A platform for monitoring aspects of human presence in real-time

X. Zabulis[1], T. Sarmis[1], K. Tzevanidis[1,2], P. Koutlemanis[1], D. Grammenos[1], and A. A. Argyros[1,2]

[1]Institute of Computer Science - FORTH Herakleion, Crete, Greece
[2] Department of Computer Science, University of Crete

Abstract. In this paper, the design and implementation of a hardware/software platform for parallel and distributed multiview vision processing is presented. The platform is focused at supporting the monitoring of human presence in indoor environments. Its architecture is focused at increased throughput through process pipelining as well as at reducing communication costs and hardware requirements. Using this platform, we present efficient implementations of basic visual processes such as person tracking, textured visual hull computation and head pose estimation. Using the proposed platform multiview visual operations can be combined and third-party ones integrated, to ultimately facilitate the development of interactive applications that employ visual input. Computational performance is benchmarked comparatively to state of the art and the efficacy of the approach is qualitatively assessed in the context of already developed applications related to interactive environments.

1 Introduction

Advances in several research and technological fields have increased the likelihood of creating interactive environments that adapt to various aspects of human presence. Towards such environments, vision based estimation of the location and geometric attributes of the human body is of interest, as it unobtrusively conveys information on user activities. Multiview scene observation enables accurate and robust 3D reasoning, particularly in environments that are imaged distantly and at which occlusions are frequent and spatially extended. The large volume of generated data combined with high framerate requirements, calls for distributed and parallel image acquisition and processing, as well as efficient communication strategies. A multiview platform is proposed in this paper, whose architecture aims at the reduction of computational and communication costs. The platform provides functionalities for synchronized and distributed image acquisition and visual processing. Moreover, its complexity is encapsulated utilizing a middleware infrastructure so that the output of multiview visual computation can be communicated to third-party applications.

Upon this platform, a set of key visual processes are implemented that estimate aspects of human presence in real time, such as the volumetric occupancy of the monitored environment, the textured visual hull of persons, their location and their head pose. The corresponding implementations capitalize on the proposed platform to pipeline processes and reduce communication costs. By encapsulating the platform functionalities implemented in a middleware infrastructure, pilot applications can be developed in simple programming environments, transparently to the implementation of the platform.

The remainder of this paper is organized as follows. In Sec. 2, related work is reviewed. Sec. 3, presents the setup and architecture of the proposed platform. The implementation of key components of the system is described in Sec. 4. In Sec. 5, the performance of the proposed platform is benchmarked and qualitatively assessed through pilot applications. Sec. 6 summarizes contributions and provides directions for future research.

2 Related work

Although the use of camera clusters has been increasing, there exist only a few platforms that facilitate the development, pipelining and integration of parallel and distributed visual processes. Even fewer are the platforms that focus on estimating person locations and geometric attributes of the human body. Some multi-camera platforms gather and display input from multiple video streams, and either process each stream individually (e.g. [1–3]), or perform simple multiview operations (i.e. stream mosaicing [2]). Such platforms are typically based on centralized architectures, as they exhibit moderate computational requirements.

Multiview computations require significantly more the computational power raising the need for parallel computation. Parallel and distributed platforms (i.e. [4]) have been utilized to reconstruct a generic scene from multiple stereo views computed in parallel. More relevant to this work are systems [5–7] that reconstruct persons in the scene through their volumetric occupancy or the visual hull [8]. Although the reconstruction is approximate, i.e. concavities are not represented, the approach is sufficient for tasks such as person detection and localization. The systems in [6, 7] compute the visual hull volumetrically, enabling massive parallelization of computation and direct application of 3D linear operations (i.e. noise filtering) on the reconstructed volume. In contrast, [5] proposes view-based parallelization, based on the silhouette extracted from each view, and results in a mesh whose nodes are irregularly arranged in 3D space. Parallelization is massive in [6, 7] which run on GPU(s), while [5] parallelizes computation in dual-core CPUs and leads to increased hardware requirements.

