The GrImage Platform: A Mixed Reality Environment for Interactions

Abstract. In this paper, we present a scalable architecture to compute, visualize and interact with 3D dynamic models of real scenes. This architecture is designed for mixed reality applications requiring such dynamic models, tele-immersion for instance. Our system is composed of 3 main parts: the acquisition, based on standard firewire cameras and using a recent shape-from-silhouette algorithm which leads to optimally precise 3D models; the computation, based on a distribution scheme over a cluster of PC; the visualization, which is achieved on a multiple display wall. The proposed distribution scheme ensures scalability of the system and allows hereby control over the number of cameras used for acquisition, the frame-rate, or the number of projectors used for high resolution visualization. To our knowledge this is the first completely scalable vision architecture for real time 3D modeling, from acquisition to visualization through computation. Experimental results show that this framework is very promising for real time 3D interactions.

1 Introduction

Interactive and mixed reality environments generally rely on the ability to retrieve 3D information about users, in real time, in an interaction space. Such information is used to make real and virtual worlds consistent with one another. Traditional solutions to this problem usually consist in tracking positions of sensors by means of various technologies including electromagnetic waves, infrared or accelerometers. However, this requires users to wear invasive equipments and usually specific body suits. Furthermore it does not lead to a shape description, as required for many applications such as tele-immersion for example. In this paper, we consider a more flexible class of methods based on digital cameras. These methods can compute 3D shape models in real-time, and without any markers or specific equipments. We propose a framework in this context, from acquisition to visualization and interactions. Our objective is to provide a flexible solution which especially focus on issues critical in such systems: precision of the 3D model, precision of the visualization and process speed.

Several multi-camera systems for dynamic modeling have been proposed. Stereo based systems were first proposed [16] for *virtualization*, but most recent systems use image silhouettes as input data to compute 3D shapes. They can be classified according to the fact that they work offline or in real-time, and also by the type of 3D models which they build. Offline systems allow complex and precise models to be built [7,6], in particular articulated models, however they do not allow real-time interaction as intended in this work. Most real-time systems,

such as [8,10], that have been proposed in the past, compute voxel models, i.e. discrete 3D models made of elementary parallelepipedic cells. Interestingly, several systems in this category [5,12,1,18] use a distribution scheme over a PC cluster to speed up computations and hence, provide some kind of control over the model precision and the process speed. However, voxel based methods are still imprecise unless a huge number of voxels is used. Furthermore they require post-processing, typically a marching cube approach, to produce surface shapes. This is computationally expensive, and generates very small-scale geometry whenever precision is required.

Another class of real time, but non-parallel, approaches directly render new viewpoint images [17] using possibly graphic cards for computations[14]. Based on the *Image Based Visual Hull* method [15], these approaches efficiently focus on the desired 2D image, but they still rely on a single PC for computations, limiting the number of video-streams or the frame-rate, and they do not provide explicit 3D shapes as required by many applications.

In contrast to the aforementioned systems, ours directly computes watertight and manifold surface models. These surface models are exact with respect to the input silhouette information available and, as such, are optimal and equivalent to voxel grids with infinite resolutions. A particular emphasis has been put on the system scalability to ensure flexibility and to adress performance and hardware cost efficiency issues. To this aim, the system is composed of multiple commodity components: FireWire cameras distributed on multiple PCs interconnected through a standard Ethernet network, as well as multiple projectors for a wall display. To reach real time performance, a careful distribution of the work load on the different resources is achieved. For that purpose we rely on a middleware library [2], dedicated to the distribution of interactive applications.

Section 2 outlines the global approach. Section 3 discusses issues related to image acquisition. The 3D modeling algorithm and its parallel implementation is then explained in section 4. In section 5, interactions and visualization is described. Section 6 details the distributed framework for our system. Section 7 presents some experimental results before concluding in section 8.

2 Outline

Fig. 1. From multi-camera videos to dynamic textured 3D models

Our goal is to compute 3D shapes of users in an acquisition space surrounded by several cameras in real time (see figure 1). Such models are subsequently used for interaction purposes, including display. In order to achieve this, several processes must be coupled.

Acquisition Fixed cameras are set to surround the scene. Their calibration is obtained offline through off-the-shelf libraries such as OpenCV. Each camera is handled by a dedicated PC. Each acquired image is locally analyzed to extract regions of interest (the foreground) which are then vectorized, i.e. their delimiting polygonal contours are computed.

3D modeling A geometric model is then computed from the silhouettes using an efficient method to compute the *visual hull* [13]. Obtained visual hull polyhedrons are sufficient for numerous VR applications including collision detection or virtual shadow computation for instance. To reach a real time execution, their computation is distributed among different processors.

