We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

3

The Cloud-Mobile Convergence Paradigm for Augmented Reality

Xun Luo
Office of the Chief Scientist, Qualcomm Inc.
USA

1. Introduction

Mobile computing for massive users has gained enormous progresses in the last few years. Among the main propelling forces for this momentum are the performance improvements of mobile application processors, increasing of storage and sensing capabilities, as well as breakthroughs in manufacturing technologies for high-quality mobile displays. Along with such progresses, the way people perceive mobile devices evolved over time as well, maybe best reflected by the changing names. During the 1990s, the mobile devices were called cellular phones and PDAs (personal digital assistants), to differentiate between the then-major functionalities of voice communication and handheld computing. Later the word "smart phone" was more popularly used, attesting the convergence between communication and computing elements. Nowadays, new technical jargons such as tablets, pads, and smart books are often used to refer to mobile devices. Behind the hype of this name changing game is the simple fact that personal computing is rapidly shifting towards being mobile.

There are large number of applications and services designed and optimized for mobile computing. Within the herd Augmented Reality (AR) stands out and attracts considerable attention. AR is the technology that superimposes computer generated graphics over real-life live video feeds, using registration as the magic glue to blend the two seamlessly together. The origin of the term "Augmented Reality" can be traced back to 1990, although applications featuring some of the basic characteristics had been in place as early as the 1960s. On the other hand, mobile AR systems which are of practical use are relatively new and did not emerge until recently. The reason is that such systems put high requirements on the hosting device's software and hardware configurations.

To better understand the challenges, let's take a look at Figure 1 which illustrates the components of a conceptual mobile AR system. In this system, sensor (including both the camera sensor and others) inputs are used to estimate the six degree-of-freedom pose of the on-device camera. The sensor inputs are also used by a user interaction module which implements functionality of gesture control. An I/O module reads graphics models from a database and provides them to a rendering module. The latter combines rendered graphics with live video. As the orchestrated output of all the modules, the interactive and augmented scene is presented to the end user.

Now we examine what are the necessities to make the illustrated system capable of providing smooth user experiences. First of all, the system needs to have *interactivity*, which translates

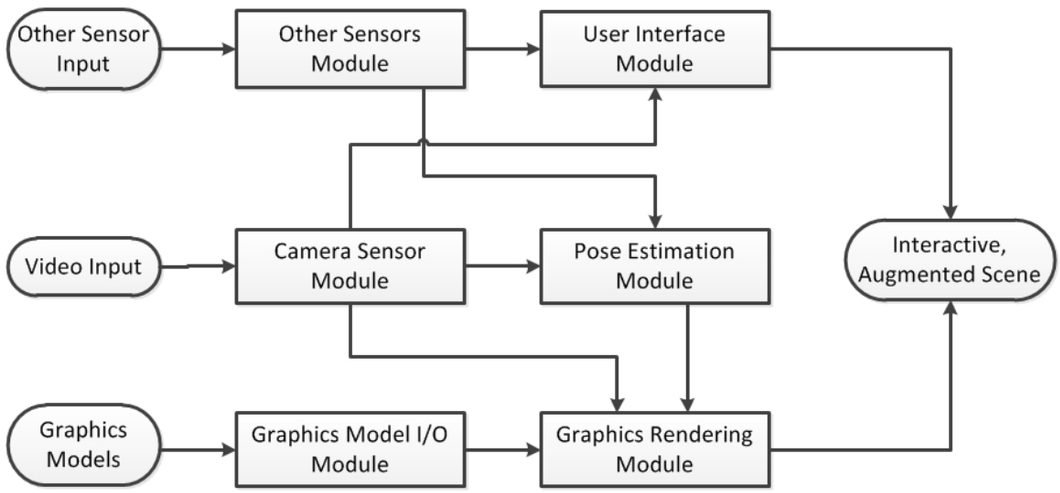


Fig. 1. The components of a typical mobile AR system.

to frame rates above a minimum number, usually 30 frames per second. *Fidelity* is another requirement, which means resolution of the final combined frames should achieve beyond a decent level such as VGA (640 by 480 pixels), with sufficient graphical details. A further requirement is *scalability*, indicating that there shall not be stringent restrictions on the quantity of graphics models, nor the size of each individual ones. Last but not least, users desire *robustness*, that is, even under noisy and complex environment conditions the pose estimation module should still function with satisfactory detection rate and accuracy. Not surprisingly, meeting these requirements calls for fast video processing, high I/O bandwidth and efficient algorithm computation. These have been equipped more or less by modern mobile devices, but not yet enough. The obstacles are multi-folds:

- Architecture difference – It is often the case that mobile application processors are designed to have different architectures from high-end servers and even low-end PCs. The rationale for such differences is mostly for power efficiency. At the price of this design choice is the performance discrepancy introduced by architecture difference.
- Hardware capability – It is not completely uncommon for mobile devices to share the same or similar architectures with their stationed counterparts. However, even under this situation, significant capability gap may still exist between them. Take multi-core architecture for example: the number of cores in a mobile multi-core processor is usually much fewer than that of a desktop multi-core processor. Programable GPU architecture is another example: most mobile GPUs implement only a subset of the functionalities of the desktop products.
- System availability – Because mobile devices do not have the luxury of unlimited power, energy conservation is a must-have for them. This impacts the system availability accordingly. For instance, we cannot assume a smart phone in the field to run an intensive computer vision task overnight without being plugged with external power supply. Albeit the same operation could be conducted by a desktop PC with the same processor clock frequency without difficulty.