3 Software platform

3.1 System setup

A typical physical setup of the system involves a $6 \times 6 \times 2.5m^3$ room, including a large (i.e. $5 \times 2\,m^2$) backprojection display providing visual feedback. The camera cluster consists of 8 *Dragonfly2 Point Gray* cameras, mounted near the ceiling of the room viewing it peripherally and employs 2 or 4 computers with an Intel *i920* quad-core CPU and an *NVIDIA GTX275* GPU each. Cameras are evenly distributed to *host* computers and have a maximum framerate of $30\,fps$ at 1280×960 image resolution. Synchronized image acquisition uses an additional *FireWire* bus across computers and timestamps, guaranteeing a maximum of $125\mu sec$ temporal discrepancy among images with the same timestamp. Henceforth, the set of the N simultaneously acquired images I_i is referred as a *multiframe*, the projection matrices for each view $i \in [1, 2, 3, ..., N]$ denoted as P_i and corresponding camera centers as $\boldsymbol{\kappa}_i$. The computers in each system are connected by $1\,GB$ Ethernet in a star network topology, where one computer is declared the *central* workstation and the rest as *satellites*.

Cameras are automatically calibrated intrinsically and extrinsically, employing a checkerboard detector [9] to find reference points and passing them to a standard calibration toolbox [10]. Corresponding reference points across views are used to increase calibration accuracy, via bundle adjustment [11].

3.2 Architecture

The complexity of camera control and synchronized image acquisition is encapsulated in a software platform. The platform supports the synchronized communication of images and intermediate computation results across processing nodes through a shared memory. Results of visual computations become available to applications via integration with a middleware infrastructure.

A broad spectrum of camera types is supported, connected to host computers by Direct Memory Access to RAM. Each host workstation maintains a fixed RAM buffer for every view in which it stores captured frames after converting them from Bayer Tile to RGB format and rectifying them for lens distortion. The operations are implemented in the GPU of host computers with image storage rate matching the camera framerate. Images are stored together with associated timestamps and, as new frames arrive, older ones are removed.

Each time a new image enters a buffer, its timestamp is notified to the central workstation. During the creation of a multiframe, the central workstation selects the appropriate timestamps for each buffer, local or remote. Then, it broadcasts timestamp queries to the satellite workstations and acquires the queried frames, while for local buffers it fetches the frames from its RAM. This way, a frame that is dropped at a view does not disturb synchronization and, also, the transmission of a frame for a multiframe that will be eventually rejected is avoided.

Both images and intermediate computational results can be stored in a shared memory at the central workstation where multiple processes may have simultaneous access. Processes can concurrently read the data from the shared memory without copying them to the process' address space. By adding another computer, the central workstation can be relieved from image acquisition and preprocessing.

The platform is further integrated with a middleware infrastructure to facilitate the development of new visual processes through an API that supports the

C/C++, .NET, Java, Python, Delphi, and *Flash/ActionScript* programming languages. Through this middleware, the output of such visual processes (i.e. those in Sec. 4), can be communicated to applications in the form of event notifications, hiding the details of network connections and data serialization. The same middleware is also employed in the control of actuating components of the environment such as displays, illumination and sound. These capabilities simplify the development of interactive applications and enable the integration of vision processes with the reasoning and actuating components of the environment.

4 Key vision processes

Four processes that compute basic aspects of human presence have been implemented. These constitute the core visual processes for developing pertinent interactive applications. The processes compute the volumetric occupancy of persons in the images environment, their textured visual hulls, their locations and motion trajectories and their 3D head poses. To meet framerate requirements, these implementations are designed to be distributed and massively parallel.

4.1 Volumetric occupancy

Volumetric occupancy of the imaged scene is represented in a voxel grid V. Along with image rectification, background subtraction of images is performed locally on the GPU of host computers. A pixelwise background subtraction [12] is employed to parallelize the operation. In contrast to [6, 7] we do not perform morphological operations to compensate for background subtraction errors, but consider them in reconstruction. The results, images B_i, are transmitted in Run Length (RL) encoding to shared memory.

The scene is reconstructed on the GPU of the central workstation by assigning a thread per voxel $\boldsymbol{v}$. A voxel $V(\boldsymbol{v})$ is labeled as 1 if found to be occupied and 0 if not. To process volumes of arbitrary size, computation of V is partitioned and results concatenated in shared memory, as opposed to [6, 7] where the dimensions of V are constrained by the GPU's memory capacity.

