HAL
open science

Marker-less Real Time 3D Modeling for Virtual Reality

Jérémie Allard, Edmond Boyer, Jean-Sébastien Franco, Clément Ménier, Bruno Raffin

To cite this version:

Jérémie Allard, Edmond Boyer, Jean-Sébastien Franco, Clément Ménier, Bruno Raffin. Marker-less Real Time 3D Modeling for Virtual Reality. Immersive Projection Technology, May 2004, Ames, United States. inria-00349066

HAL Id: inria-00349066

https://inria.hal.science/inria-00349066v1

Submitted on 23 Dec 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Marker-less Real Time 3D Modeling for Virtual Reality

Jérémie Allard Edmond Boyer Jean-Sébastien Franco Clément Ménier Bruno Raffin
firstname.lastname@inrialpes.fr
Laboratoire Gravir, Laboratoire ID
INRIA Rhône-Alpes
655 avenue de l'Europe, 38334 Saint Ismier, France

Abstract

Today, immersive environments mainly use a few 3D positions given by trackers to align the real and virtual worlds. In this paper we present an alternative approach using commodity components to achieve real time marker-less 3D modeling in virtual reality environments. The goal is to compute in real time a 3D shape of the objects and users present in the interaction space without having to equip them with markers. Having full 3D shapes opens new possibilities for full-body interactions and a tight virtual/real world integration. Data acquisition is performed through cameras surrounding the interaction space. A visual hull reconstruction algorithm is parallelized on a PC cluster to compute in real time the 3D shapes of the scene observed. Experimental results using 4 cameras and 20 processors lead to a precise 3D human model built in real time.

1 Introduction

The sense of immersion provided by interactive virtual reality environments relies on the ability to retrieve 3D positions of user(s) in the interaction space. Such information is used to align real and virtual worlds. For instance, in cave like environments [9], stereoscopic images are computed from the user's viewpoint and require therefore the user's head position.

Today's tracking technology mainly focuses on giving the 3D positions of a few points in real time and with high precision. These points are identified with active or passive markers fixed on the user's body. Different technologies are used like electromagnetic waves, infrared, accelerometers, cameras or even GPS. These solutions provide different trade-offs regarding price, action range, precision, reliability, wearability and compatibility with other equipments. Such technology allows to track tens of points; it is commonly used in motion capture environments where the user is equipped with several markers giving the position of key parts of the user's body, like arms, forearms, head, legs, feet, etc. But such systems usually require to wear invasive equipments (and a tight body suit). In immersive environments, most of the time, only a few points are tracked, the user's head and an interaction device, a wand for example. Tracking only a few points shorten calibration procedures and ease passing the equipment from one user to the other.

In this paper we consider a non-invasive and marker-less technology based on a multi camera environment for 3D modeling. The idea is to use commodity cameras focused on the interaction space. Any new object introduced in the interaction space appears to several cameras. Knowing the positions of the cameras and combining the information acquired by each of them, it is possible to compute approximations of the observed objects' 3D shape. This approach being marker-less, it is easy to introduce or remove any objects or users in the scene. Each camera's pixel carrying a color information, it is also possible to extract texture data.

The 3D shape approximations that can be computed depend on the primitive that is considered in the 2D images. We use the *silhouettes* that are easy to extract from images in real time. The associated 3D model that can be computed, when several silhouettes of the same scene are available, is called the *visual hull*. It is the approximation of the scene objects' shape defined as the maximal volume consistent with all the silhouettes in the input views [16]. 3D models provided by visual hulls enable collision detection between the real objects and virtual world objects. These 3D models can also be rendered in the virtual world. This enables to render lifelike objects of the scene in a distant VR environment for instance.

In this paper we address the problems of the 3D modeling quality that depends on the number of cameras as well as on the algorithm chosen. In particular, we focus on an algorithm known for the quality of the 3D model obtained [10]. We also address performance and scalability issues. High quality reconstruction of a human can require 10 or more cameras. Multiple PCs are then necessary for video acquisition. Requirements for computation power are also important to reach real time processing. For interactive applications, it is essential to ensure a low latency and to match the frame rate of the cameras. We present parallelization

techniques on a PC cluster that enable real time executions. We also detail how the visual hull modeling is coupled with a multi-projector environment driven by a PC cluster using Net Juggler [3].