Up to now, the approaches to tackle these constraints have been mainly focused on designing algorithms as "lean and mean" as possible on the mobile devices. From this point of view,

the endeavors of making AR to work on mobile devices are very similar to those of making AR to work on PCs a decade ago. But unlike PCs by then, mobile devices nowadays often find themselves surrounded by abundantly available cloud-based resources. In addition to that, it is even often the case that correspondent AR components have been implemented on these cloud peers more efficiently already. In this chapter we attempt to look at the problem from a new and different perspective, i.e. closely integrate the computing resources on the mobile device as well as in its cloud environment. We propose a novel alternative paradigm, named Cloud-Mobile Convergence (CMC). Both the high-level design principle and low-level proof-of-concept implementations of CMC are presented.

The rest of this chapter is organized as follows. In section 2 a review of traditional techniques to solve "performance vs. resource" problem in AR is briefly given. Section 3 describes key concepts and high-level design principles of CMC. Following that, several sample scenarios are presented in section 4. Section 5 conducts a deeper dive by studying in detail a proof-of-concept application, which applies CMC paradigm to accelerate computer vision-based human-computer interaction. We conclude the chapter in section 6 with discussions.

2. Existing optimization measures for mobile AR

As illustrated in Figure 1, AR systems can use multiple sources of sensor inputs for camera pose estimation and interaction. In practice, the camera sensor is mostly used. For simplicity reasons, from now on if not specifically mentioned, the AR systems we discuss only include vision-based ones. In such systems, vision-based processing tasks consists of *recognition* and *tracking*.

Recognition is the task to find one or multiple pre-defined objects in the video scene. The objects are first summarized – off-line or on-line – into digital presentation of features, and then matched with the features extracted from the video scene. If there are matches found, the corresponding objects are recognized. There have been a large number of feature detection and description methods proposed and developed in recent decades. Representative ones among them include Harris Corners (Harris & Stephens, 1988), Shi and Tomasi (Shi & Tomasi, 1994), SIFT (Lowe, 2004), SURF (Bay et al., 2006), FERNS (Ozuysal et al., 2007), and FAST (Rosten & Drummond, 2003; 2005). Tracking is the task to keep track of the recognized objects in video scene, and estimate camera pose continuously. In contrast to recognition task, tracking task is free (or largely liberated) from the obligations of feature detection, description and matching. This advantage makes it relatively faster to perform.

Although often underlooked, video acquisition itself is by no means a cheap operation. Quite a few pre-processing steps need to be done before the video is supplied to recognition and tracking tasks with satisfying quality: exposure control, focusing tuning, color adjustment etc. State-of-art camera sensors try best to automate these steps, at the price of pre-processing time that could be of the magnitude of tens of mini-seconds. The time taken by video acquisition further tightens tracking and detection budgets.

One of the research directions undertaken is to make use of hardware elements that are more efficient for computer vision algorithms processing than general-purpose CPUs. For instance, with the rapid growing of programmable GPU capabilities, researchers have explored the feasibility of porting some of the fundamental computer vision algorithms used by AR to

GPU and showcased proven performance gains. Wu developed a software library for the SIFT feature detector and descriptor (Wu, 2007). Terriberry et al. focused their efforts on SURF (Terriberry et al., 2008) and provided a working prototype. But the gains provided by following this direction were not as large as AR system designers and users had expected. On one hand, parallelism potentials of many computer vision algorithms are intrinsically limited, leaving a constrained space to fully utilize the GPU capability. On the other hand, heavy communications cost introduced by data transfer between the CPU and GPU, or among multiple GPU cores further handicapped the benefits introduced by the usage of parallel computing. It is worth noting that the numbers in (Wu, 2007) and (Terriberry et al., 2008) were based on the more favorable experimental results of using desktop GPU processors. The benchmark values on mobile GPUs would obviously be even worse.

Another viable path is to reduce the complexity of the AR tasks. Not surprisingly, this has been more or less used by most mobile AR developers. There have been several popular practices:

1. Image resolution reduction. This is intuitive to understand with a simple example: the number of pixels contained in a QVGA (320 by 240) resolution video frame is only a quarter of those in a VGA (640 by 480) resolution counterpart. Although processing complexity is not strictly linearly correlated with number of pixels, the reduction of pixels normally brings down required CPU cycles considerably. Many mobile AR systems down-sample camera frames with a gaussian kernel to archive the goal of pixel reduction.
2. Parametric tuning for computer vision algorithms. For example, the original SIFT (Lowe, 2004) implementation used a 128-dimension feature descriptor. Some variant implementation, such as (Wagner et al., 2009) used data dimension less than that to lower processing time and memory usage. Similar measures had been taken by some systems by restricting the number of feature points to detect and match.
3. Utilization of assumptions and heuristics. In a specific AR application, some useful assumptions and heuristics can be made use of to make the computer vision tasks easier. Quite a few mobile AR systems in the literature assumed that there should be no drastic camera displacements across adjacent frames. By asserting this, the burden of extensive area processing for tracking is eased. In the gesture recognition system described in (Kolsch et al., 2004), users are required to put the hand within a certain sub-region of the camera view in order to start the system. Consequently the detection cost was much lower than a full frame search. Similar requirements were also imposed by some real-time text translation applications on mobile devices (Fragoso et al., 2011).

Strictly speaking, the practices listed above should not be regarded as optimization measures. The reason is that they are not able to realize repeatable results of tackling the problem at original complexity: Down-sampling VGA resolution frames to QVGA might improve the frame rate, but it is more likely to decrease the detection rate and tracking continuity; Tuning feature descriptor dimension brings down both memory usage and detection accuracy; Assumptions and heuristics work "most of the time" and fail in corner cases, which are sometimes not rare at all to spot. Use of such techniques should be carefully thought of and conducted.

