Hindawi Publishing Corporation
International Journal of Digital Multimedia Broadcasting
Volume 2010, Article ID 247108, 12 pages
doi:10.1155/2010/247108

Research Article

Multicamera Real-Time 3D Modeling for Telepresence and Remote Collaboration

Benjamin Petit,[1] Jean-Denis Lesage,[2] Clément Menier,[3] Jérémie Allard,[4] Jean-Sébastien Franco,[5] Bruno Raffin,[6] Edmond Boyer,[7] and François Faure[7]

[1] *INRIA Grenoble, 655 avenue de l'Europe, 38330 Montbonnot Saint Martin, France*
[2] *Université de Grenoble , LIG, 51 avenue Jean Kuntzmann, 38330 Montbonnot Saint Martin, France*
[3] *4D View Solutions, 655 avenue de l'Europe, 38330 Montbonnot Saint Martin, France*
[4] *INRIA Lille-Nord Europe, LIFL, Parc Scientifique de la Haute Borne, 59650 Villeneuve d'Ascq, France*
[5] *Université Bordeaux, LaBRI, INRIA Sud-Ouest, 351 cours de la Libération, 33405 Talence, France*
[6] *INRIA Grenoble, LIG, 51 avenue Jean Kuntzmann, 38330 Montbonnot Saint Martin, France*
[7] *Université de Grenoble, LJK, INRIA Grenoble, 655 avenue de l'Europe, 38330 Montbonnot Saint Martin, France*

Correspondence should be addressed to Benjamin Petit, benjamin.petit@inrialpes.fr

Received 1 May 2009; Accepted 28 August 2009

Academic Editor: Xenophon Zabulis

We present a multicamera real-time 3D modeling system that aims at enabling new immersive and interactive environments. This system, called Grimage, allows to retrieve in real-time a 3D mesh of the observed scene as well as the associated textures. This information enables a strong visual presence of the user into virtual worlds. The 3D shape information is also used to compute collisions and reaction forces with virtual objects, enforcing the mechanical presence of the user in the virtual world. The innovation is a fully integrated system with both immersive and interactive capabilities. It embeds a parallel version of the EPVH modeling algorithm inside a distributed vision pipeline. It also adopts the hierarchical component approach of the FlowVR middleware to enforce software modularity and enable distributed executions. Results show high refresh rates and low latencies obtained by taking advantage of the I/O and computing resources of PC clusters. The applications we have developed demonstrate the quality of the visual and mechanical presence with a single platform and with a dual platform that allows telecollaboration.

1. Introduction

Teleimmersion is of central importance for the next generation of live and interactive 3DTV applications. It refers to the ability to embed persons at different locations into a shared virtual environment. In such environments, it is essential to provide users with a credible sense of 3D telepresence and interaction capabilities. Several technologies already offer 3D experiences of real scenes with 3D and sometimes free-viewpoint visualizations, for example, [1–4]. However, live 3D teleimmersion and interaction across remote sites is still a challenging goal. The main reason is found in the difficulty to build and transmit models that carry enough information for such applications. This not only covers visual or transmission aspects but also the fact that such models need to feed 3D physical simulations as required for interaction purposes. In this paper, we address these issues and propose a complete framework allowing the full body presence of distant people into a single collaborative and interactive environment.

The interest of virtual immersive and collaborative environments arises in a large and diverse set of application domains, including interactive 3DTV broadcasting, video gaming, social networking, 3D teleconferencing, collaborative manipulation of CAD models for architectural and industrial processes, remote learning, training, and other collaborative tasks such as civil infrastructure or crisis management. Such environments strongly depend on their ability to build a virtualized representation of the scene of interest, for example, 3D models of users. Most existing systems use 2D representations obtained using mono-camera systems [5–7]. While giving a partially faithful representation of the user, they do not allow for natural interactions, including

consistent visualization with occlusions, which require 3D descriptions. Other systems more suitable for 3D virtual worlds use avatars, as, for instance, massive multiplayer games analog to *Second Life*. However, avatars only carry partial information about users and although real-time motion capture environments can improve such models and allow for animation, avatars do not yet provide sufficiently realistic representations for teleimmersive purposes.

To improve the sense of presence and realism, models with both photometric and geometric information should be considered. They yield more realistic representations that include user appearances, motions and even sometimes facial expressions. To obtain such 3D human models, multicamera systems are often considered. In addition to appearance, through photometric information, they can provide a hierarchy of geometric representations from 2D to 3D, including 2D and depth representations, multiple views, and full 3D geometry. 2D and depth representations are viewpoint dependent and though they enable 3D visualization [8] and, to some extent, free-viewpoint visualization, they are still limited in that respect. Moreover they are not designed for interactions that usually require full shape information instead of partial and discrete representations. Multiple view representations, that is, views from several viewpoints, overcome some of the limitations of 2D and depth representations. In particular, they increase the free-viewpoint capability when used with view interpolation techniques, for example, [3, 9, 10]. However, interpolated view quality rapidly decreases when new viewpoints distant from the original viewpoints are considered. And similarly to 2D and depth representations, only limited interactions can be expected. In contrast, full 3D geometry descriptions allow unconstrained free viewpoints and interactions as they carry more information. They are already used for teleimmersion [2, 4, 11, 12]. Nevertheless, existing 3D human representations, in real-time systems, often have limitations such as imperfect, incomplete, or coarse geometric models, low resolution textures, or slow frame rates. This typically results from the complexity of the method applied for 3D reconstruction, for example, stereovision [13] or visual hull [14] methods, and the number of cameras used.

This article presents a full real-time 3D-modeling system called Grimage (http://grimage.inrialpes.fr/) (Figure 1). It is an extended version of previous conference publications [15–19]. The system relies on the EPVH-modeling algorithm [14, 20] that computes a 3D mesh of the observed scene from segmented silhouettes and robustly yields an accurate shape model [14, 20] at real-time frame rates. Visual presence in the 3D world is ensured by texturing this mesh with the photometric data lying inside the extracted silhouettes. The 3D mesh also enables a mechanical presence, that is, the ability for the user to apply mechanical actions on virtual objects. The 3D mesh is plugged into a physics engine to compute the collisions and the reaction forces to be applied to virtual objects. Another aspect of our contribution is the implementation of the pipeline in a flexible parallel framework. For that purpose, we rely on FlowVR, a middleware dedicated to parallel interactive application [21–23]. The application is structured through a hierarchy of components, the leaves being computation tasks. The component hierarchy offers a high-level of modularity, simplifying the maintenance and upgrade of the system. The actual degree of parallelism and mapping of tasks on the nodes of the target architecture are inferred during a preprocessing phase from simple data like the list of cameras available. The runtime environment transparently takes care of all data transfers between tasks, being on the same node or not. Embedding the EPVH algorithm in a parallel framework enables to reach interactive execution times without sacrificing accuracy. Based on this system we developed several experiments involving one or two modeling platforms.