Multi-camera based systems also raise the problem of their practical integration into immersive environments. Cameras can mask some parts of the display surfaces and can be disturbed by the projected images. We do not yet address this issue as we experiment our system with a semi-immersive environment, a display wall. However, other research groups have experimented solutions on workbench [17] or cave like system [12].

After an outline of the global approach in section 2, we detail the visual hull 3D modeling algorithm tested in section 3. In section 4, we detail the implementation with the parallelization techniques used. Section 5 presents some experimental results.

2 Outline

Our goal is to provide a marker-less 3D modeling approach suitable for interactive applications in multi-projector environments. We target environments based on a PC cluster, multiple cameras for video acquisition and multiple projectors for rendering. PC clusters are cost effective, modular, scalable, and support a large range of devices. Digital cameras are today commodity components available from low cost webcams to high-end 3-CCD cameras. Images provided by webcams proved of insufficient quality (low resolution and refresh rate, important optic distortions) that made them unsuitable for our purpose. Mid-range Firewire cameras are used in our case.

Cameras are set to surround the scene. The number of cameras required depends on the size of the scene and the complexity of the objects to model. Beyond a certain number of cameras, the quality of the 3D model obtained does not improve or can even decrease due to numerical imprecision. More details about this issue are given in section 5. The cameras have to be carefully calibrated, i.e. their relative locations and orientations as well as their intrinsic characteristics must be determined. This process of camera calibration is today well understood and several algorithms exist. We use the OpenCV library from Intel for that purpose [14].

From the 2D video streams provided by the cameras to the 3D model, several steps are required. First of all, each 2D image obtained from a camera must be analyzed to extract regions of interest. We assume that the scene is composed of a static background that can be learned in advance and foreground objects (a person typically). Regions of interest -the foreground- are thus extracted using a standard background subtraction process [23]. These regions are then vectorized, i.e. theirs delimiting contours are computed. We do not keep texture information but only geometric information -*the silhouettes*- about these regions since we focus on a geometric model, the *visual hull*. Such models are sufficient for a large range of computations like collision detection or virtual shadow computation for instance. Texturing the obtained model surface is an on-going work. Foreground extraction and silhouette segmentation are performed locally on each PC connected to a camera.

The visual hull computation then takes place. Basically, a visual hull is computed from the intersection of the viewing cones. A viewing cone is associated to a camera and corresponds to the volume that projects into the silhouette. The final result is a polyhedron model. More details about the algorithm used are given in section 3. Computing the visual hull is time consuming. To reach a real time execution, the 3D modeling workload is distributed on different processors. Once computed, the obtained 3D mesh is asynchronously sent to the rendering PCs. Multi-projector rendering is handled by the VR Juggler and Net Juggler association [3, 2, 6]. Net Juggler provides cluster support for VR Juggler 1.0. Details about the parallel implementation are given in section 4.

3 3D Modeling Using Cameras

3.1 Overview

As previously mentioned, we elected to model scene objects by reconstructing their shape from silhouettes gathered by multiple cameras. The interest in using silhouettes for modeling is that they are easily extracted from known static backgrounds in the images. Using silhouettes, we have access to an approximation of the shape of scene objects called the *visual hull*. It is the maximal solid shape consistent with the silhouettes, i.e. it projects onto silhouettes in all images. Geometrically, the visual hull is the intersection of the *viewing cones*, the generalized cones whose apex are the cameras' projective centers and whose cross-sections coincide with the scene silhouettes.

Silhouettes were first considered by Baumgart [5] who proposed to compute polyhedral shape approximations by intersecting silhouettes or viewing cones. The term visual hull was later coined by Laurentini [16] to describe the maximal volume compatible with a set of silhouettes. Following Baumgart's work, a number of modeling approaches based on silhouettes have been proposed. They can be roughly separated into two categories: volume based approaches and surface based approaches.

The first category includes methods that approximate visual hulls by collections of voxels, i.e. cells from regular space discretizations, which are such that they project onto silhouettes in any images. Efficient approaches [22, 20, 7]

have been presented to compute such voxel-based representations. See [21] for a review on volume based modeling approaches. Real time and parallel implementations have also been proposed [15, 4]. All these approaches are based on regular voxel grids and can handle objects with complex topologies. However, the space discretizations used are computationally expensive and lead to approximations only which lack precision.