Interactions and Visualization The 3D mesh is asynchronously sent to the interaction engines and to the visualization PCs. Multi-projector rendering is handled by a mixed replicated/sort-first approach.

3 Acquisition

Acquisition takes place on a dedicated set of PCs, each connected to a single camera. These PCs perform all necessary preliminary image processing steps: color image acquisition, background subtraction and silhouette polygonalization (see figure 2). All cameras are standard firewire cameras, capturing images at 30 fps with a resolution of 780x580 in YUV color space.

3.1 Synchronization

Dealing with multiple input devices raises the problem of data synchronization. Indeed, our applications rely on the assumption that the input data chunks received from different sources are coherent, i.e. that they relate to the same scene event. We use an hardware synchronization where image acquisition is triggered by externally gen-locking the cameras, ensuring a delay between images below $100\mu s$.

3.2 Background Subtraction

Regions of interest in the images, i.e. the foreground or silhouette, are extracted using a background subtraction process. As most of the existing techniques [11,8], 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.* [11] for shadow detection. A crucial remark here is that the quality 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 ourself to such specific environments in our setup.

Fig. 2. The different steps in the acquisition process: (a) the original image; (b) the binary image of the silhouette; (c) the exact silhouette polygon (250 vertices); (d) a simplified silhouette polygon (55 vertices).

3.3 Silhouette Polygonalization

Since our modeling algorithm computes a surface and not a volume, it does not use image regions as defined by silhouettes, but their delimiting polygonal contours. We extract such silhouette contours and vectorize them using the method of Debled *et al.* [9]. Each contour is decomposed into an oriented polygon, which approximates the contour to a given approximation bound. With a pixel bound, obtained polygons are strictly equivalent to the silhouettes in the discrete sense (see figure 2-c). However in case of noisy silhouettes this leads to numerous small segments. A higher approximation bounds results in significantly fewer segments (see figure 2-d). This enables to control the model complexity, and therefore the computation time, in an efficient way.

4 3D Modeling

The visual hull is a well studied geometric shape [13] which is obtained from a scene object's silhouettes observed in n views. It is the maximum shape consistent with all silhouettes. As such, it can be seen as the intersection of the images' *viewing cones*, the volumes that backproject from each view's silhouette (see figure 3).

Fig. 3. Visual hull of a sphere with 3 views.

We use a distributed surfacic method we have developed [3]. It recovers the exact polyhedral visual hull from the input silhouette polygons in three steps. First, a subset of the polyhedron edges – the viewing edges – is computed. Second, starting from this partial description of the polyhedron's mesh, all other edges and vertices are recovered by a recursive series of geometric deductions. Third, the shape's faces are recovered by traversing the obtained mesh. The following paragraphs briefly detail these steps, and their distribution over p CPUs.

4.1 Computing the Viewing Edges

Viewing edges are intervals along viewing lines associated from silhouette contours' vertices. They are obtained by computing the set of intervals along such

a viewing line that project inside all silhouettes. The distribution of this computation uses the fact that each viewing line's contributions can be computed independently. Viewing lines are partitioned into p identical cardinality sets and each batch is distributed to a different CPU. Final set is obtained by gathering partial results.

4.2 Computing the Visual Hull Mesh

The viewing edges give us an initial subset of the visual hull geometry. The missing chains of edges, are then recovered recursively starting from the viewing edges set. To allow concurrent task execution, the 3D space is partitionned into p slices. Slice width is adjusted by attributing a constant number of viewing edge vertices per slice for workload balancing. Each CPU computes the missing edges in its assigned slice. Partial meshes are then gathered and carefully merged across slice borders.

4.3 Computing the Faces

Faces of the 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 contours. Each CPU independently computes a subset of the face information, the complete mesh being previously broadcasted to each CPU.

5 Interactions and Visualization

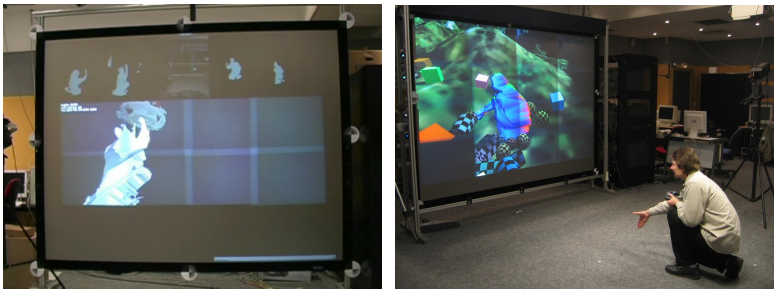
5.1 Real-Time interactions

We experimented two different interactions. The first one consists in a simple object carving (see figure 4(a)). The user can sculpt an object using any part of his body. This is done with octree-based boolean operations to update the object where it intersects with the user's model. Update operations include removal, addition of matter and change in sculpture color. The object can be rotated to simulate a potter's wheel.