The value at $V(\boldsymbol{v})$ is computed amongst the views $i' \in C \subseteq [1, 2, 3, ..., N]$, that $\boldsymbol{v}$ projects within their visual field. Ideally, occupied voxels should project in foreground regions in *all* $B_{i'}$ and vice versa. To compensate for errors in background subtraction, a more lenient rule is applied, and a voxel is considered as occupied if at least half of the views in which it is visible concur that it projects in a foreground region. That is, $V(\boldsymbol{v})$ is 1 if

$$\sum_{i'} \left(B_{i'}(P_{i'} \cdot [\boldsymbol{v}; 1]^T) \right) > card(C)/2 \qquad (1)$$

and 0 otherwise. Fig. 1(left), illustrates the process for a challenging scene. Optionally, V can be filtered with a 3D kernel to suppress voxelization artifacts; see Fig. 1(right).

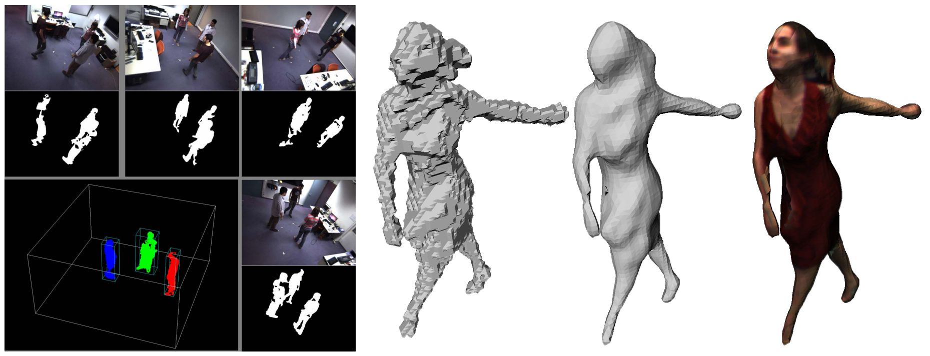

Fig. 1. Left: Original images, background subtraction and volumetric reconstruction, for a scene imaged by 4 cameras. Persons are imaged against cluttered background and uneven illumination, resulting in inaccurate background subtraction. Persons occlude each other in all views. Reconstruction is not accurate, but sufficient for person tracking. Right: Visual hull (left), smooth visual hull (middle) and textured visual hull (right), for a benchmark dataset [13].

4.2 Person localization and tracking

A multiview approach is adopted for person localization as such approaches typically outperform single-view [14, 15], due to the systematic treatment of occlusions. As in [16–20], we employ a planar homography constraint to map imaged persons to the ground plane, but we consider the occupied volume instead of the projection area. As in [19], we also utilize volumetric occupancy to increase the localization robustness, but do not require that the number of tracked persons is a priori known.

A GPU process projects V on the ground plane and sums occupied voxels along the direction perpendicular to the floor. This results in a 2D buffer F, registered to the ground plane. Image F is transferred to shared memory where it is collected by a tracker [21] that establishes temporal correspondence of person locations. In F, persons appear as intensity blobs, which are tracked only if they exhibit a sufficient amount of volume, as measured by their intensity sum in F. New persons are detected as new blobs that exhibit a temporal persistence over a few frames. The tracker is implemented in CPU and modified to track intensity blobs, rather than skin-colored blobs in color images for which it was originally formulated. It is robust to transient localization failures and, most importantly, designed to retain the tracking of blobs even if those appear merged for long temporal intervals. This way, tracking succeeds even if subjects are close to each other forming a single connected component in V (see Fig. 2).

Tracking robustness is supported by the high frame rate ($> 10\,Hz$) of operation and, also, by fine granularity of volumetric representation ($1cm^3$). High framerate casts blob motion in F smooth and simpler to track. Fine granularity increases the precision of blob localization in F. To conserve communication

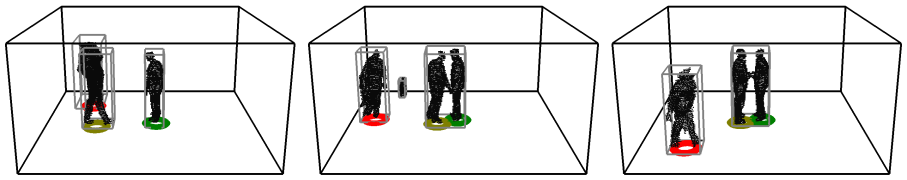

Fig. 2. Person localization and tracking. Estimated person location is marked with colored circles. Tracking is successful although in some frames visual hulls are merged. Occurring transiently, the spurious structure in frame 2 is disregarded by the tracker.

cost, middleware events are created upon change of person location, or when persons enter and leave the scene.