When considering piecewise-linear image contours for silhouettes, the visual hull becomes a regular polyhedron. The second category of approaches estimates elements of this polyhedron by intersecting viewing cones. This includes several works which focus on individual points reconstructed using local second order surface approximations, see [8] for a review. Approaches have also been proposed to compute individual strips [19] of the visual hull. The approach we use in this paper belongs to this category and computes the complete visual hull polyhedron as a whole with an optimal number of operations [10]. Details are given in the next section.

Some of the approaches in the above categories make use of a graphic card for computations [18, 13]. Using graphic card hardware highly accelerates computations but such systems still rely on a single PC for computations, limiting scalability of the acquisition process to few cameras. Moreover, results from graphic cards are 2D rendered images and significant additional efforts are required to provide explicit 3D models, as frequently needed in virtual reality applications. The strategy we employ to achieve real time is therefore to distribute 3D modeling computations over a cluster of PCs as detailed in section 4.

3.2 The Modeling Method

Figure 1: Real time reconstruction of a real person with 4 cameras.

As mentioned above, several methods are available to reconstruct the visual hull of scene objects. We choose to use surface-based methods, that is we aim to recover the scene objects' shape as a triangle-mesh surface.

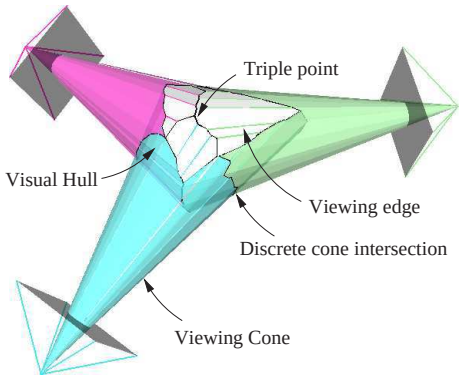


Figure 2: Visual hull of a sphere with 3 views.

Recall that the visual hull, with a finite number of viewpoints, is the result of a generalized cone intersection (figure 2). It is important to realize that all surface information of the visual hull is implicitly contained in the contours of silhouettes in the images. This is because silhouettes backproject to *viewing cones* in space, and that the visual hull surface consists of cone intersection primitives. Namely, each image's silhouette contour will contribute to the visual hull surface in the form of a *strip*, that is the cone surface fragment lying inside all other viewing cones.

Computing such geometry components is made easier by discretizing the silhouette contours in images, that is by building a polygonal representation of these contours. Such a representation induces a set of polyhedral viewing cones, and therefore a polyhedral visual hull. All the geometry of the contours, and therefore of the viewing cones, implicitely reduces to the knowledge of the contour polygon's vertices and their positions. As such, each contour vertex contributes to the visual hull's explicit representation, i.e. its triangle-mesh surface. The contribution for a single contour vertex lies along this vertex's *viewing line*, the line which backprojects from this vertex into space with respect to the viewpoint in which it is defined. Such a contribution is the part of this viewing line that belongs to the visual hull: it is the part of the viewing line which is inside all viewing cones in space (see figure 3). Or equivalently, it is the segments along this viewing line which project inside the silhouettes in all other images. These segments are called *viewing edges*, as they happen to be edges of the final visual polyhedron, as visible in figures 1 and 2.

The viewing edges are a set of segments on the visual hull surface. They form an incomplete representation of the visual hull and work remains to be done to retrieve the surface information, the triangle mesh description of the visual hull. In this paper, we choose to use an efficient method [10] to compute the cone intersection exactly.

In order to achieve this goal this method aims at incrementally recovering the missing geometry. As seen previ-

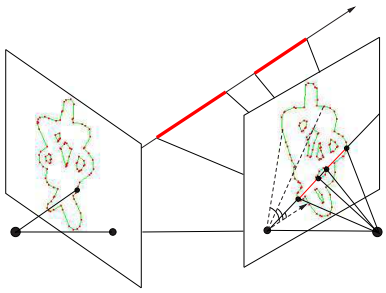


Figure 3: Viewing edges (in bold) along the viewing line of a contour vertex, using 2 silhouettes.

ously, the viewing edges give us an initial subset of the visual hull geometry. However, this does not account for all edges and points of the visual hull polyhedron (see figure 2). Typically, *triple points*, points of the surface which are the locus of three viewing cone intersections, project on each image's silhouette contour, at an arbitrary point. In general, this projection point does not coincide with any vertex of the 2D silhouette contour polygon. For this reason, these triple points are not computed as part of the initial viewing edge vertex set.