In the following, we detail the full pipeline, starting with acquisition steps in Section 2, the parallel EPVH algorithm in Section 3, the textured-model rendering and the mechanical interactions in Section 4. A collaborative set up between two 3D-modeling platforms is detailed in Section 6. Section 7 present a few experiments and the associated performance results, before concluding in Section 8.

2. A Multicamera Acquisition System

To generate real-time 3D content, we first need to acquire 2D information. For that purpose we have built an acquisition space surrounded by a multicamera vision system. This section will focus on the technical characteristics needed to obtain an image stream from multiple cameras and to transform it into suitable information for the 3D-modeling step, that is, calibrated silhouettes.

2.1. Image Acquisition. As described previously, the 3D-modeling method we use is based on images. We thus need to acquire video streams from digital cameras. Today digital cameras are commodity components available from low cost webcams to high-end 3-CCD cameras. Images provided by current webcams proved to be of insufficient quality (low resolution and refresh rates, important optical distortion), which made them unsuitable for our purpose. Consequently we use mid range firewire cameras acquiring up to 30 fps and 2 megapixels color images. Our acquisition platform is equipped with up to 16 cameras. Each camera is connected to a cluster node dedicated to image processing. A software library is used to control our cameras, managing cameras' configurations and frame grabbing under Linux.

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 as well as on the quality required for texturing and 3D-modeling. Beyond a certain number of cameras, the accuracy of the model obtained does not significantly improve while the network load increases resulting in higher latencies. Experiments have shown that 8 cameras is generally a good compromise between the model accuracy and the CPU and network load.

The camera locations in the acquisition space usually depends on the application: one can choose to emphasize a side of the set up to have better texture quality in a particular direction, the user's face for example, or to place more cameras in a certain area to get better models of the arms and hands for interaction purposes.

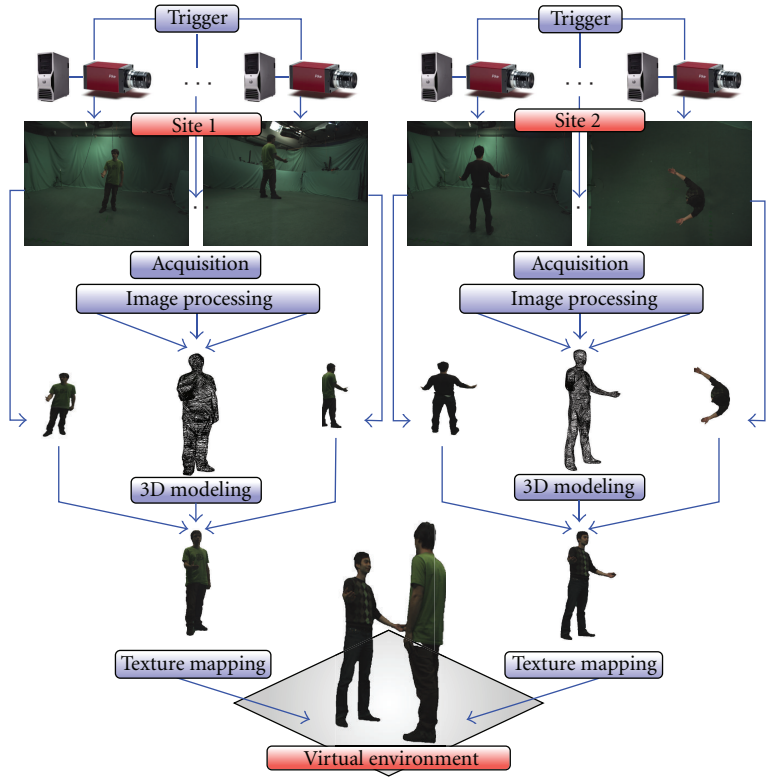


Figure 1: An illustration of the telepresence application pipeline.

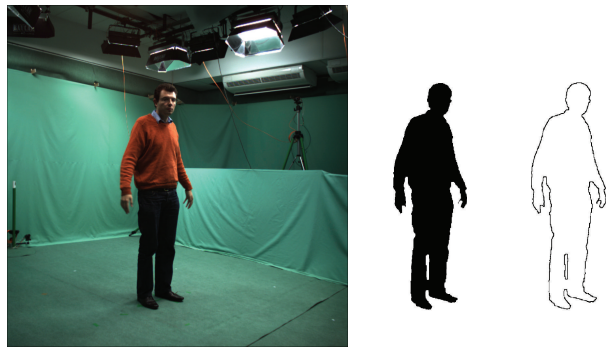
2.2. Synchronization. Dealing with multiple input devices raises the problem of data synchronization. In fact all our applications rely on the assumption that the input images captured from the different cameras are coherent, that is, that they relate to the same scene event. Synchronization information could be recovered directly from silhouettes using their inconsistency over several viewpoints as suggested in [24]; however, a hardware solution appears to be more practical and efficient in a dedicated environment such as ours. The image acquisition is triggered by an external signal sent directly through the cameras' genlock connector. This mechanism leads to delays between images below 100 microseconds

2.3. Calibration. Another issue when dealing with multiple cameras is to determine their spatial organization in order to perform geometric computations. In practice we need to determine the position and orientation of each camera in the scene as well as its intrinsic characteristics such as the focal length. This is done through a calibration process that computes the function giving the relationship between real 3D points and 2D-image points for each camera.

As with the synchronization step, the silhouette information could also be used to recover calibration information, using for instance [24, 25]. However practical considerations on accuracy favor again a solution which is specific to our dedicated environment. We perform this step using a software we developed which is based on standard off-the-shelf calibration procedures, see for instance [26, 27]. The calibration process consists in sweeping around the scene a wand with four lights with known relative positions on the wand. Once the lights are tracked through time in each image, a bundle adjustment iteratively lowers the reprojection error of the computed 3D light positions into the original images by adjusting the extrinsic and intrinsic parameters of each camera.