4.3 Textured visual hull

Collecting the output of volumetric occupancy from shared memory (Sec. 4.1), a GPU process estimates the visual hull of persons and, optionally, textures it. The hull is computed as the 0-isosurface in V, by a parallel implementation of the "Marching Cubes" algorithm [22]. To benefit from operations optimized on GPU hardware, V is treated as a 3D texture. A thread is assigned to each voxel and accesses V's values at the locations of mesh vertices by interpolating the 3D texture at the corresponding locations; further details can be found in [23]. The isosurface represents the visual hull and is encoded as a mesh M of triangles in shared memory.

Texture mapping on each triangle j of M also makes use of GPU-optimized operations. Initially, the views in which j is visible are identified, by employing a depth buffer Z_i for each view i. Each pixel in Z_i encodes the distance of $\boldsymbol{\kappa}_i$ to the surface that is imaged at that pixel. Buffer Z_i is computed by calculating the distance $\delta_{ij} = |\boldsymbol{\tau}_j - \boldsymbol{\kappa}_i|$ for each triangle, where $\boldsymbol{\tau}_j$ is the triangle's 3D centroid. Triangles are projected on Z_i and the minimum distance that is imaged in each pixel of Z_i is assigned to that pixel. Let Δ the length of a voxel's side. Then,

$$|\delta_{ij} - Z_i(P_i \cdot [\boldsymbol{\tau}_j; 1]^T)| < \Delta, \tag{2}$$

is a criterion that indicates if triangle j is indeed imaged at location of view i; (2) is false, if triangle j is occluded in view i. Threshold Δ is sufficient as M's triangles are contained within voxel size. This criterion also facilitates parallelization since, otherwise, the sequential maintenance of the list of triangles imaged at $Z_i(P_i \cdot [\boldsymbol{\tau}_j; 1]^T)$ would be required to cope with pixels imaging multiple triangles along the images of M's vertices. Aiming at efficiency, the number of considered triangles is reduced by disregarding those whose normal forms an angle greater than $\pi/2$ with the optical axis of view i. Texture coordinates of triangle nodes, $P_i \cdot [\boldsymbol{\tau}_j; 1]^T$, have been already computed during the evaluation of (2) and are retrieved instead of recomputed.

To resolve multi-texturing conflicts in triangles visible in multiple views, textures are blended according to a weighting factor proportional to the size of the projected area, on a pixel shader of the GPU, so that distal and oblique views are weighted less. Further details can be found in [23]. Figure 1(right) visualizes the obtained results.

4.4 Head pose estimation

Head pose estimation provides information about the direction at which a person is facing at. The task is challenging in wide areas, because faces are imaged in poor resolution and are often occluded. The multiview head pose estimation method in [24] is parallelized and employed to provide an estimate of 3D head center location $\boldsymbol{c}$ and the 3 rotational pose components (*pitch, yaw, roll*).

The method exhibits increased accuracy in the context of distant viewing, over other head pose estimation methods that fuse single-view pose estimates [25–27] and yield only 2 pose components. It method utilizes the textured visual hull M to collect all available facial texture fragments and resolve occlusions, which is received as input from the module of Sec. 4.3.

An instance of this method is applied independently on each person detected by the module in Sec. 4.2. For each person, head center $\boldsymbol{c}$ is tracked by a variant of the Mean Shift [28] algorithm, using a 3D spherical kernel S matched to the part of M that reconstructs the persons head. The system broadcasts $\boldsymbol{c}$ to host workstations which perform concurrently face detection [29] within the areas α_i of I_i where S projects. Per-view orientation estimates (*pitch, yaw*) are obtained as $\boldsymbol{o}_k = \boldsymbol{f}_k - \boldsymbol{c}_k$, where k enumerates the views where a face was detected in α_k; f_k is the intersection of M with the ray from camera k through the face's detection in α_k. Estimates $\boldsymbol{o}_k$ are fused at the central workstation into a median vector $\boldsymbol{o}_c$.