For a complete visual hull, one must therefore compute not only the missing triple points, but also the missing edges connecting triple points to the already computed viewing edges. Due to the discrete nature of the visual hull in our context, it is possible to incrementally recover these edges and triple points. Information about previously created edges is collected to infer the position and direction of neighbouring edges. This enables the algorithm to incrementally follow and reconstruct the missing geometry.

Once the complete mesh is recovered, faces of the visual hull polyhedron surface are extracted, by walking through the complete oriented mesh while always taking left turns at each vertex, so as to identify each face's 2D contours.

To summarize, this method achieves the reconstruction task in three steps. The first step computes viewing edges. Since this only partially accounts for all surface geometry, the second step's goal is to incrementally compute the missing surface points and edges, as described above. The third step identifies faces in the mesh and builds the triangle set.

See [10] for a detailed description of the modeling method.

4 Parallel Implementation

In this section we detail the approach used to distribute the work load on different processors to reach a real time execution. We distinguish 3 main parts: video acquisition and processing, 3D modeling and multi-projector rendering.

4.1 Video Acquisition and Processing

Cameras are distributed on different PCs to ensure that acquisition can be performed at the highest frame rate and resolution. Each of these PCs locally executes the background subtraction as well as the silhouette vectorization. We call a `frame` each new set of data obtained from the different cameras at a given time. Once the silhouettes are computed, each one is sent to the PCs in charge of the first step of the 3D Modeling.

4.2 3D Modeling Parallelization

3D Modeling is divided in a 3 stage pipe-line distributed on the processors of the cluster. To balance the work load of the different pipe-line stages, heavily loaded stages have a pool of processors, called *processing units*, that cyclically process new incoming frames. A multi processing unit pipe-line can significantly improve the processing frame rate. However, the latency is negatively affected. The pipe-line introduces extra communication time that increases the latency, thus affecting the system reactivity. To improve latency, one can reduce the time required by an individual stage to process a single frame. This can be done by parallelizing the work done on a single frame among several hosts of a single processing unit.

The first modeling step consists in the computation of a set of numerous partial but independent results. That is, each viewing line's contributions – the silhouettes – can be computed regardless of all others, assuming the silhouette information from all images is available. Viewing edge computation can hence be obtained by partitioning all viewing lines from all images in p sets and distributing each batch to one of p hosts for processing. One must be careful in balancing the workload between hosts, in order to reduce the time spent waiting for the slowest host. Building sets of identical cardinality proved to be efficient. This step ends by gathering the union of all partial results on the hosts that require it for the next pipe-line stage. Speed-ups, obtained dividing the sequential execution time by the parallel execution time, of the order of 8 with 10 hosts are thus obtained.

The second step's parallelization consists in partitioning space into p different regions using $p-1$ parallel planes, thus subdividing space in p "slices". Slice width is adjusted by attributing a constant number of viewing edge vertices per slice for workload balancing. Each host is assigned a slice $\mathcal{R}_i$ and follows recovered edges within $\mathcal{R}_i$, until one edge crosses the border toward another region $\mathcal{R}_j$. The host then stops processing this edge, thereby delegating the remaining computation related to this edge to the host in charge of $\mathcal{R}_j$. Partial meshes are then gathered and carefully merged across slice borders. Speed-ups of 6 with 10 hosts are thus reached.

For the third step, the surface extraction, parallelization

is quite simple. The complete mesh is first broadcasted to p hosts, then the hosts independently compute a subset of the face information. The step ends by gathering the results on a single host to build the final 3D mesh. This leads to very good speed-ups of the order of 7 for 10 hosts.

4.3 Multi-Projector Rendering

The multi-projector rendering is handled by the VR Juggler and Net Juggler association [3, 2, 6]. Net Juggler provides cluster support for VR Juggler 1.0. A copy of the VR Juggler application is executed on each PC driving a video projector. To keep the different copies consistent, Net Juggler intercepts all incoming events and broadcasts them to all copies. It also implements a swaplock to ensure a synchronized frame buffer swap. Though this parallelization leads to data and computation replication, it leads to a good performance as the amount of data sent over the network is limited.

Once a new 3D mesh model is computed, it must be sent to each rendering node. The volume of data being quite small, we use a simple TCP point to point channel. As the visualization process should not be limited by the speed of the reconstruction, we use an asynchronous approach. At each rendering iteration the rendering PCs read any new result available without waiting for it if none has arrived yet. This architecture proved to be very simple and quite effective.