2.4. Background Subtraction. Regions of interest in the images, that is, the foreground or silhouette, are extracted using a background subtraction process. We assume that the scene is composed of a static background, the appearance of which can be learned in advance. As most of the existing techniques [28, 29], we rely on a per-pixel color model of the background. For our purpose, we use a combination of a Gaussian model for the chromatic information (UV) and an interval model for the intensity information (Y) with a variant of the method by Horprasert et al. [28] for shadow detection (Figure 2(b)). A crucial remark here is that the accuracy of the produced 3D model highly depends on this process since the modeling approach is exact with respect to the silhouettes. Notice that a high-quality background subtraction can easily be achieved by using a dedicated environment (blue screen). However, for prospective purposes, we do not limit our approach to such specific environments in our set up.

2.5. Silhouette Polygonalization. Since our modeling algorithm computes a surface and not a volume, it does not use



(a) (b) (c)

Figure 2: The different steps of the image processing: (a) image acquisition, (b) background subtraction (binary image), and (c) exact silhouette polygon (250 vertices).

image regions as defined by silhouettes, but instead their delimiting polygonal contours. We extract such silhouette contours and vectorize them using the method of Debled-Rennesson et al. [30]. Each contour is decomposed into an oriented polygon, which approximates the contour to a given approximation bound. With a single-pixel bound, obtained polygons are strictly equivalent to the silhouettes in the discrete sense (Figure 2(c)). However in case of noisy silhouettes this leads to numerous small segments. A higher approximation bound results in significantly fewer segments. This enables to control the model complexity, and therefore the computation time of the 3D-modeling process, in an efficient way.

3. 3D Modeling

To obtain a 3D geometric model of objects and persons located in the acquisition space, we use a shape-from-silhouette method, which builds a shape model called the visual hull. Shape-from-silhouette methods are well adapted to our context for several reasons. First, they yield shape models as required later in the process, for example, texture mapping or interaction. Second, they provide such models in real-time. Even though more precise approaches exist, for example, [31–33], most will fail at providing 3D models in real-time over long period of time and in a robust and efficient way as shape-from-silhouette approaches do. Below, we precise our shape-from-silhouette method.

3.1. Visual Hull. The visual hull is a well-studied geometric shape [34, 35] which is obtained from scene object's silhouettes observed in n views. Geometrically, the visual hull is the intersection of the *viewing cones*, the generalized cones whose apices are the cameras' projective centers and whose cross-sections coincide with the scene silhouettes (Figure 3). When considering piecewise-linear image contours for silhouettes, the visual hull becomes a regular polyhedron. A visual hull cannot model concavities but can be efficiently computed and yield a very good human shape approximation.

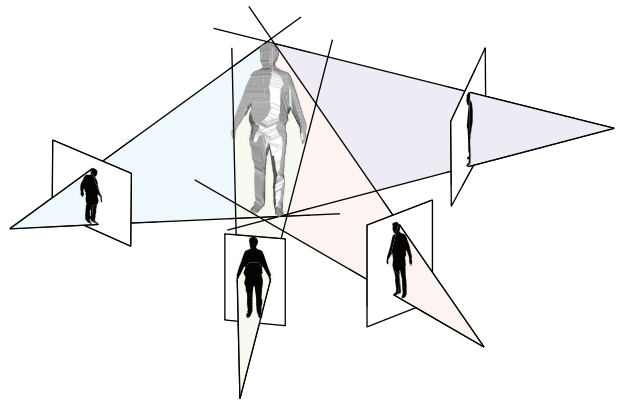


Figure 3: Visual hull of a person with 4 views.

Our work is based on the exact polyhedron visual hull (EPVH) algorithm [14, 20]. The EPVH algorithm has the particularity of retrieving an exact 3D model, whose projection back into the images coincides with the observed silhouettes. This is an important feature when the models need to be textured as it makes textures, extracted from silhouettes, directly mappable on the 3D model.

The method we present here recovers the visual hull of a scene object in the form of a polyhedron. As previously explained, silhouette contours of the scene object are retrieved for each view as a 2D polygon. Such a discrete polygonal description of silhouettes induces a unique polyhedron representation of the visual hull, the structure of which is recovered by EPVH. To achieve this, three steps are performed. First, a particular subset of the polyhedron edges is computed: the viewing edges, which we describe below. Second, all other edges of the polyhedron mesh are recovered by a recursive series of geometric deductions. The positions of vertices not yet computed are gradually inferred from those already obtained, using the viewing edges as an initial set. Third, the mesh is consistently traversed to identify the faces of the polyhedron.

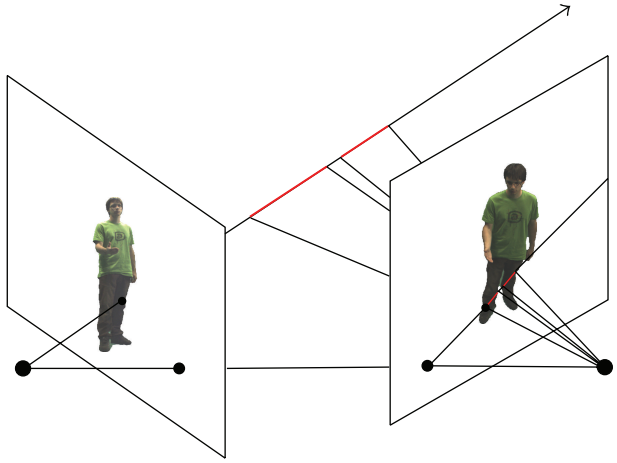


Figure 4: Viewing edges (in bold) along the viewing line.

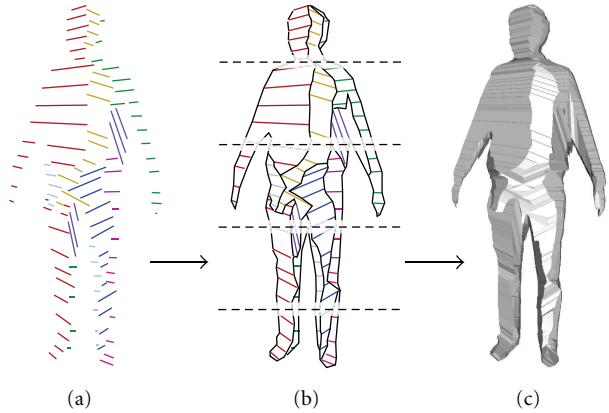


Figure 5: The three main steps of the EPVH algorithm, (a) viewing edge computation, (b) mesh connectivity (horizontal slices depict the partitioning used for the parallel version of the algorithm), and (c) face generation.

We first give an overview of the sequential algorithm. For more details refer to [14]. Then we explain how we distribute this algorithm in order to reach real-time performance.