The texture of the visual hull is then projected on S, with $\boldsymbol{c}$ being the center of projection to form a spherical image I_s, an operation optimized on the GPU as texture mapping. By construction, exactly one frontal face occurs in I_s and, thus, a generic CPU-based frontal face detector [29] suffices for its detection. To form I_s, only areas α_i of I_i are transmitted to the central workstation. Orientation $\boldsymbol{o}$ is provided by the face center $\boldsymbol{p} \in I_s$ using a look up table to find its 3D correspondence on S. The orientation γ of the face in I_s provides the *roll* component of 3D pose and is optionally computed by optimizing a correlation-based symmetry operator in the $[0, \pi)$ range. The availability of the precomputed estimate $\boldsymbol{o}_c$ simplifies the above process as follows. S is in-place rotated by R, $R \cdot [1\,0\,0]^T = \boldsymbol{o}_c$, so that the projected face occurs approximately (a) at the equator of S, where spherical distortions are minimized and (b) at the center of I_s thus reducing the area were a face is searched for.

The output, head center $\boldsymbol{c}$ and 3 pose angles, is communicated through middleware and associated to the tracking id of the person as this has been derived in Sec. 4.2. Representative head pose estimation results are illustrated in Fig. 3.

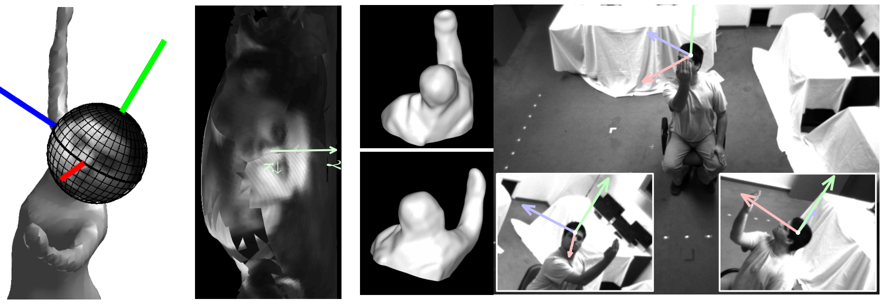

Fig. 3. 3D head pose estimation. On the left, shown is the visual hull with facial texture projected on S, for a benchmark dataset [13]. To its right the generated spherical image I_s is shown, with a the green vector indicating the face center and its 2D orientation in I_s. In the right panel, shown are two views of the visual hull of a subject during our experiments and the estimation result superimposed on some of the images I_i.

5 Experiments

5.1 Computational performance

The evaluated system consists of 8 cameras, using 1, 2 or 4 computers and full or lower resolution versions of I_i. Columns a - f in Table 1 benchmark the performance of the proposed implementations and compare it with pertinent methods (left three columns), for *visual hull* computation, for the same amount of voxels. In columns $T1$ and $T2$ the performance of computing the *textured visual hull* is reported. The 1^{st} row marks the achieved framerate, the 2^{nd} the amount of computational power utilized, the 3^{rd} the number of voxels in V, the 4^{th} the resolution of I_is, and the 5^{th} the number of computers employed. Analytical performance measurements in more conditions and for intermediate operations (i.e. lens distortion compensation, background subtraction) can be found in [23]. The latency between a person's motion and the reception of the corresponding event is $\approx 140\,ms$ and localization accuracy is $\approx 4\,cm$.

Table 1. Performance measurements and comparison (see text).