Yet another approach is to optimize the AR processing pipeline. Because there are multiple computer vision tasks in a mobile AR system, processing of them could be

optimized by using concurrent programming techniques such as multi-threading. In a typical acquisition-recognition-tracking pipeline, each downstream stage of the pipeline can start as a separate thread and synchronize with the upper stream counterparts in parallel. When the camera imager thread acquires a new frame and delivers it to recognition thread, it starts the next frame acquisition right away instead of waiting for the latter to complete. Recognition thread works in a similar relationship with the tracking thread. In general, this mechanism can be applied for any intermediate processing stages in the pipeline: In the Envisor (DiVerdi et al., 2008) system, an inertial sensor interpolation thread and a vision-based calibration thread complemented each other for tracking and map construction. In the HandyAR (Lee & Hollerer, 2007) system, separation was made for a foreground recognition thread using low-cost features and a background calibration thread using expensive features.

Limitation of the pipeline optimization approach is that even under the most idealistic situation, total processing time of the pipeline is still capped by the duration of the most time-consuming stage. The work of (Wagner et al., 2009) proposed a solution which partially alleviated this problem. The solution amortized the most expensive processing task, i.e. target detection into multiple consecutive frames. By using this technique, it achieved the relatively best performance among peers at the time of publication (multiple target recognition and tracking in real-time on mobile phones). The caveat was that the number of frames involved for task amortization could not be too large otherwise the system could be subject to stability issues.

3. The Cloud-Mobile Convergence paradigm

In section 2 we briefly reviewed the existing performance optimization measures for mobile AR. It can be seen that none of them could solve the performance issue with full satisfaction. These measures differ largely from each other, but share one thing in common: they rely only on the mobile device itself to overcome its intrinsic shortcoming of constrained resources.

In this section, we think "outside of the device" and present a novel paradigm called Cloud-Mobile Convergence (CMC). CMC is based on the observation that present day mobile devices are surrounded by abundantly available cloud resources, and access to these resources has never been as convenient before. The fundamental difference between CMC and other state-of-art solutions is that CMC considers performance improvement factors external of the mobile device, in conjunction with those on the device itself.

Under the CMC paradigm, when a mobile AR system is designed at high level, the resources pool available to it is not considered to merely reside locally. Instead, all the resources are regarded as being possible to present either internally or externally of the device. The only differentiations among the resources are the benefits they bring and the costs to use them, rather than whether they reside on the mobile device or in the cloud. This is the "convergence" essence of the paradigm.

The CMC paradigm considers computing for mobile AR tasks an optimization problem with the optional requirement of satisfying multiple constraints. If we denote R_1, R_2, ..., R_N as the N available resources, $B(R_i)$ to be the benefits brought by resource i, $C(R_i)$ to be the cost of using resource i, T_1, T_2, ..., T_M to be the M constraints to be satisfied, then the problem of optimizing a mobile AR system can be abstracted as:

$$
\begin{aligned}
&maximize \sum_{i=1}^{N} B(R_i) \\
&s.t. \\
&T_1, T_2, ..., T_M \\
&optionally, \\
&minimize \sum_{i=1}^{N} C(R_i)
\end{aligned}
\tag{1}
$$

Now that the abstraction is in place, it is time to look at how to apply the CMC paradigm to a specific mobile AR system. In a real world application, each of the variables in Equation 1 is mapped to an application-specific factor. Let's use one hypothetical system as example. This system employs two resources in the recognition stage: a worker thread R_1 that matches features extracted from the video scene to the local database, and another worker thread R_2 which performs the same matching task but against a video library stored in a data center. In this scenario two sets of definitions can be used to model benefits and costs, leading to two different resource convergence strategies adopted by the system. In one set of definition, $B(R_i)$ is the total number of objects available in R_i, and $C(R_i)$ is the average recognition time for each object in the video scene. This set of definition is obviously favorable to cloud resource R_2, because R_2 is able to provide more object candidates and perform faster object recognition using data center processor farms. In another set of definition, $B(R_i)$ is the number of objects accessible in unit time in R_i, and $C(R_i)$ is the average access latency. In contrast to the first set of definition, this one favors local resource R_1, because of the higher I/O throughput and lower access latency provided by R_1. Under both sets of definitions, constraint T can be defined as a threshold of unit time power consumption, i.e. T regulates the battery draining rate.

The above example clearly manifests the power of CMC paradigm: the generalized description is concise, yet the parameterized interpretation is flexible. Because of these advantages, CMC is an effective meta-method and philosophy that has the potential to be used widely for mobile AR system design and implementation.

4. Sample scenarios for CMC paradigm

In this section we showcase three different sample scenarios where CMC paradigm could be beneficial. Each of the scenarios sets up an unique environmental mix of on-device and cloud resources. It is worth noting that the classification criteria of these scenarios resemble those used for mobile networking characterization. The subtle difference here is that they are modeled by computing and not communications contexts. In all three scenarios, we define the benefit of a resource $B(R)$ to be its processing power, and the cost $C(R)$ to be its access latency.

4.1 Enterprise

It is common that in an enterprise environment, mobile device users have access to high quality-of-service infrastructure of Wi-Fi network, as well as powerful backend servers. Such

servers could be maintained by the IT department of the enterprise itself, or leased from server farm providers such as the Amazon Elastic Computing Cloud[1]. In this scenario, $B(R)$ is high and $C(R)$ is low for the cloud resources. It is thus possible to fully offload some of the computing intensive tasks, like recognition and tracking to the cloud resources, and render the processed result on mobile device, making the latter a thin client (e.g. (Taylor & Pasquale, 2010)).