3.1.1. Computing the Viewing Edges. *Viewing edges* are the edges of the visual hull induced by viewing lines of contour vertices, see Figure 4. There is one viewing line per silhouette 2D vertex. On each viewing line, EPVH identifies segments that project inside silhouettes in all other images. Each segment, called a viewing edge, is an edge of the visual hull and each segment extremity a 3D vertex. Each 3D vertex is trivalent, that is, the intersection point of 3 edges. Higher valence is neglected because it is highly unlikely in practice.

3.1.2. Computing the Visual Hull Mesh. After the first step, the visual hull is not yet complete. Some edges are missing to fulfill the mesh connectivity. Some vertices, called triple points, are also missing. A triple point is a vertex of the visual hull generated from the intersection of three planes defined by silhouette segments from three different images. EPVH completes the visual mesh by traveling along 3D edges as defined by two silhouette edges as long as these 3D edges project inside all silhouettes. At the limit, that is, when it projects on a silhouette contour, it identifies new triple points or recognize already computed visual hull vertices.

A last step consists in traversing the mesh to identify the polyhedron faces. 3D face contours are extracted by walking through the complete oriented mesh while always taking left turns at each vertex. Orientation data is inferred from silhouette orientations (counterclockwise oriented outer contours and clockwise oriented inner contours).

3.2. Distributed Algorithm

3.2.1. Algorithm. For real-time execution we developed a parallel version of the EPVH algorithm using a three-stage pipeline:

Stage 1: Viewing Edges. Let V be the number of thread—each thread being distributed on different CPUs across the cluster's hosts—in charge of computing the viewing edges. The silhouettes extracted by all image processing hosts are broadcasted to the V threads. Each thread computes locally the viewing edges for n/V viewing lines, where n is the total number of viewing lines (Figure 5(a)).

Stage 2: Mesh Connection. Let M be the number of thread in charge of computing the mesh. The V threads from the previous step broadcast the viewing edges to the M threads. Each thread is assigned a slice of the space (along the vertical axis as we are usually working with standing humans) where it computes the mesh. Slices are defined to have the same number of vertices. Each thread completes the connectivity of its submesh, creating triple points when required. The submeshes are then gathered on one host that merges the results, taking care of the connectivity on slice boundaries removing duplicate edges or adding missing triple points (Figure 5(b)).

Stage 3: Face Identification. The mesh is broadcasted to K threads in charge of face identification. Workload is balanced by evenly distributing the set of generator planes among processors (Figure 5(c)).

3.2.2. Evaluation. Consider an acquisition space surrounded by up to 8 cameras and with 1 or 2 persons. In that case, the algorithm reaches the cameras' refresh rates (tuned in between 20 and 30 frames per second) and ensures a latency below 100 milliseconds (including video acquisition and 2D-image processing) with between 4 and 8 processors. While the algorithm is flexible and allows for more processors, it did not prove to significantly increase the performance in this experimental context. More information and results about the parallelization of this algorithm can be found in [15].

Using a large number of cameras raises several issues. The algorithm complexity is quadratic in the number of

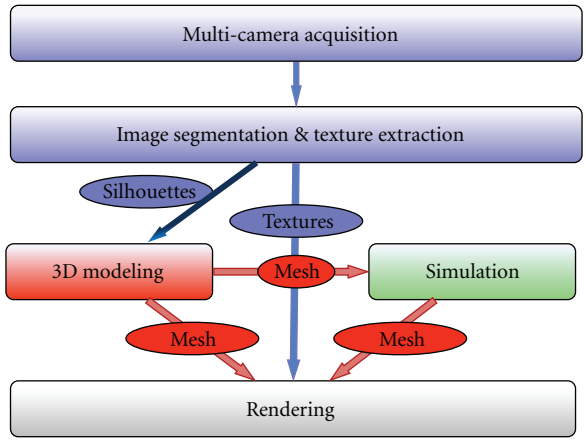


Figure 6: Application architecture coupling a multicamera acquisition space and a virtual physical environment.

cameras, quickly leading to non acceptable latencies. Today our efforts focus on using higher resolution cameras rather than significantly more cameras. The algorithm complexity is in $n\log(n)$, where n is the maximum number of segments per silhouette, making it more scalable on this parameter. Having more cameras makes sense for large acquisition spaces, where 3D models are computed per subsets of cameras.

3.3. Texture Extraction. We also extract from each silhouette the photometric data that will be used later during the rendering process for photorealistic rendering.

4. Interaction and Visualization

The 3D mesh, and the associated textures, acquired by the multicamera system are sent over the network to the visualization node and the physical simulation node that handle interactions (Figure 6). These tasks are detailed below.

4.1. Simulation

4.1.1. Collision Based Interactions. Coupling real-time 3D-modeling with a physical simulation enables interaction possibilities that are not symbolic and feel therefore natural to the user. Using the SOFA (http://www.sofa-framework.org/) framework, we developed a distributed simulator that handles collisions between soft or rigid virtual objects and the user's body. Unlike with most traditional interactive applications, it allows to use any part of the body or any accessories seen inside the acquisition space without being invasive. Some interactions are intricate, the prehension of objects, for example, is very difficult as there is no force information linked to the model.

4.1.2. SOFA. SOFA (simulation open framework application) is an open source framework primarily targeted at medical simulation research [36]. Its architecture relies on several innovative concepts, in particular the notion of multimodel representation. In SOFA, most simulation components, for instance, deformable models, collision models or instruments, can have several representations, connected together through a mechanism called mapping. Each representation is optimized for a particular task such as mechanical computations, collision detection or visualization.

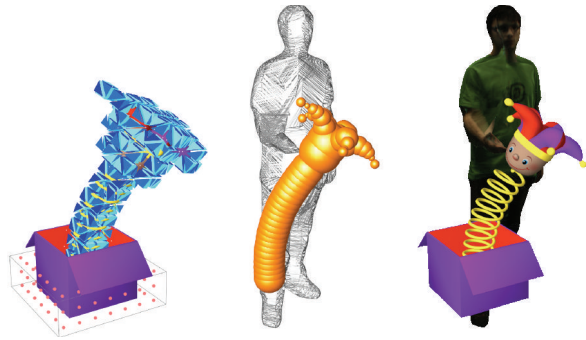
Integrating a SOFA simulation in our applications required adding a new component, receiving the stream of 3D meshes modeling the user and packaging it as an additional collision model (Figure 7). The triangulated polyhedron as computed by the 3D-modeling step can directly be used for collision detection. It is seen from the physics simulation point of view as a rigid mesh insensible to external forces (infinite mass), similar to predefined obstacles such as the floor, with the difference that it is changed each time a new mesh is received.