The second interaction results from the integration of the user's model inside a rigid body simulation (see figure 4(b)). Several dynamic objects where added in the scene, and the system handles collisions with the user's body. This interaction requires all available information about the user's 3D surface, which is not available using classical tracking methods. Using our surface modeling approach, such fine level collision detection is something our system can achieve.

5.2 Multi-projector Visualization

To provide the user with a wide field of view while preserving image detail, as necessary in semi-immersive and immersive applications, we have chosen to use a multi-projector display. The most scalable approach to implement this setup



(a) (b)

Fig. 4. Interaction experiments: (a) Carving, (b) Collision.

is to use one PC to drive each projector. To obtain a coherent image, each PC will have to synchronously render the same scene with a different view point, corresponding to the position of the related projector.

Several methods are available to implement parallel visualization, depending on the level of the primitives exchanged. We use a new framework [4], allowing to use a different scheme for each part of the scene. Large static objects, such as the landscape, use a replicated scheme so that they are sent locally on each host. Other objects, such as the reconstructed mesh, are created on specific hosts and then sent to all visualization hosts, possibly culling invisible data (sort-first scheme).

The rendering of the 3D mesh itself is quite simple as it is already a polygonal surface. We can optionally compute averaged normal vectors at each vertex to produce a smoothly shaded rendering. It is relatively small (approximately 10000 triangles) so it can be broadcasted to all visualization hosts.

6 Implementation

6.1 The middleware library

To provide the I/O and computing power necessary to run our applications in real time, we use a PC cluster. However, coupling all pieces of code involved, distributing them on the PCs and insuring data transfers can be cumbersome. To get a high performance and modular application, we use a tool we developed [2] to manage distributed interactive applications. It relies on an data-flow model. Computation and I/O tasks are encapsulated into modules. Each module endlessly iterates, consuming and producing data. Modules are not aware of the existence of other modules. A module only exchanges data with the middleware daemon that runs on the same host. The set of daemons running on a PC cluster are in charge of implementing the data exchange network that connects modules. Daemons use TCP connections for network communications or shared-memory for local communications. The middleware network defined between modules

can implement simple module-to-module connections as well as complex message handling operations like synchronizations, data filtering operations, data sampling, broadcasts, etc. This fine control over data handling enables to take advantage of both the specificity of the application and the underlying cluster architecture to optimize the latency and refresh rates.

6.2 Data-flow Graph

We propose for our application the following distributed data-flow graph from acquisition to rendering (see figure 5).

Fig. 5. Data-flow graph from 4 cameras acquisitions to 4 video projectors rendering.

Each dedicated acquisition PC locally performs the data acquisition to obtain the silhouettes which are then broadcasted to the PCs in charge of the first modeling step, the viewing edge computation step. Follows the two other modeling steps, the global mesh recovery and the surface extraction. The resulting reconstructed surface is broadcasted to the PCs in charge of interaction computation and to the visualization hosts. These PCs also receive data from the interaction modules of the VR environment.

To obtain good performance and scalability it is necessary to setup specific coupling policies between the different parts of the application so they can run at different frequencies. The acquisition part typically runs at the frequency of the cameras while interactions run at more than $100Hz$. The visualization stage runs independently, allowing to change the viewpoint without waiting for the computation of the next 3D model. To implement these coupling policies we use two dataflow control policies: *FIFO* connections between modules running at the same frequency and *greedy sampling* connections (receivers always use last available data) between modules running asynchronously.

7 Results

We present the results obtained with our platform. It gathers 11 dual-Xeon 2.6 GHz PCs and 16 dual-Opteron PCs connected together by a gigabit Ethernet network. 6 FireWire Cameras are connected to the dual-Xeon machines. 16 projectors are connected to the dual-Opteron machines through NVIDIA 6800 Ultra graphics cards. The projectors display images on a flat screen of 2.7×2 meters. The acquisition space where the cameras are focused is located 1 meter away from the screen. Submitted videos show user interaction results.