4.2 Hotspot

In some public locations, limited computing service might be provided by the venue management. For example, at a metropolitan airport, in addition to the complementary Wi-Fi service, computer nodes attached to the Wi-Fi access points could be offering users with their processing capacities complementarily or at a nominal charge. In some literatures, such enhanced network access points are called "Access Point 2.0". The characteristics of this scenario is that $B(R)$ has a moderate value, but $C(R)$ is also considerable. With the presence of "Access Point 2.0", the preferred approach is to rely on the on-device resources to perform tasks that require low latency, in the mean time utilize the cloud resources to perform tasks that are not sensitive to latency. For example, tracking could be conducted on-device, while recognition could be accomplished in the cloud. Because the latter is either conducted at application starting time or as a background task, latency introduced by using cloud-based resources is less detrimental.

4.3 At home

In the home environment, each family member could own one or more mobile devices. Additionally, these devices are capable of communicating with each other through ad-hoc or home Wi-Fi network, or other short range radios such as Bluetooth®. The resources provided by each mobile devices to others can be regarded as in the cloud from the other parties' point of view. In this scenario, $B(R)$ is low and $C(R)$ is high for the cloud resources, making them less popular than in the previous two scenarios. Cloud resources thus cannot play primary roles in computing, but only serve as auxiliary assistants. Possible usage of the CMC paradigm include assigning some of the off-line tasks in AR to cloud resources. Such kind of tasks include object database construction, feature updating in the database, and so on.

5. A proof-of-concept application

In this section we analyze an application in-depth to further study the CMC paradigm. This application employs vision-based techniques for natural human-computer interaction with the mobile device. As a result, it is subject to expensive computation of processing camera frames and track the hand in real-time, which exceeds the capability of the mobile device itself. We show that by using a computer cluster which is in the mobile device's proximity, the system performance can be speeded up and achieve significantly better usability. Abstract of this work was originally published in (Luo & Kenyon, 2009). This section is an extended version of the original publication.

[1] http://aws.amazon.com/ec2/

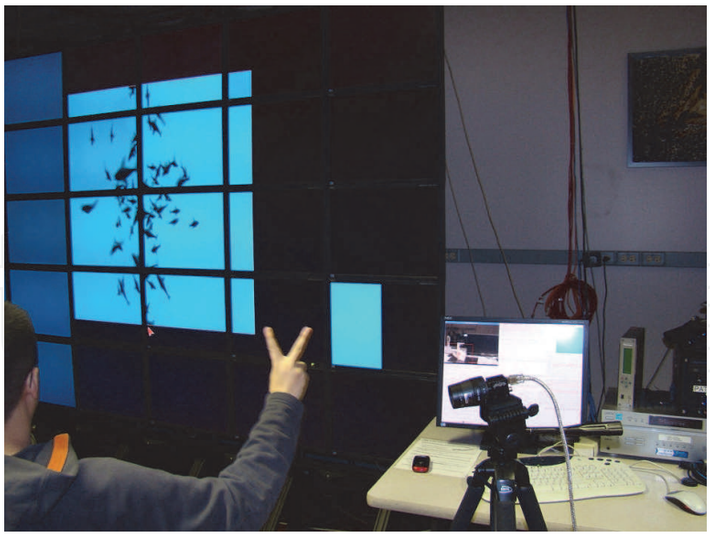


Fig. 2. A user interacts with a display wall with gestures, performing 2D window resizing and moving tasks. Video frames are captured by a single camera on the tripod but processed by multiple nodes of the cluster driving the display wall.

5.1 Background of the application

The last decade witnessed an increasing need for multi-million pixel resolution displays, propelled by applications such as scientific visualization, viewing satellite/aerial imagery, teleconferencing and a plethora of others. A successfully implemented solution to address this need is to construct display systems that consist of multiple screen tiles driven by a computer cluster. In such systems, each node in the computer cluster powers one or more of the display tiles. Building on this hardware configuration, the synergy among the cluster nodes ensures that pixel rendering across all the tiles are coordinated to provide the user with a large "single" screen experience. Cluster-driven large displays can be a projection-based wall(Mechdyne, 2008), flat panel-based wall (Jeong et al., 2006; Sandin et al., 2005; Sandstrom et al., 2003) or flat panel-based tabletop (Krumbholz et al., 2005). Two detailed reviews of recent progress on large displays can be found in (Wallace et al., 2005) and (Ni et al., 2006).

Vision-based gesture interaction methods have focused on the most dexterous part of the human body – the hand – so as to provide an intuitive means for human computer interaction (HCI). For a large number of applications that use large displays, these approaches can be especially advantageous compared to instrument-based counterparts under certain circumstances. For instance, interacting from-afar the screens will provide more freedom for the user to do arms-reach tasks. Similarly, during a tele-collaborative session, remote speaker's gestures can be automatically mapped to proper computer operation commands (e.g. playing forward or backward presentation slides), in this way the users' actions become intuitive and they need not resort to any additional devices. That said, gesture-based methods face a number of the performance challenges because of the requirements for low latency and high throughput needed for video processing. High latency causes lengthy response time and jeopardizes the user interactivity. On the other hand, low throughput forces the system to monitor a limited field of view, which in turn constrains the user's movement space.