In order to obtain accurate interactions, the collision response additionally requires the speed and direction of motion at collision points, so that the user can push virtual objects, for example, kicking a ball, instead of only blocking them. We currently provide this information by querying the minimum distance of the current surface point to the previous modeled mesh. This is efficiently implemented by reusing the proximity-based collision detection components in SOFA. The computed distance is an estimation of the user's motion perpendicular to the surface, which is enough to give the colliding objects the right impulsion. However tangential frictions cannot be captured. The different parameters, like mass, spring stiffness, of the simulated scene are empirically tuned based on a trade-off between real-time constraints and a visibly plausible behavior.

Another key aspect of SOFA is the use of a scene-graph to organize and process the components while clearly separating the computation tasks for their possibly parallel scheduling. This data structure, inspired by classical rendering scene-graphs like OpenSG, is new in physically-based animation. Physical actions such as force accumulation or state vector operations are implemented as traversal actions. This creates a powerful framework for differential equation solvers suitable for single objects as well as complex systems made of different kinds of interacting physical bodies: rigid bodies, deformable solids or fluids.

We use an iterative implicit time integration solver. The maximum number of iterations is tuned to limit the computation time. This creates a trade-off between accuracy and computation time that allows us to reach the real-time constraint without sacrificing stability. Parallel versions of SOFA on multicore processors [37] and on GPU have been developed, allowing to interactively simulate rich environments.

4.2. Rendering. Data to be rendered, either provided by the simulation software, a static scene loader, or from the 3D-modeling algorithm, are distributed to dedicated rendering nodes. The rendering can be performed on heterogeneous display devices such as standard monitors, multiprojector walls, head-mounted displays or stereoscopic displays.



(a) (b) (c)

Figure 7: An interactive deformable object, (a) collides with the 3D-reconstructed mesh, (b) allowing interactions with the user in (c).

4.2.1. FlowVR Render. To distribute efficiently the rendering part, we use FlowVR Render [38]. Existing parallel or remote rendering solutions rely on communicating pixels, OpenGL commands, scene-graph changes or application-specific data. We rely on an intermediate solution based on a set of independent graphics primitives that use hardware shaders to specify their visual appearance. Compared to an OpenGL based approach, it reduces the complexity of the model by eliminating most fixed function parameters while giving access to the latest functionalities of graphics cards. It also suppresses the OpenGL state machine that creates data dependencies making primitive reordering and multistream combining difficult.

Using a retained-mode communication protocol transmitting changes between each frame, combined with the possibility to use shaders to implement interactive data processing operations instead of sending final colors and geometry, we are able to optimize the network load. High-level information such as bounding volumes is used to set up advanced schemes where primitives are issued in parallel, routed according to their visibility, merged and reordered when received for rendering. Different optimization algorithms can be efficiently implemented, saving network bandwidth or reducing texture switches for instance.

4.2.2. 3D Model and Texture Mapping. Rendering the 3D model is quite simple as it is already a polygonal surface. To apply the textures extracted from the silhouettes, we use a shader that projects the mesh vertices in source images consistently with camera calibration parameters. The exact polyhedral visual hull algorithm guarantees that the 3D model can be projected back to the original silhouette with minimal error, a property that leads to a better quality texture mapping. Taking into account the surface normal, viewing direction, as well as self-occlusions, the pixel shader smoothly combines the contributions from the different cameras. Having access to the full 3D surface enables interactive and unconstrained selection of rendering viewpoints, and yields realistic views of the reconstructed person.

5. Platform Integration

Coupling the different software components involved into this project and distributing them on the nodes of a PC cluster for reaching real-time executions is performed through the FlowVR (http://flowvr.sourceforge.net/) middleware [21, 23], a middleware we developed conjointly with the Grimage project.

FlowVR enforces a modular programming that leverages software engineering issues while enabling high performance executions on distributed and parallel architectures. FlowVR relies on a dataflow and component-oriented programming approach that has been successfully used for other scientific visualization tools. Developing a FlowVR application is a two-step process. First, modules are developed. Modules are endless loops consuming data on input ports at each iteration and producing new data on output ports. They encapsulate a piece of code, imported from an existing application or developed from scratch. The code can be multithreaded or parallel, as FlowVR supports parallel code coupling. In a second step, modules are mapped on the target architecture and assembled into a network to define how data are exchanged. This network can make use of advanced features, from bounding-box-based routing operations to complex message filtering or synchronization operations.

The FlowVR runtime engine runs a daemon on each node of the cluster. This daemon is in charge of synchronization and data exchange between modules. It hides all networking aspects to modules, making module development easier. Each daemon manages a shared memory segment. Messages handled by modules are directly written and read from this memory segment. If data exchange is local to a node, it only consists in a pointer exchange, while the daemon takes care of transferring data through the network for internode communications.

The largest FlowVR applications are composed of thousand of modules and connections. To be able to deal with the network design of such applications, FlowVR is based on hierarchical component model [22]. This model introduces a new kind of components called composite. A composite is designed by assembling other FlowVR components. This

hierarchy enables to create a set of efficient and reusable patterns or skeletons, for example, one-to-many broadcast or scatter collective communications are encapsulated into communication tree patterns made generic and parametric to be used in various contexts. A compilation step instantiates skeletons parameters to fit to the target architecture, for example, in case of communication trees, parameters to be set are the tree arity and the mapping of each node. Using description files or parameters, the compilation step unfolds the hierarchical description and produces a flat FlowVR network optimized for the target architecture.

The Grimage network (Figure 8) is a thousand modules and connections application developed by assembling this set of skeletons. The network relies on several communications patterns, for example, in the acquisition component, a pipeline is associated to each camera. The 3D reconstruction algorithm needs a strong coherency between these pipelines. To reach real-time execution, application needs sometimes to discard a metaframe because it will not be able to compute it under real-time constraints. Therefore a pattern is in charge to do this sampling and keep the coherency. This pattern synchronizes all pipelines and discards the metaframe in the distributed context. The FlowVR compilation process enforces the modularity of the application. The hierarchical description of Grimage is totally independent from the acquisition set up and the target architecture. A file describing the architecture and the acquisition set up is used to compile the network. The compilation process will create the appropriate number of pipelines or reconstruction parallel processes based on the architecture description file. Therefore, in case of set up modification (add a stereoscopic display, integration of a new SMP node in the PC cluster or modification of the number of cameras), the only change needed is to update the architecture description file. This modularity is critical to the Grimage application that has been developed over several years by various persons. FlowVR is a key component that made it possible to aggregate and efficiently execute on a PC cluster the various pieces of code involved. The level of modularity achieved significantly eases the maintenance and enhancements of the application.