To evaluate the potential of 3D modeling for interaction purposes, we identified the following classical criteria as being relevant:

- Latency: it is the delay between a user's action and the perception of this action on the displayed 3D model. It is the most important criterion. A large latency can significantly impair the interaction experience. For most experiments on our system the overall latency, including all stages from video acquisition to visualization, was around $100ms$. This can be noticed by the user but is small enough to maintain a high level of interactivity. The quality of the background subtraction step as well as the simplification threshold applied to the resulting contours have a high impact on the latency as they determine the computational cost of the 3D modeling.
- Update frequency (modeling framerate): in our experiments, using 4 CPUs was enough to provide an update frequency of 30 Hz with 6 cameras when one user was in the interaction space.
- Quality (model's level of detail): in our experiments, the user was able to use its hands to carve virtual objects, and, depending on the angle relative to the cameras, it was possible to distinguish his fingers.
- Robustness to acquisition noise: our modeling algorithm is exact with respect to provided input silhouettes however noisy. The resulting 3D model is always watertight (no holes) and manifold (no self intersections). These properties are very important as many interaction applications or visualization (shadows, ...) rely on them. Moreover the approximation of silhouette contours removes most of the background subtraction noise.
- Model Content (the type of information available, surfaces, and textures in our case). When texturing the 3D models with the images obtained form the cameras, this property enables to avoid artefacs (see figure 6). Notice that in the applications presented the model is not textured. Real-time texturing is a challenging issue as the amount of data to handle in a distributed contex is important. This is an on going work with very promising preliminary results.

Fig. 6. Details of a 3D model and its textured version (off-line).

8 Conclusion

We presented a marker-less 3D shape modeling approach which optimally exploits all the information provided by standard background subtraction techniques and produces watertight 3D models. The shape can easily be used for visual interactions, like rendering, shading, object occlusion, as well as mechanical interactions, like collision detection with other virtual objects. I/O devices and computing units are commodity components (FireWire cameras, PCs, gigabit Ethernet network, classroom projectors). They provide a scalable and efficient environment. The aggregation of multiple units and an adequate work-load distribution enable us to achieve real time performance.

Future works investigate two directions. One is to focus on data quality, in particular background subtraction and temporal consistency. The other is to focus on recovering semantic information about scene objects. The goal is to identify parts of the users body for motion tracking, gesture recognition and more advanced interactions with the virtual world.

References

1. 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 *Proceedings of ICVS, Vancouver (Canada)*, 2001.
2. Authors.
3. Authors.
4. Authors.
5. Eugene Borovikov and Larry Davis. A Distributed System for Real-time Volume Reconstruction. In *proceedings of CAMP-2000, IEEE*, 2000.
6. J. Carranza, C. Theobalt, M. Magnor, and H.P. Seidel. Free-viewpoint video of human actors. In *Proc. of ACM SIGGRAPH, San Diego*, pages 569–577, 2003.
7. G. Cheung, S. Baker, and T. Kanade. Visual Hull Alignment and Refinement Across Time: A 3D Reconstruction Algorithm Combining Shape-From-Silhouette with Stereo. In *Proceedings of CVPR, Madison*, 2003.
8. 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 CVPR, Hilton Head Island*, volume 2, pages 714 – 720, June 2000.
9. I. Debled-Rennesson, S. Tabbone, and L. Wendling. Fast Polygonal Approximation of Digital Curves. In *Proceedings of ICPR*, volume I, pages 465–468, 2004.
10. 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.
11. T. Horprasert, D. Harwood, and L.S. Davis. A Statistical Approach for Real-time Robust Background Subtraction and Shadow Detection . In *IEEE ICCV'99 frame-rate workshop*, 1999.
12. 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*, pages 1422–1428, 2000.
13. A. Laurentini. The Visual Hull Concept for Silhouette-Based Image Understanding. *IEEE Transactions on PAMI*, 16(2):150–162, February 1994.
14. M. Li, M. Magnor, and H.-P. Seidel. Improved hardware-accelerated visual hull rendering. In *Vision, Modeling and Visualization Workshop, Munich*, 2003.
15. W. Matusik, C. Buehler, R. Raskar, S. Gortler, and L. McMillan. Image Based Visual Hulls. In *Proceedings of ACM SIGGRAPH*, pages 369–374, 2000.
16. P.J. Narayanan, P.W. Rander, and T. Kanade. Constructing Virtual Wolrds Using Dense Stereo. In *Proceedings of ICCV, Bombay, (India)*, pages 3–10, 1998.
17. W. Stephan, L. Edouard, and G. Markus. 3D Video Fragments: Dynamic Point Samples for Real-time Free-Viewpoint Video. *Computers & Graphics*, 28(1):3–14, 2004.
18. X. Wu and T. Matsuyama. Real-Time Active 3D Shape Reconstruction for 3D Video. In *Proceedings of the 3rd International Symposium on Image and Signal Processing and Analysis, Rome (Italy)*, pages 186–191, September 2003.