A promising way to address the performance challenge is using scalable computing techniques to accelerate vision-based gesture interactions. This is based on the observation that large displays boast not only display size, but also computing power. The number of computer nodes in a cluster-driven large display system is usually considerable. In the mean time, the CPU performance of each node is growing exponentially with Moore's Law. For example, Varrier (Sandin et al., 2005), a 35-tile system uses an 18-node cluster. While Hyperwall (Sandstrom et al., 2003), a 49-tile system uses a 49-node cluster. In this section we describe the scalable computing techniques for vision-based gesture interaction that utilize the computing power inherent in cluster-driven displays. The processing of captured video frames from a single camera is parallelized by multiple nodes of the cluster. Consequently, we are able to achieve a lower processing latency and a higher processing throughput. When the user's hand is in the field of view of the camera, performance of predefined gestures is interpreted into interaction commands for the large display system. Figure 2 shows a prototypical vision-based gesture interface in use by a display wall, with the help of the proposed methods. Our work has three major contributions: first, it analyzes the task partitioning strategies for frame image scanning, a key task in both hand detection and gesture identification. By examining in detail the advantages and disadvantages of two strategies: by-region and by-scale, a hybrid strategy is developed by combining both of them. Furthermore, a novel data structure, named the scanning tree, is devised for computing nodes management. A load balancing algorithm for workload distribution across scanning tree nodes is also presented. Last but not least, implementation has been accomplished to incorporate the scalable computing techniques into a vision-based gesture interface for a ultra-high-resolution tiled display wall. Evaluation results show the effectiveness of our proposed techniques.

The rest of this section is organized as follows. Section 5.2 reviews related work in the existing literature. The by-region and by-scale task partitioning strategies are discussed in section 5.3. Based on the discussions, our final hybrid approach is presented. Section 5.4 describes the scanning tree data structure, together with the load balancing algorithm. Integration work of the prototypical vision-based gesture interface with the display wall is described in section 5.7. Section 5.8 illustrates the evaluation results which validate the effectiveness of the proposed techniques. At last, section 5.9 concludes the section.

5.2 Similar systems

Gesture-based HCI has been an active research topic for decades (Crowley et al., 2000; Kage et al., 1999). Due to hardware limitations, early systems used tethered mechanical or electromagnetic sensors to monitor hand gestures. For example, the MIT Media Room (Bolt, 1980) used an electromagnetical tracking sensor embedded in the wrist cuff to track spatial translation of the user's hand. The tracked gestures acted as an auxiliary control to move objects displayed on a projection screen. The CHARADE system (Baudel & Beaudouin-lafon, 1993) improved gesture detection by having the user wear a data glove. Finger segments positions reported by the glove are interpreted as gestures and then mapped to commands to control computerized objects. With the advances of camera hardware and CPU processing power, vision-based systems began to emerge for interaction with display systems. The early systems used simple features and could recognize a limited number of gestures. Freeman et al (Freeman & Weissman, 1995) used a video camera to detect hand movement. In their system only one hand gesture, i.e. the open hand facing the camera could be

identified. By waving the hand from side to side, the user was able to remotely control the sound volume of a television. Segen et al (Segen & Kumar, 1998; 2000) constructed a two-camera system to extract the fingertips of a hand from video, applied reverse kinematics to derive an articulated hand model, and subsequently used the derived hand posture for gesture-commands interpretation. Their system was fast (60Hz) because only the fingertips needed to be tracked. However it was also prone to low accuracy since reverse kinematics compute multiple hand postures for the the same fingertip configuration, which leaded to ambiguous gesture interpretations. A comprehensive survey of these systems can be found in (Joseph J. LaViola, 1999).

Besides the images of the hand itself, projected hand shadows could also be utilized for the recognition of certain gestures. The Barehands system (Ringel, 2001) used infrared LED arrays to illuminate a translucent touch screen from the rear. When the user's hand touched the screen a camera (with infrared filter) captured the hand's shadow and translated it into gesture commands. Image processing for Barehand was not computationally intensive, averaged 13.37 ms per frame. However, the needs for a touch-screen and high-intensity infrared illumination limited its use for large displays needing interaction from-afar.

HandVu (Kolsch et al., 2004) is a vision-based gesture HCI developed by Kolsh et al. It allows real-time capturing and processing of VGA-quality video frames, from a HMD-mounted camera. HandVu is capable of working under different illumination conditions as well as with various image backgrounds. Not surprisingly, such advantages are gained at the price of computation-intensive image processing on multiple features: shape, color, optical flow, and motion constraints. HandVu employs MPI(Message Passing Interface)-based parallel processing for off-line model training but not for online video processing. To optimize online processing performance, it takes several trade-offs in the implementation. For example, hand presence is only detected in a sub-region of the camera's field of view to initiate the interaction. In the mean time, when running under asynchronous mode where response time is guaranteed, unprocessed frames are discarded when timed out in a wait queue.

Researchers at the University of Toronto investigated interaction with large displays with pen-based (Cao & Balakrishnan, 2003) and gesture-based techniques (Malik et al., 2006). The work presented in (Malik et al., 2006) was a vision-based bimanual interaction system, which used two 320×240 cameras to track the user's fingers on a touch pad of black background. The system mapped a rich set of gestural input on the touch pad to interaction commands for a large display. Although detailed performance data was not available from the paper, the authors discussed that frame processing time prevented the deployment of higher resolution cameras. As a consequence, some users' object-manipulation experiences were undermined.

It can be learned from (Kolsch et al., 2004) and (Malik et al., 2006) that the performance of a single computer has already been one of the bottlenecks deterring the deployment of computation-intensive image processing methods for vision-based gesture interaction systems. Our scalable techniques are designed to address this challenge. The proposed solution partitions the processing workload of each captured video frame across multiple cluster nodes. Through such approach, it effectively improves the system performance.