	[6]	[7]	[30]	a	b	c	d	e	f	T1	T2
Hz	25	14	30	23.8	34	40	98.1	71.4	103.2	25.3	24.1
$GFLOPS$	1614	933	836	437	437	894	894	1788	1788	1788	1788
voxels	2^{11}	2^{11}	2^{11}	2^{11}	2^{11}	2^{11}	2^{11}	2^{11}	2^{11}	2^{24}	2^{24}
$pixels$	5×2^7	5×2^7	5×2^7	5×2^7	5×2^6	5×2^7	5×2^6	5×2^7	5×2^6	5×2^7	5×2^8
computers	5	1	11	1	1	2	2	4	4	4	4

In [6], the computation of V is distributed in the GPUs of multiple computers. For each $\boldsymbol{v}$, a partial estimate of $V(\boldsymbol{v})$ is computed and transmitted centrally, to be fused with the rest of estimates. Albeit the RL compression, communication overhead is significant as it corresponds to N times the number of voxels in V. The increased performance of the proposed method stems both from the parallelization of the background subtraction stage and, most importantly, from the transmission of (RL encoded) images B_i that requires significantly less capacity. In [5], a minimal communication cost is obtained, by transmitting only the silhouettes in B_is, but then only per-view parallelization is achieved, as opposed to per-voxel. The system in [7] eliminates this communication cost by centralizing all computation in one computer. This solution does not scale well with the number of views (the online version is limited to four). In contrast to [6], increasing granularity of V does not increase communication cost, as transmission cost of images B_i is constant.

Regarding head pose estimation, coarse orientation estimation $\boldsymbol{o}_c$ runs at $\approx 10\,Hz$ and drops to $\approx 2\,Hz$ when estimating 3D pose to the precision of 1°, for 8 views in 2 computers. Accuracy experiments replicate the $\approx 3^\circ$ accuracy reported in [24] for I_s of 1280×960 resolution, but accuracy drops to $\approx 5^\circ$ when I_i are 640×480. The bottleneck of the whole process is face detection which is implemented in the CPU. To parallelize it, the task is distributed to the computers hosting the cameras.

Overall, results indicate improvement of state of the art in performance, efficient use of computational resources and linear scaling of computational demands with respect to reconstructed volume and number of views.

5.2 Pilot applications

Pilot applications were developed to evaluate (a) the proposed system in interactive scenarios and (b) the development process based on the proposed platform. Interactivity is supported by audiovisual feedback, provided by wall mounted displays and a surround audio system. Applications are implemented in *Flash Action Script* and communicate with the platform through the middleware.

In a cultural heritage application, a fresco is projected on a wall of the room. Visitors observing the projection are localized and their position in front of the projection is tracked (Fig. 4, bottom). Depending on current and previous user locations, the display is augmented with visual and textual information in the language of each individual user. The application utilizes person localization events to implement the interaction scenario, which also incorporates contextual environmental constraints, i.e. further information about the fresco region is provided if a user re-visits a particular location. More details on the above scenario can be found in [31].

Using the middleware layer, the visual processes in Sec. 4 can be customized. For example, in a gaming application called *footprints*, as players move around, the contact area of their feet with the floor is used to virtually paint the floor. The method in Sec. 4.2 is employed to determine user location and then, the method in Sec. 4.1 is re-invoked at higher voxel tessellation only at the corresponding volumes to reconstruct the volume around the feet of the user. The increased reconstruction accuracy is also employed in *walkman*, where alternating black