5 Experimental Results

In this section, we detail experimental results obtained on the GrImage platform. We first tested the 3D model quality, latency and frame rate with 4 cameras. Next, tests with synthetic data show that sustained performance is obtained with a large number of view points (virtual cameras).

Our implementation uses the standard MPI message passing library [11] for communications. All presented results are based on sets of 10 experiments.

5.1 The GrImage Platform

Our tests are performed on the GrImage platform located at the INRIA Rhône-Alpes. GrImage gathers 4 Sony Cameras DFW-VL500 each one connected to a Pentium 4 PC, 11 dual-Xeon 2.6 GHz PCs connected by a gigabit Ethernet network, and a display wall of 8 projectors. Four PCs are dedicated to rendering, each one driving 2 projectors with a NVIDIA GeForce FX 5600 graphics board. The cameras provide 640x480 images. They were set to capture the scene in front of the display. The cameras were not synchronized to keep a 30 Hz frame rate. Genlocking the cameras decreases the frame rate to 15 Hz. The lack of genlock can lead to silhouette inconsistencies. As the user does not generally move too fast, we chose to favor a higher frame rate to test the real time abilities of our system. The test application consists in rendering the raw 3D model on the multi-projector display. It allows to evaluate the quality of the results and the latency.

5.2 Four Camera Setup

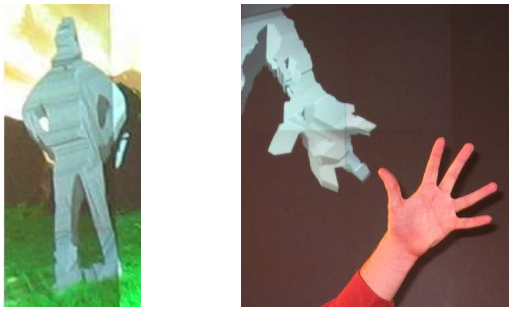
Figure 4 shows the display wall of GrImage platform with in front the user being shot by 4 cameras. On the display we can distinguish the 3D model obtained and also shown in figure 5(a). A user consists produces about 150 contour vertices per image. The 3D model is obtained at a frame rate of about 30 fps with an average latency of about 300 ms. For more complex scenes, with several users for example, the frame rate drops to 15 fps. This limitation should disappear using more PCs.

Figure 4: Real time modeling and visualization in the GrImage platform.

With only 4 cameras, we obtain a good quality 3D model. For instance, figure 5(b) shows that from the 4 cameras views the 3D modeling algorithm can build hand fingers. Note that this result is obtained when the user from figure 4 raises a hand. Camera position and tuning are not modified.

Videos[1] give some insights of the real time abilities of the system.

[1] http://www-id.imag.fr/~allardj/ipt04



(a) (b)

Figure 5: (a) Close-up on the 3D model of figure 4 (b) Detail of a 3D model: its precision is sufficient to detect fingers.

5.3 Scalability for a Large Number of Viewpoints

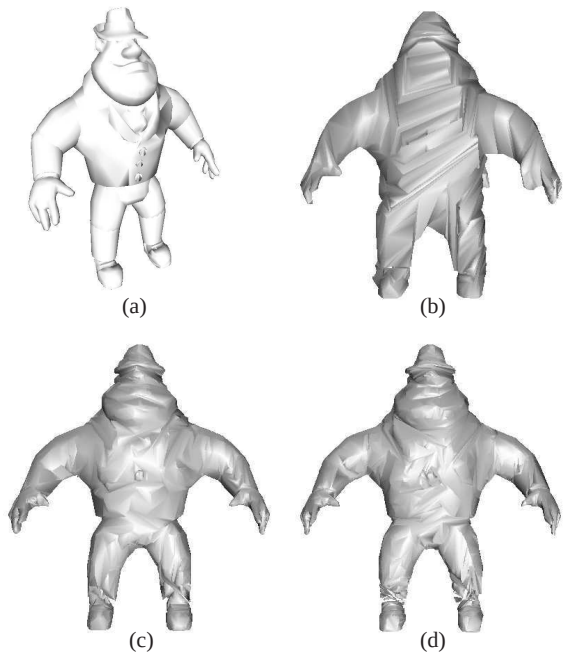
At the moment, 4 cameras are available in our experimental setup. To test the scalability of our 3D modeling method we used a synthetic model (see figure 6(a)) with a complexity close to a real person (about 130 contour vertices per image). This model is used to compute images from as many viewpoints as required. Next, our parallel 3D modeling algorithm is executed on this frame. It enables to focus on latency as keeping a real time frame rate is just a matter of multiplying the number of processing units (assuming the network switch performance scales with the number of PCs). We used the same number p of processors for each step. Due to the limited number of processors available, we reuse the same hosts for each pipe-line stage since it does not significantly influence latency.