Our solution complements other research work in the literature that uses computer cluster to speed up image processing. In the RPV (Real-time Parallel Vision) programming environment (Arita et al., 2000), three low level cluster computing functionalities, namely data transfer, synchronization and error recovery are taken care of by the environment, and the user can

use these functionalities directly instead of implement his/her own. An extension work called RPV-II (Arita & ichiro Taniguchi, 2001) introduces stream data transfer to RPV, thus the transmission latency is reduced. However, both RPV and RPV-II are only programming environments and put the responsibility on their users to design parallel algorithms for a specific computer vision application. A similar middleware is FlowVR (Allard & Raffin, 2006), which provides a data-flow model to encapsulate computation and I/O tasks for parallel image processing. Most recently, the GrImage platform (Allard et al., 2007) proposes a complete scalable vision architecture for real-time 3D modeling but not gesture interaction, which is the question to be addressed by this paper.

5.3 Design of task partitioning strategy

Vision-based gesture interaction can be modeled as a pipeline process that consists of four stages: detection, tracking, identification and interpretation. These stages can be disjointly sequential with each other or have overlaps depending on the specific system. Generally the pipeline runs as follows. First, *detection* senses the presence of the hand in the camera's field of view. *Tracking* is then conducted to monitor the hand position and report its spatial coordinates. Any user gesture that has matches an entry in the predefined gesture vocabulary is recognized by *identification*. As a final step, identified gestures are mapped to interaction commands through *interpretation*.

It would be impractical to devise a universal solution for all vision-based gesture HCI systems due to the large variety of implementations. To strike a balance between specificity and generality, we examine the group of systems that use template matching during the detection and identification stages. Such systems count for a considerable portion of the state-of-art vision-based gesture HCIs. In them each incoming frame from the camera is scanned to find possible matches with defined templates. If a match is found during the detection stage, the hand is regarded to be in the camera's field-of-view. Similarly, matches found during the identification stage are recognized as gestures performed by the user. From this point of view both the detection and identification stages can be abstracted to the same task of image scanning. The difference is that they might use distinct template databases. Section 5.3.1 elaborates on the image scanning task.

5.3.1 Analysis of the image scanning task

In the image scanning task, each sub-window of the frame image of the templates' size are compared with templates stored in the vocabulary database. If the similarity between the sub-window image and a template image satisfies a certain threshold, a template match is reported. To enumerate all sub-windows of the frame image, the task scans along X and Y dimensions in small steps until the whole image has been examined.

The size of the template remains constant once defined, but the size of the hand appearing in a frame image could vary dynamically with viewing depth. To address this, it is necessary for the image scanning task to do comparisons at multiple scales. Either the template or the captured image can be scaled for this purpose. Which one is scaled does not affect the overall computation complexity. Algorithm 1, which assumes that the captured frame image is scaled to multiple levels, illustrates an abstracted image scanning task.

Algorithm 1 shows that the image scanning task fulfills a group of comparisons between sub-windows of the captured frame image and the template. Because size of the template

Algorithm 1 An abstracted image scanning task

Input:
frame image: F.
template: T.
scanning translation steps: X_{step}, Y_{step}.
scanning start/stop scales: S_{start}, S_{stop}.
scanning scale step: S_{step}.

Output:
set of template matches: M.

```
1: M ← Φ
2: S ← S_start
3: while S < S_stop do
4:     Scale F at S
5:     X ← 0
6:     Y ← 0
7:     W ← width of F at scale S
8:     H ← height of F at scale S
9:     while X < W do
10:        while Y < H do
11:            Compare sub-window of F starting at (X, Y) with T
12:            if there is a match then
13:                Add the match into M
14:            end if
15:            Y ← Y + Y_step
16:        end while
17:        X ← X + X_step
18:    end while
19:    S ← S + S_step
20: end whilereturn M
```

is an invariant, processing load of each comparison (line 11 in Algorithm 1) is a constant. If this group of comparisons could be distributed evenly over multiple computing nodes, the image scanning task would be gracefully parallelized. In the next sections the two partitioning strategy candidates, by-region and by-scale, are discussed. The by-region strategy divides a frame image into multiple sub-regions, and assigns the processing of each sub-region to a computing node. The by-scale strategy assigns the processing of the whole frame image to computing nodes, but each of these nodes only handles certain scale levels.

5.3.2 The by-region choice

Figure 3 illustrates the principle of the by-region task partitioning strategy. Blocks of identical color indicate that they are processed by the same cluster node. With this strategy, a given cluster node is always assigned to process a fixed portion of the whole frame image, and processes this portion at all scales levels. For example, assuming that the frame image is captured at 640×480 resolution, and node A is assigned to process a sub-region with the corner coordinates to be [0, 0, 320, 240] (coordinates are in the form of

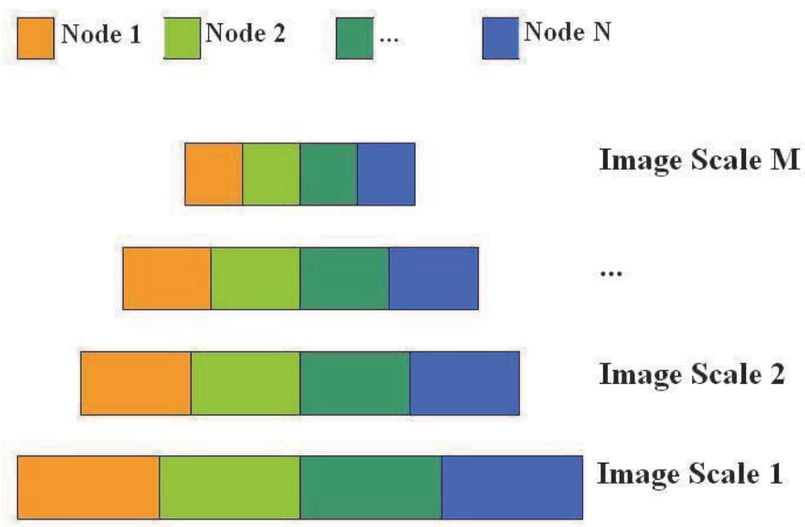


Fig. 3. Illustration of the by-region task partitioning strategy.