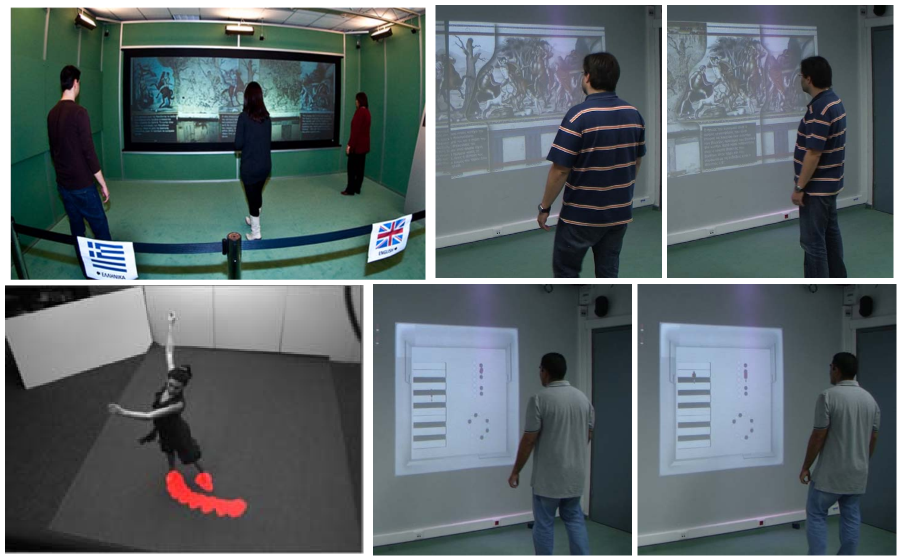

Fig. 4. Pilot applications. Top: The display is updated based on the location and walk-through trajectories of visitors. Bottom-left: footprint extraction; obtained footprints over 20 frames are shown superimposed. Bottom-right: the interactive game's interface is projected on the wall providing visual feedback to the users, as they play music by stepping in appropriate spots.

and white areas on the floor represent piano keys and players can play music by stepping on them. Similarly, the cultural application above has been extended to collaborate with the method in Sec. 4.4 to determine the fresco region that each user is facing at. Using the result of person localization, only the volume around the person's head is reconstructed at high resolution.

Pilot applications were evaluated and showcased to more than 100 persons, which exhibited diversity in age, gender, cultural and professional background (see [31] for more details on this usability evaluation). The overall impression was that due to brisk system response the applications are considered as exciting and engaging and, thus, system response was adequate towards supporting the aforementioned interactive applications.

6 Conclusions

This paper presented a multiview platform that facilitates the development and integration of parallel and distributed visual processes. On top of this platform, basic visual processes have been efficiently implemented. The functionalities of the platform become accessible via an integrating middleware layer that communicates high-level visual computation results to the application layer. The efficacy of the proposed implementation and architecture is assessed by the development, in a simple prototyping language, of pilot applications that utilize the developed infrastructure. The proposed system is characterized by increased robustness in tracking persons at high framerate, and its reduced requirements in computational hardware. The requirements are linearly related to the volume of required computation, or otherwise, the spatial extent of the area to be covered and the number of views. System architecture adapts to the availability of resources, few or abundant, and system performance scales linearly with respect to them.

Future work regards the adoption of GPU implementation of the face detection process (i.e. [32], the adoption of occlusion maps so that room furniture can be accounted in visibility computation [7]. Additionally, we plan to expand the set of available visual processes (e.g., by implementing gesture recognition) in order to enrich the repertoire of interaction capabilities and facilitate the development of more elaborate applications over the proposed platform.

Acknowledgements

This work was partially supported by the FORTH-ICS internal RTD Programme "Ambient Intelligence and Smart Environments".

References

1. Ramachandran, U., Nikhil, R., Rehg, J., Angelov, Y., Paul, A., Adhikari, S., Mackenzie, K., Harel, N., Knobe, K.: Stampede: a cluster programming middleware for interactive stream-oriented applications. IEEE Trans. Parallel and Distributed Systems **14** (2003) 1140 – 1154
2. Gualdi, G., Prati, A., Cucchiara, R., Ardizzone, E., Cascia, M.L., Presti, L.L., Morana, M.: Enabling technologies on hybrid camera networks for behavioral analysis of unattended indoor environments and their surroundings. In: ACM Multimedia Workshops. (2008) 101–108
3. Chen, P., Ahammad, P., Boyer, C., Huang, S., Lin, L., Lobaton, E., Meingast, M., Oh, S., Wang, S., Yan, P., Yang, A., Yeo, C., Chang, L., Tygar, J., Sastry, S.: CITRIC: A low-bandwidth wireless camera network platform. In: ACM/IEEE Int. Conference on Distributed Smart Cameras. (2008) 1–10
4. S.H. Jung, R.B.: A framework for constructing real-time immersive environments for training physical activities. Journal of Multimedia **1** (2006) 9–17
5. Allard, J., Franco, J., Menier, C., Boyer, E., B., R.: The Grimage platform: A mixed reality environment for interactions. In: ICCVS. (2006)
6. Ladikos, A., Benhimane, S., Navab, N.: Efficient visual hull computation for real-time 3d reconstruction using CUDA. In: CVPR Workshops. (2008) 1–8
7. Schick, A., Stiefelhagen, R.: Real-time GPU-based voxel carving with systematic occlusion handling. In: DAGM Symp. on Pattern Recognition. (2009) 372–81
8. Laurentini, A.: The visual hull concept for silhouette-based image understanding. PAMI **16** (1994) 150–162

