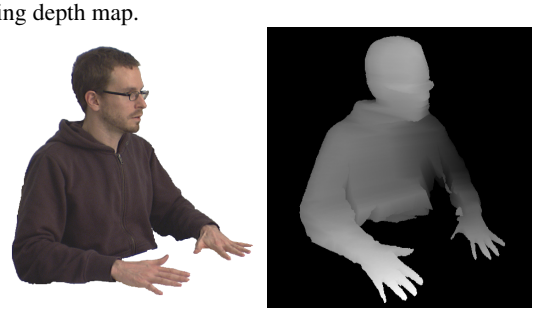

Fig. 7. Visual Hull result of a conferee computed by our 3D videoconferencing system. **left)** One of the segmented input images, **right)** the depth map of a desired view.

	64^3	128^3	256^3	512^3
GPU2_OT	39.94 ms	99.89 ms	296.71 ms	–
VENP-IBVH	38.95 ms	47.9 ms	82.5 ms	280.6 ms
VE-IBVH	**37.7 ms**	**41.6 ms**	**60.9 ms**	**150.6 ms**

Table 1. Timing values for the system reported in [9] (GPU2_OT) and our method (VE-IBVH) on the *dinoRig* dataset.VENP-IBVH denotes the runtimes without using our pixel pre-selection technique.

Ladikos et al. reported the peak performance of their system for using an octree based volume representation and an online computation of the voxel to image projections (GPU2_OT). For a comparison of timing values please refer to table 1. In order to show the impact of our novel pixel pre-selection technique, timings with no pre-selection (VENP-IBVH) are listed additionally. The timing values for our algorithm include the visual hull computation, the upload time of the image masks to GPU and the generation of the cuboids used for visualization. It can be clearly seen that our method outperforms the system proposed by Ladikos for all investigated resolutions on the *dinoRig* dataset up to a factor of about five. Since marching squares computation and line segment cache generation are identical for each voxel resolution and have to be done as preceding steps, only small timing changes can be observed by switching from 64^3 to 128^3. Even for the step from 128^3 to 256^3 the runtime of our method increases only by 50% while the runtime for GPU2_OT triples.

The VH input data of our real-time 3D videoconferencing system is a set of segmented images from 9 cameras at a resolution of 768×576. The desired view is also rendered at this resolution. The runtime of the depth map generation clearly depends on the scene complexity. For a one person video conferencing setup it is on average 33 ms. This allows us to run our system at 25 fps. For an illustration of an IBVH result of a video conferee refer to fig. 7.

8. CONCLUSION

In this work we presented a computationally efficient implementation of the Image Based Visual Hull approach on graphics hardware. A novel caching strategy was introduced and used for the preselection of relevant pixels in the desired image. We have shown, that especially for higher resolutions the speedup gained by the proposed pixel pre-selection is significant. Besides the view generation, we discussed the real-time computation of voxel equivalent representations using IBVH with cameras at infinity. Accordingly, we provided a unique framework which covers depth map rendering and 3D model computation as well.

9. ACKNOWLEDGMENTS

This work is part of the FP7 project "3DPresence", Proposal no.: FP7-215269, which is funded by the European Commission.

10. REFERENCES

[1] O. Schreer, I. Feldmann, W. Waizenegger, N. Atzpadin, P. Eisert, and H. Belt, "3dpresence a system concept for multi-user and multi-party immersive 3d videoconferencing," in *5th European Conf. on Visual Media Production (CVMP 2008)*, 2008, pp. 321–334.

[2] N. Atzpadin, P. Kauff, and O. Schreer, "Stereo analysis by hybrid recursive matching for real-time immersive video conferencing," in *IEEE Trans. on Circuits and Systems for Video Technology, Special Issue on Immersive Telecommunications, Vol. 14, No. 3*, 2004, pp. 321–334.

[3] S. Seitz, B. Curless, 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, pp. 519–528.

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

[5] Y. Furukawa and J. Ponce, "Accurate, dense, and robust multi-view stereopsis," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

[6] J. Franco and E. Boyer, "Exact polyhedral visual hulls," in *Proceedings of the Fourteenth British Machine Vision Conference*, 2003, pp. 329–338.

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

[8] 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.

[9] A. Ladikos, S. Benhimane, and N. Navab, "Efficient visual hull computation for real-time 3d reconstruction using cuda," in *Proceedings of the 2008 Conference on Computer Vision and Pattern Recognition Workshops*, 2008.

[10] NVIDIA Corp., "CUDA," http://www.nvidia.com/cuda.

[11] W. Matusik, C. Buehler, R. Raskar, L. McMillan, and S. Gortler, "Image-based visual hulls," in *Proceedings of ACM SIGGRAPH*, 2000, pp. 369–374.

[12] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," in *SIGGRAPH*. 1987, vol. 21, pp. 163–169, ACM Press.

[13] W. Matusik, C. Buehler, L. McMillan, and S. Gortler, "Efficient view-dependent sampling of visual hulls," Technical memo 624, Massachusetts Institute of Technology, 2002.