$[X_{left}, X_{right}, Y_{top}, Y_{bottom}]$) for the scanning scale of 1. When the scanning scale becomes 0.5, the frame image is reduced to 320 $\times$ 240, and the sub-region to which node A is assigned changes accordingly to [0, 0, 160, 120].

One advantage of the by-region strategy is straightforward load balancing: as long as the sub-regions assigned to different cluster nodes are of same size, workload distribution will be theoretically even. However, this strategy is subject to the drawback of high processing overhead. To illustrate this, consider the situation where the hand appears in the image area that crosses two sub-regions. If the two sub-regions are disjoint, the appearance of the hand will be missed by the image scanning task, causing reduced detection and identification rates. Thus it is necessary for the sub-regions to overlap with each other to make sure that sub-region borders are properly taken care of. The size of the overlapped area is solely determined by the size of the template. In Equation 2, the quantitative measure $R_{overhead}$ is defined as the overhead ratio. $R_{overhead}$ is calculated as the ratio between the size of the sub-region that a cluster node actually has to process versus the sub-region size that is assigned to it[2]. Table 1 shows how the overhead ratio values grow at several representative scanning scales.

$$R_{overhead} = \frac{(W_{subregion} \times S + W_{template})(H_{subregion} \times S + H_{template})}{(W_{subregion} \times S) \times (H_{subregion} \times S)} \quad (2)$$

5.3.3 The by-scale choice

Figure 4 illustrates the principle of the by-scale task partitioning strategy. The same legends for Figure 3 are used here. With this strategy, all cluster nodes are assigned to process the whole frame image, but each with a different range of scanning scales. For instance, for an

[2] For simplicity reasons, boundary conditions are not discussed for all equations and algorithms listed in this section.

Scanning Scale	Overhead Ratio
0.2	211%
0.3	170%
0.4	151%
0.5	140%
0.6	133%
0.7	128%
0.8	124%
0.9	121%
1	119%

Table 1. Overhead Ratio vs. Scales for By-region Strategy
($W_{subregion} = 320, H_{subregion} = 240, W_{template} = 25, H_{template} = 25$)

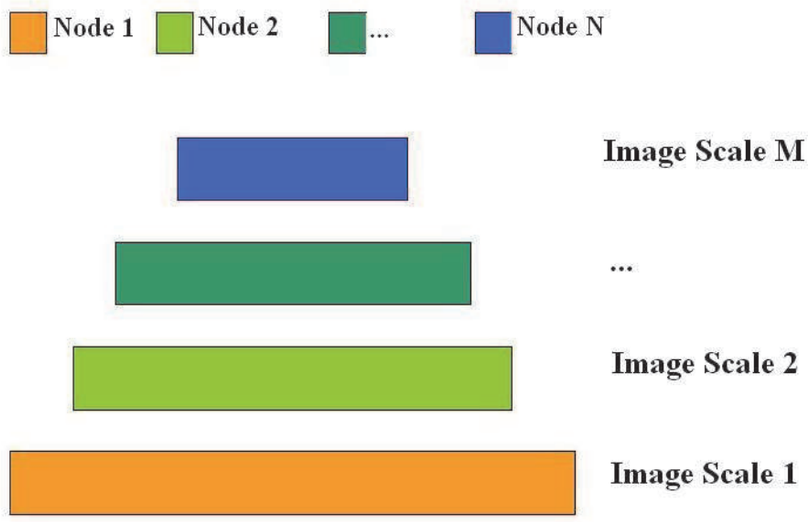


Fig. 4. Illustration of the by-scale task partitioning strategy.

image frame of 640 × 480 resolution, node A is assigned to process the scale levels within the range of 1-0.8. Node B is assigned to process the scale levels within the range of 0.7-0.5, and so on. The advantage of the by-scale strategy is that no processing overhead is introduced by sub-region overlapping, because each cluster node processes the whole image frame. However, there are unbalanced workloads at each image scale. When the image scanning task is performed at high scales, there are more sub-windows to compare. Consequently, the number of template comparisons is large. Similarly this number is small when the image scanning task is performed at a low scale. Because each template comparison takes the same amount of processing time, cluster nodes with different assigned scale ranges will have different workloads. Equation 3 presents the number of template comparisons needed at a given scanning scale S. With this equation, Table 2 shows that the number of comparisons needed increases at roughly the second order of the scanning scale. It is clear in the table that the highest number of comparisons (at scale 1.0) can be a magnitude higher than the lowest value (at scale 0.2).

Scanning Scale	Number of Comparisons
0.2	1219
0.3	3312
0.4	6430
0.5	10571
0.6	15736
0.7	21926
0.8	29138
0.9	37376
1	46638

Table 2. Number of Template Comparisons vs. Scales for By-scale Strategy ($W_{frame} = 640, H_{frame} = 480, W_{template} = 25, H_{template} = 25, X_{step} = 3, Y_{step} = 2$)

$$N_{comparison} = \frac{(W_{frame} \times S - W_{template})}{X_{step}} \times \frac{(H_{frame} \times S - H_{template})}{Y_{step}} \quad (3)$$

5.3.4 The final hybrid approach

Based on the previous discussions, it can be asserted that using either by-region or by-scale strategy alone will not be sufficient to achieve effective partitioning for the image scanning task. The by-region strategy creates high overhead for each single cluster node, and lowers the overall performance of parallel processing. On the other hand, the by-scale strategy overly engages the cluster nodes which scan at large scale levels, at the same time underly utilizes the cluster nodes which scan at small scale levels. The result is that parallel processing performance as a whole is affected.