5.3.1 3D Modeling Quality

The precision of the 3D modeling depends on the number of cameras. Figure 6 shows the impact of the number of viewpoints on the 3D model quality. In this case, using 25 viewpoints clearly enhance the precision of the modeling compare to using only 6 views. However increasing the number of viewpoints does not always lead to higher precision, due to the increased sensitivity to camera calibration and pixel discretization noise.

5.3.2 Latency

Experimental results (see figure 7) show that parallelization enables to divide the latency by almost an order of magnitude for 25 cameras. To reach real time processing with 12 cameras (100 ms in latency and 30 frames per second), we would assign one processing unit per pipe-line stage, each one having 5 processors (a total of 15 PCs). Notice that latency is here measured from the beginning of the viewing-edge step to the completion of the 3D mesh computation.



(a) (b) (c) (d)

Figure 6: 3D Modeling of a synthetic person: (a) original model; (b),(c) and (d) 3D model obtained with respectively 6, 25 and 64 cameras.

6 Future Work

The algorithm presented only provides a geometric model. For applications requiring to display the 3D model, collaborative work for instance, this model is usually too poor to easily be able to identify a person or an object. We are currently working on extracting color information from the different cameras to dynamically compute a texture and apply it to the geometrical model. Preliminary results are encouraging (see figure 8). Adding a texture, even with some important defects, greatly improve the likeness of the computed model with the original one.

The parallel 3D modeling algorithm presented here was programmed using the MPI library [11]. MPI is well adapted to implement large scientific parallel applications, but MPI communication calls are deeply embedded into the code, making it difficult to change a communication pattern or to modify one part of the algorithm. We are currently porting our code on top of FlowVR [1], a middleware dedicated to distributed interactive applications. FlowVR clearly separates the code of each distributed component from the communications. It enables to completely re-

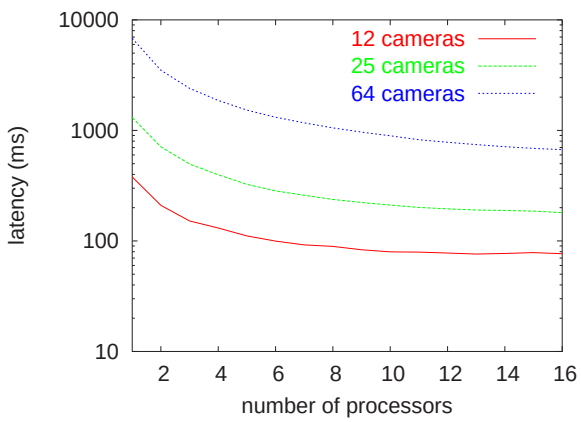


Figure 7: Log plot latencies for the synthetic person (see figure 6(a)).

Figure 8: Texturing the 3D model: preliminary results with 4 cameras.

design the communication patterns without modification to the component code. This eases code reuse and code coupling.

Experiments with more cameras are planned as GrImage should be upgraded to 20 cameras, 30 PCs and 16 projectors. We are also considering executing the visual hull algorithm on an Itanium 2 cluster, keeping the video acquisition and rendering stages on GrImage.

7 Conclusion

In this paper we presented a commodity component based approach for marker-less 3D modeling in virtual reality environments. Digital cameras surround the interaction space where the objects and humans to model are introduced. A visual hull modeling algorithm is parallelized on a PC cluster to compute in real time 3D shapes of the observed scene. Experimental results using 4 cameras and 20 processors lead to a precise 3D human model built at a sustained 30 Hz frame rate and with an average latency close to 300 ms. It shows that visual hull modeling is a suitable technology to provide a global 3D shape for virtual reality while marker based tracking rather provides a few points usually with a low latency and high precision.

References

[1] FlowVR. `http://flowvr.sf.net`.

[2] Net Juggler. `http://netjuggler.sf.net`.