6. Collaborative Environment

Virtual environments, such as multiplayer games or social network worlds, need a representation of each user that is often an avatar controlled with a keyboard and a mouse. In contrast, our system virtualizes the user into a model that has the user's geometry and appearance at any instant, hence relaxing the need for control devices and enabling new types of interactions with virtual worlds as discussed below.

6.1. Visual Presence. From the user's point of view, the sense of presence is drastically improved with an avatar that has the user's shape and aspect instead of those of a purely synthetic avatar taken from a 3D model database. In addition, the avatar is moving with respect to the user' body gestures and not to a preprogrammed set of actions. Different users can therefore recognize themselves and have life-like conversations. Also emotions can be communicated through face expressions and body gestures.

6.2. Mechanical Presence. Sharing our appearance is not the only advantage of our environment. 3D meshes can also be used to interact with shared virtual objects. The server managing the virtual environment receives user information (geometric 3D models, semantic actions, etc.), runs the simulation and sends back the transformation of the virtual scene to the users (Figure 9). The dynamic deformable objects are handled by this server while heavy static scenes can be loaded at initialization on each user's rendering node. Such an environment can be used by multiple users to interact together from different locations with the same virtual objects. For each iterative update the physical simulation detects collisions and computes the effect of each user interaction on the virtual world. It is of course impossible to change the state of the input models themselves as there are no force-feedback devices on our platforms. Physically simulated interactions between participants are also impossible for the same reason.

6.3. Remote Presence. Remote site visualization of models requires the transfer of 3D model streams and their associated textures under the constraints of limited bandwidth and minimal latency. The mesh itself is not bandwidth intensive and can be easily broadcasted over the network. The textures, one per camera in our current implementation, induce much larger transfers and represent the bulk of the data load. We will provide data bandwidth measurements in Section 7 for a particular set up. We do not consider any specific transfer protocol, which is beyond the scope of this work.

The FlowVR middleware handles the synchronization of both texture and mesh streams to deliver consistent geometric and photometric data, that is, the texture stream gathered from the acquisition nodes must be rendered at the same time as the 3D model reconstructed with this same image stream, otherwise it would lead to visual artifacts. It also prevents network congestion by resampling the streams (discarding some 3D metaframe) in order to send only up-to-date data to the end-user nodes. As the physical simulation only needs the mesh, each site sends it only the meshes as soon as available. We did not experience incoherency issues requiring to enforce a strong time synchronization between meshes.

7. Experiments

We report below on preliminary experiments that were conducted with two platforms located in the same room.

7.1. Practical Set up. The first acquisition platform is built with 8 firewire cameras with 1 MP resolution, allowing an acquisition space of 2 by 2 meters, suitable for a full person. The PC cluster used is composed of 10 dual xeon PCs connected through a gigabit Ethernet network.

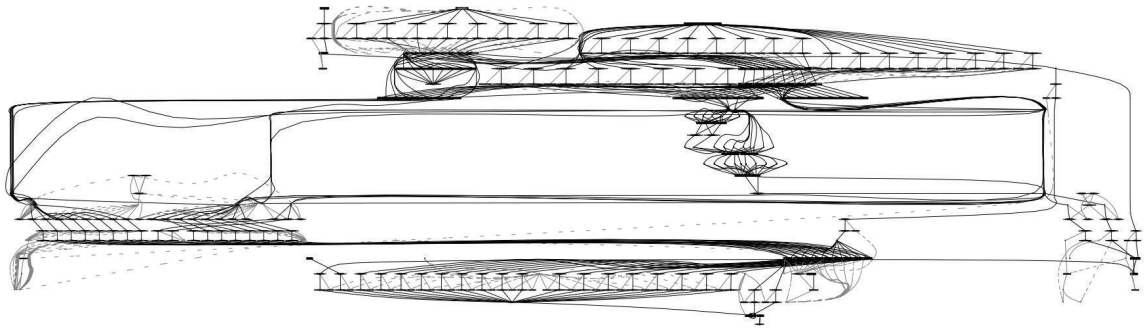


Figure 8: The FlowVR flat network of the Grimage application. Nodes are modules and edges communication channels. This network is compiled for 8 cameras and EPVH parallelized on 6 CPUs.

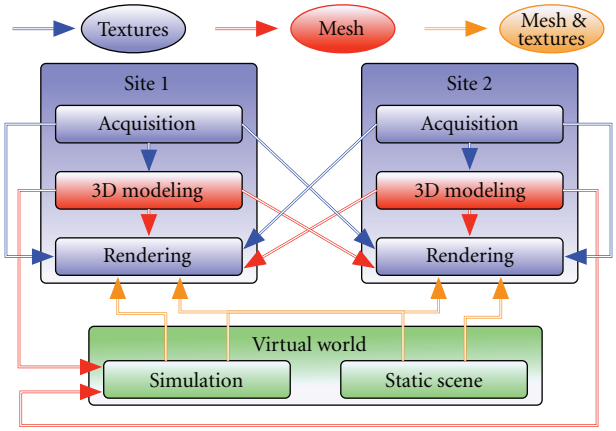


Figure 9: Application architecture for two multicamera acquisition spaces and a virtual physical environment.

The second acquisition platform is a portable version of the first one with an acquisition space of 1 square meter at table height used for demonstration purpose. It uses 6 firewire cameras and is suitable for hand/arm interactions. The cluster is built with 6 mini-PCs used for camera acquisition, 1 dual xeon server for computation, and a laptop for supervision. This platform was presented at Siggraph in 2007 [36].

The two platforms are connected by a gigabit Ethernet network using one PC as gateway between the two platforms. This PC gathers the data from the two platforms and handles the physical simulation.