9. Sarmis, T., Zabulis, X., Argyros, A.A.: A checkerboard detection utility for intrinsic and extrinsic camera cluster calibration. Technical Report TR-397, FORTH-ICS (2009)
10. Bouguet, J.Y.: Camera calibration toolbox for Matlab. (http//www.vision.caltech.edu/bouguetj/calib_doc)
11. Lourakis, M., Argyros, A.: SBA: A software package for generic sparse bundle adjustment. ACM Transactions on Mathematical Software **36** (2009)
12. Zivkovic, Z.: Improved adaptive Gaussian mixture model for background subtraction. In: International Conference on Pattern Recognition. (2004) 28–31
13. INRIA Perception Group http://4drepository.inrialpes.fr/.
14. Tran, S., Lin, Z., Harwood, D., Davis, L.: UMD VDT, an integration of detection and tracking methods for multiple human tracking. In: CLEAR. (2008)
15. Wu, B., Singh, V., Kuo, C., Zhang, L., Lee, S., Nevatia, R.: CLEAR'07 evaluation of usc human tracking system for surveillance videos. In: CLEAR. (2008) 191–196
16. Khan, S., Shah, M.: A multiview approach to tracking people in crowded scenes using a planar homography constraint. In: ECCV. (2006) 133–146
17. Mittal, A., Davis, L.: M2tracker: A multi-view approach to segmenting and tracking people in a cluttered scene. In: IJCV. (2003) 189–203
18. Reddy, D., Sankaranarayanan, A., Cevher, V., Chellappa, R.: Compressed sensing for multi-view tracking and 3-D voxel reconstruction. In: ICIP. (2008) 221–224
19. Fleuret, F., Berclaz, J., Lengagne, R., Fua, P.: Multicamera people tracking with a probabilistic occupancy map. PAMI **30** (2008) 267–282
20. Liem, M., Gavrila, D.: Multi-person tracking with overlapping cameras in complex, dynamic environments. In: BMVC. (2009)
21. Argyros, A., Lourakis, M.: Real time tracking of multiple skin-colored objects with a possibly moving camera. In: ECCV. Volume 3. (2004) 368–379
22. Lorensen, W., Cline, H.: Marching cubes: A high resolution 3D surface construction algorithm. In: SIGGRAPH. (1987) 163–169
23. Tzevanidis, K., Zabulis, X., Sarmis, T., Koutlemanis, P., Kyriazis, N., , Argyros, A.: From multiple views to textured 3d meshes: a gpu-powered approach. In: ECCV Workshops. (2010) 5–11
24. Zabulis, X., Sarmis, T., Argyros, A.A.: 3D head pose estimation from multiple distant views. In: BMVC. (2009)
25. Voit, M., Nickel, K., Stiefelhagen, R.: Neural network-based head pose estimation and multi-view fusion. In: CLEAR. (2007) 291–298
26. Zhang, Z., Hu, Y., Liu, M., Huang, T.: Head pose estimation in seminar room using multi view face detectors. In: CLEAR. (2007) 299–304
27. Tian, Y., Brown, L., Conell, J., Pankanti, S., Hapapur, A., Senior, A., Bolle, R.: Absolute head pose estimation from overhead wide-angle cameras. In: AMFG. (2003) 92–9
28. Comaniciu, D., Meer, P.: Mean shift : A robust approach toward feature space analysis. PAMI **24** (2002) 603–619
29. Viola, P., Jones, M.: Rapid object detection using a boosted cascade of simple features. In: CVPR. (2001) 511–8
30. Franco, J., Menier, C., Boyer, E., Raffin, B.: A distributed approach for real time 3D modeling. In: CVPR Workshops. (2004) 31
31. Zabulis, X., Grammenos, D., Sarmis, T., Tzevanidis, K., Argyros, A.A.: Exploration of large-scale museum artifacts through non-instrumented, location-based, multi-user interaction. In: VAST. (2010)
32. Naruniec, J.: Using GPU for face detection. In: SPIE. Volume 7502. (2009) 204–206