[3] J. Allard, V. Gouranton, L. Lecointre, E. Melin, and B. Raffin. Net Juggler: Running VR Juggler with Multiple Displays on a Commodity Component Cluster. In *IEEE VR*, pages 275–276, Orlando, USA, March 2002.

[4] D. Arita and R.-I. Taniguchi. RPV-II: A Stream-Based Real-Time Parallel Vision System and Its Application to Real-Time Volume Reconstruction. In *Computer Vision Systems, Second International Workshop, ICVS, Vancouver (Canada)*, 2001.

[5] B.G. Baumgart. A polyhedron representation for computer vision. In *AFIPS National Computer Conference*, 1975.

[6] A. Bierbaum, C. Just, P. Hartling, K. Meinert, A. Baker, and C. Cruz-Neira. VR Juggler: A Virtual Platform for Virtual Reality Application Development. In *IEEE VR 2001*, Yokohama, Japan, March 2001.

[7] G. Cheung, T. Kanade, J.-Y. Bouguet, and M. Holler. A real time system for robust 3d voxel reconstruction of human motions. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Hilton Head Island, (USA)*, volume 2, pages 714 – 720, June 2000.

[8] R. Cipolla and P.J. Giblin. *Visual Motion of Curves and Surfaces*. Cambridge University Press, 1999.

[9] C. Cruz-Neira, D. J. Sandin, T. A. DeFanti, R. V. Kenyon, and J. C. Hart. The Cave Audio VIsual Experience Automatic Virtual Environement. *Communication of the ACM*, 35(6):64–72, 1992.

[10] J.S. Franco and E. Boyer. Exact Polyhedral Visual Hulls. In *Proceedings of the 14th British Machine Vision Conference, Norwich, (UK)*, 2003.

[11] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. Scientific and Engeneering Computation Series. The MIT Press, 1994.

[12] M. Gross, S. Wuermlin, M. Naef, E. Lamboray, C. Spagno, A. Kunz, E. Koller-Meier, T. Svoboda, L. Van Gool, K. Strehlke S. Lang, A. Vande Moere, and O. Staadt. Blue-c: A spatially immersive display and 3d video portal for telepresence. In *ACM SIGGRAPH 2003*, San Diego, 2003.

[13] J.-M. Hazenfratz, 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 Conference*, 2003.

[14] Intel. Open Source Computer Vision Library. http://www.intel.com/research/mrl/research/opencv/.

[15] Y. Kameda, T. Taoda, and M. Minoh. High Speed 3D Reconstruction by Spatio Temporal Division of Video Image Processing. *IEICE Transactions on Informations and Systems*, (7):1422–1428, 2000.

[16] A. Laurentini. The Visual Hull Concept for Silhouette-Based Image Understanding. *IEEE Transactions on PAMI*, 16(2):150–162, February 1994.

[17] B. Leibe, D. Minnen, J. Weeks, and T. Starner. Integration of wireless gesture tracking, object tracking, and reconstruction in the responsive workbench. In B. Schiele and G. Sagerer, editors, *ICVS 2001*, volume 2095 of *LNCS*, pages 73–92, 2001.

[18] M. Li, M. Magnor, and H.-P. Seidel. Improved hardware-accelerated visual hull rendering. In *Vision, Modeling and Visualization Workshop, Munich, (Germany)*, 2003.

[19] W. Matusik, C. Buehler, and L. McMillan. Polyhedral Visual Hulls for Real-Time Rendering. In *Eurographics Workshop on Rendering*, 2001.

[20] W. Niem. Automatic Modelling of 3D Natural Objects from Multiple Views. In *European Workshop on Combined Real and Synthetic Image Processing for Broadcast and Video Production, Hamburg, Germany*, 1994.

[21] G. Slabaugh, B. Culbertson, T. Malzbender, and R. Schafe. A Survey of Methods for Volumetric Scene Reconstruction from Photographs. In *International Workshop on Volume Graphics*, 2001.

[22] R. Szeliski. Rapid Octree Construction from Image Sequences. *Computer Vision, Graphics and Image Processing*, 58(1):23–32, 1993.

[23] K. Toyama, J. Krumm, B. Brumitt, and B. Meyers. Wallflower: Principles and Practice of Background Maintenance. In *Proceedings of the 7th International Conference on Computer Vision, Kerkyra, (Greece)*, pages 255–261, 1999.