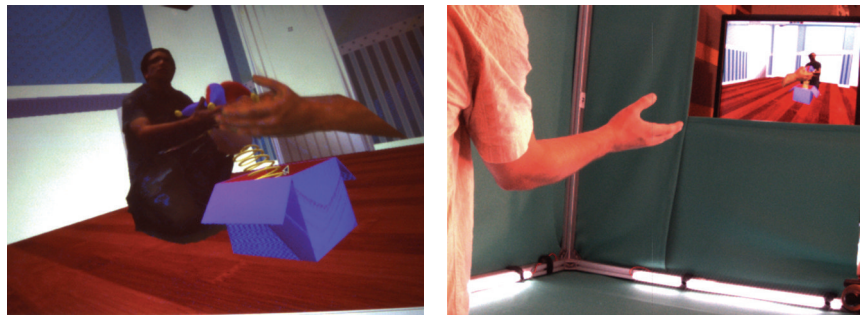
7.2. Data Estimation. Our 8-camera platform produces 1 MP images, yielding 3 MB images and thus a theoretical 24 MB multiimage frame throughput. In practice the only image data needed for texturing lies inside the silhouettes, which we use to reduce transfer sizes. When one user is inside the acquisition space the silhouettes occupy usually less than 20% of the overall image in a full-body set up. Thus an average multitexture frame takes 4.8 MB. We also need to send the silhouette mask to decode the texture. A multisilhouette mask frame takes about 1 MB. The overall estimated stream is about 5.8 MB. To decrease the needed bandwidth we decided to use the full resolution of the camera for 3D model computation but only half the resolution for texture mapping, reducing the full multitexture frame to a maximum of 1.45 MB to transfer at each iteration. The mesh itself represents less than 80 KB (about 10000 triangles).

Running at 20 frames per second, which is reasonable for good interactions, the dual platform requires a 29 MB/second bandwidth for 3D frame streaming which is easily scalable to a Gigabit Ethernet network (120 MB/s).

7.3. Results. We are able to acquire images and to generate the 3D meshes at 20 fps on each platform. The simulation and the rendering processes are running respectively at 50–60 fps and 50–100 fps, depending of the load of the system. As they run asynchronously from the 3D model and texture generation we need to resample the mesh and the texture streams independently. In practice the mesh and texture transfer between sites oscillates between 15 fps and 20 fps, depending on the size of the silhouette inside the images. Meanwhile the transfer between the 3D-modeling and the rendering node inside a platform and the transfer going to the simulation node are always running at 20 fps. We do not experience any extra connection latency between the two platforms. During execution, the application does not overload the gigabit link.

The accuracy of the model obtained using EPVH is satisfactory both for visual experience and for physical simulation precision. The level of detail of the model is good enough to distinguish the user's fingers. Our application is robust to input noise, the obtained 3D model is watertight (no holes) and manifold (no self intersection). It is also robust to network load change as the data transmitted could be resampled to avoid latency.

We did not conduct any user study about the sense of presence achieved through this system (Figure 10). However the numerous users that experienced the system, in particular during the Emerging Technology show at Siggraph 2007 [36], were generally impressed by the quality of the visual and mechanical presence achieved without requiring handheld devices, markers or per-user calibration steps. Interaction was intuitive, often requiring no explanation, as it was direct,



(a) (b)

Figure 10: (a) the 3D virtual environment with a "full-body" user and a "hand" user, interacting together with a virtual puppet, and (b) the "hand-size" acquisition platform.

full-body and relying on physical paradigms that somehow mimicked a common real world experience. This positive feedback was achieved despite the non immersive display used (a 2D display located 75 cm in front of the user) and the third-person visualization. Future work will focus on associating Grimage and an immersive visualization environment such as a HMD to enable first-person visualization and allow for better depth perception.

A similar experiment was showcased at VRST 2008 in Bordeaux [39], involving two "hand-size" platforms in the same room. Visitors could see each other's hands immersed in the same virtual room. A virtual puppet was animated by the physics simulation. Each user could push or grab the puppet. They could also try to collaborate for instance to grab the puppet using two hands, one from each user. No direct user-to-user physical interaction was possible. The meshes would simply intersect each other when both hand positions superpose in the virtual world. The videos (http://grimage.inrialpes.fr/telepresence/) of our experiments give a good overview of the sense of presence achieved.

8. Conclusion

We presented in this article the full Grimage 3D-modeling system. It adopts a software component-oriented approach offering a high-level of flexibility to upgrade part of the application or reuse existing components in different contexts. The execution environment supports the distribution of these components on the different nodes of a PC cluster or Grid. We can thus harness distributed I/O and computing resources to reach interactive execution times but also to build multiplatform applications. 3D-modeling relies on the EPVH algorithm that computes from 2D-images a 3D mesh corresponding to the visual hull of the observed scene. We also retrieve photometric data further used for texturing the 3D mesh. Experiments show that Grimage is suitable for enforcing the visual and mechanical presence of the modeled users.

The actual state of development shows some limitations. For instance we do not extract a complete velocity field on the mesh surface, our algorithm only provide an estimation of the normal velocity and does not provide any tangential velocity. This lack of data limits the range of possible mechanical interactions. As a consequence, the user can modulate the force it applies to a given virtual object but has difficulties to keep an object on his hand or to grab anything. The first steps of the vision pipeline are crucial for the accuracy of the final 3D model. We experienced that a higher quality background subtraction could significantly improve the 3D mesh. We are working on advanced background subtraction algorithms using fine-grain parallel processing to keep the computation time low.

This paper includes a preliminary experiment with a dual platform. We are today conducting telepresence and collaboration experiments between distant sites, each one having its own multicamera environment. This context will require further optimizations to control the amount of data exchanged between sites to keep an acceptable latency.

Acknowledgment

This work was partly funded by Agence Nationale de la Recherche, contract ANR-06-MDCA-003.

References

[1] P. J. Narayanan, P. W. Rander, and T. Kanade, "Constructing virtual wolrds using dense stereo," in *Proceedings of the 6th International Conference on Computer Vision*, pp. 3–10, Bombay, India, 1998.

[2] M. Gross, S. Würmlin, M. Naef, et al., "Blue-c: a spatially immersive display and 3D video portal for telepresence," *ACM Transactions on Graphics*, vol. 22, no. 3, pp. 819–827, 2003.

[3] W. Matusik and H. Pfister, "3D TV: a scalable system for real-time acquisition, transmission, and autostereoscopic display of dynamic scenes," *ACM Transactions on Graphics*, vol. 23, no. 3, pp. 814–824, 2004.

[4] G. Kurillo, R. Bajcsy, K. Nahrsted, and O. Kreylos, "Immersive 3D environment for remote collaboration and training of physical activities," in *Proceedings of IEEE Virtual Reality Conference*, pp. 269–270, 2008.

[5] L. Gharai, C. S. Perkins, R. Riley, and A. Mankin, "Large scale video conferencing: a digital amphitheater," in *Proceedings of*

the 8th International Conference on Distributed Multimedia Systems, San Francisco, Calif, USA, September 2002.

[6] H. Baker, D. Tanguay, I. Sobel, et al., "The coliseum immersive teleconferencing system," in *Proceedings of the International Workshop on Immersive Telepresence*, Juan Les Pins, France, December 2002.

[7] D. Nguyen and J. Canny, "MultiView: spatially faithful group video conferencing," in *Proceedings of of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 799–808, 2005.

[8] Philips 3D Solutions, "WoWvx technology".

[9] E. Chen and L. Williams, "View interpolation for image synthesis," in *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '93)*, pp. 279–288, Anaheim, Calif, USA, 1993.

[10] C. L. Zitnick, S. B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski, "High-quality video view interpolation using a layered representation," in *Proceedings of International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '04)*, pp. 600–608, 2004.

[11] J. Mulligan and K. Daniilidis, "Real time trinocular stereo for tele-immersion," in *Proceedings of IEEE International Conference on Image Processing*, vol. 3, pp. 959–962, 2001.

[12] P. Kauff and O. Schreer, "An immersive 3D videoconferencing system using shared virtual team user environments," in *Proceedings of International Conference on Collaborative Virtual Environments*, pp. 105–112, 2002.

[13] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International Journal of Computer Vision*, vol. 47, no. 1–3, pp. 7–42, 2002.

[14] J.-S. Franco and E. Boyer, "Efficient polyhedral modeling from silhouettes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 3, pp. 414–427, 2009.

[15] J.-S. Franco, C. Ménier, E. Boyer, and B. Raffin, "A distributed approach for real time 3D modeling," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshop (CVPRW '04)*, p. 31, June 2004.

[16] J. Allard, E. Boyer, J.-S. Franco, C. Ménier, and B. Raffin, "Marker-less real time 3D modeling for virtual reality," in *Immersive Projection Technology*, 2004.

[17] J. Allard, J.-S. Franco, C. Ménier, E. Boyer, and B. Raffin, "The GrImage platform: a mixed reality environment for interactions," in *Proceedings of the 4th IEEE International Conference on Computer Vision Systems (ICVS '06)*, pp. 46–52, 2006.

[18] J. Allard, C. Ménier, B. Raffin, E. Boyer, and F. Faure, "GrImage: markerless 3D interactions," in *Proceedings of the International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '07)*, San Diego, Calif, USA, 2007.

[19] J. Allard, C. Ménier, E. Boyer, and B. Raffin, "Running large VR applications on a PC cluster: the FlowVR experience," in *Immersive Projection Technology*, 2005.

[20] J.-S. Franco and E. Boyer, "Exact polyhedral visual hulls," in *Proceedings of the British Machine Vision Conference*, vol. 1, pp. 329–338, 2003.

[21] J. Allard, V. Gouranton, L. Lecointre, et al., "FlowVR: a middleware for large scale virtual reality applications," in *Euro-Par Parallel Processing*, vol. 3149 of *Lecture Notes in Computer Science*, pp. 497–505, Springer, Berlin, Germany, 2004.

[22] J.-D. Lesage and B. Raffin, "A hierarchical component model for large parallel interactive applications," *The Journal of Supercomputing*, vol. 7, no. 1, pp. 67–80, 2008.

[23] J.-D. Lesage and B. Raffin, "High performance interactive computing with FlowVR," in *Proceedings of IEEE Virtual Reality SEARIS Workshop*, pp. 13–16, 2008.

[24] S. N. Sinha and M. Pollefeys, "Synchronization and calibration of camera networks from silhouettes," in *Proceedings of the 17th International Conference on Pattern Recognition (ICPR '04)*, vol. 1, pp. 116–119, Cambridge, UK, August 2004.

[25] E. Boyer, "On using silhouettes for camera calibration," in *Proceedings of the 7th Asian Conference on Computer Vision (ACCV '06)*, vol. 3851 of *Lecture Notes in Computer Science*, pp. 1–10, Hyderabad, India, January 2006.

[26] A. Hilton and J. Mitchelson, "Wand-based multiple camera studio calibration," Tech. Rep. VSSP-TR-2, CVSSP, 2003.

[27] Z. Zhang, "Camera calibration with one-dimensional objects," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 7, pp. 892–899, 2004.

[28] T. Horprasert, D. Harwood, and L. S. Davis, "A statistical approach for real-time robust background subtraction and shadow detection," in *Proceedings of the 7th IEEE International Conference on Computer Vision (ICCV '99)*, vol. 99, pp. 1–19, Kerkyra, Greece, September 1999.

[29] G. K. M. Cheung, T. Kanade, J.-Y. Bouguet, and M. Holler, "Real time system for robust 3D voxel reconstruction of human motions," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2, pp. 714–720, 2000.

[30] I. Debled-Rennesson, S. Tabbone, and L. Wendling, "Fast polygonal approximation of digital curves," in *Proceedings of the International Conference on Pattern Recognition*, vol. 1, pp. 465–468, 2004.

[31] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski, "A comparison and evaluation of multi-view stereo reconstruction algorithms," in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '06)*, vol. 1, pp. 519–526, New York, NY, USA, June 2006.

[32] D. Vlasic, I. Baran, W. Matusik, and J. Popović, "Articulated mesh animation from multi-view silhouettes," *ACM Transactions on Graphics*, vol. 27, no. 3, article 97, 2008.

[33] J. Gall, C. Stoll, E. de Aguiar, C. Theobalt, B. Rosenhahn, and H.-P. Seidel, "Motion capture using joint skeleton tracking and surface estimation," in *Proceedings of International Conference on Computer Vision and Pattern Recognition (CVPR '09)*, Miami, Fla, USA, June 2009.

[34] A. Laurentini, "The visual hull concept for silhouette-based image understanding," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 2, pp. 150–162, 1994.

[35] S. Lazebnik, E. Boyer, and J. Ponce, "On how to compute exact visual hulls of object bounded by smooth surfaces," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, pp. 156–161, 2001.

[36] J. Allard, S. Cotin, F. Faure, et al., "SOFA—an open source framework for medical simulation," in *Medicine Meets Virtual Reality*, pp. 1–6, 2007.

[37] E. Hermann, B. Raffin, and F. Faure, "Interactive physical simulation on multicore architectures," in *Proceedings of Symposium on Parallel Graphics and Visualization (EGPGV '09)*, pp. 1–8, Munich, Germany, March 2009.

[38] J. Allard and B. Raffin, "A shader-based parallel rendering framework," in *Proceedings of the IEEE Visualization Conference*, pp. 127–134, 2005.

[39] B. Petit, J.-D. Lesage, J.-S. Franco, E. Boyer, and B. Raffin, "Grimage: 3D modeling for remote collaboration and telepresence," in *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST '08)*, pp. 299–300, 2008.