We propose a hybrid approach that uses both by-region and by-scale task partitioning. The basic idea is to take advantage of the better load balancing inherent in by-region strategy, and lower the computation overhead, as much as possible, by exploiting the characteristics of by-scale strategy. More specifically, this approach works as follows:

- The captured image frame is distributed to all cluster nodes at the same level. Each cluster node is assigned to process the whole frame image at a single scale level, or within a certain scale level range.
- Scanning tasks at small scale levels, which are less computationally demanding, are grouped and assigned to a single cluster node. In this way the light workloads are aggregated to avoid under-utilized cluster nodes.
- A cluster node which has been assigned to scan at a large scale level further partitions its workload using the by-region strategy, and assigns the processing of sub-regions to next level cluster nodes.
- If the workload of scanning at a scale level is comparable to either aggregated small scanning scale levels or partitioned large scanning scales, it is assigned to a single cluster node.
- The partitioning process is repeated recursively until an optimized overall system performance is achieved.

Implementing this hybrid approach requires two additional components. The first component is a data structure that is capable of managing the cluster nodes involved in the image

scanning task with a layered architecture, because the hybrid approach partitions workload at multiple levels. Therefore, a data structure called a scanning tree is designed to handle this need. The second is a load balancing algorithm based on the scanning tree data structure. This algorithm is responsible for fine tuning three configuration tasks in the hybrid approach: workload grouping for scanning at small scale levels, workload partitioning for scanning at large scale levels, and the determination of scale levels that need neither grouping nor further partitioning. Section 5.5 introduces the scanning tree and Section 5.6 the load balancing algorithm.

5.4 Node management and load balancing

Designing of the hybrid task partitioning strategy lays out the foundation for a concrete implementation. One part of the implementation is the formation of the underlying system, i.e. management of the cluster nodes. The other part of the implementation is the synergistic scheme, of which the the load balancing algorithm is a main component.

5.5 The scanning tree

In our solution the scanning tree data structure is designed to manage the cluster nodes for the image scanning task. Figure 5 gives a graphical presentation of this data structure. The scanning tree can have one or more levels. Every node of the tree represents a cluster node and is indexed with a unique key, which is the node's IP address. Each edge of the scanning tree maps to a dulplex network link between the two connected nodes.

A node of the scanning tree has two data attributes: scan_region and scale_range. The scan_region attribute is a quadruple $[X_{left}, X_{right}, Y_{top}, Y_{bottom}]$ that specifies the unscaled sub-region coordinates to be scanned in the frame image. The scale_range attribute is a triple $[S_{start}, S_{step}, S_{stop}]$ that denotes the start, step and stop of scanning scales performed by the node on its scan_region. To make these descriptions more intuitive, two examples are given below, assuming that the frame image is of resolution 640×480:

- Node A has the index "10.0.8.120" and the data attributes {scan_region: [0, 0, 640, 480], scale_range [1.0, 0, 1.0]}. This is a cluster node with an IP address 10.0.8.120 which scans the whole frame image, at the single scale value of 1.0.
- Node B has the index "10.0.8.121" and the data attributes {scan_region: [0, 0, 320, 240], scale_range [0.5, 0.1, 0.8]}. This is a cluster node with an IP address 10.0.8.121 which scans the upper left quarter portion of the frame image, at four scales 0.5, 0.6, 0.7 and 0.8.

Each node of the scanning tree does three jobs: distributing the frame image to its children (with the leaf nodes being exceptions); scanning the specified sub-region at the specified scale_range; and reporting to its parent (with the head node being an exception) any template matches found by the subtree headed by itself. Under this mechanism, starting from the head node, a frame image is passed to all the nodes of the scanning tree in a top-down fashion. In a reverse flow fashion, template match results are gathered in a bottom-up manner and eventually converge at the head node.

The workload of a scanning tree node is determined by its data attributes. These data attributes can be manually planned based on the principles described in section 5.3.4, or constructed by an automated process as described in the next section.

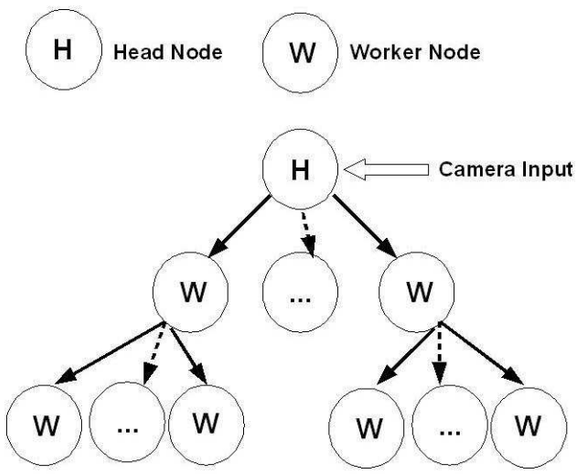


Fig. 5. The scanning tree.

5.6 Load balancing algorithm

Algorithm 2 automates the planning work for:

- **Construction of the scanning tree:** given a desired speedup P_{desire}, this algorithm constructs a scanning tree T_{scan}, the number of nodes in the scanning tree N_{nodes}, and an estimated best-effort speedup P_{est} to match P_{desire}.
- **Load balancing across the scanning tree nodes:** this algorithm distributes workload across scanning tree nodes as evenly as possible, using the hybrid task partitioning strategy.
- **Estimation of the communication costs:** this algorithm outputs the estimations for two communication costs: the cost at the head node C_{head} and the cost across the scanning tree $C_{overall}$. The unit of these costs is number of frame image copies transmitted.

Following the pseudo-code for Algorithm 2 below, the algorithm works as follows. It first calculates the number of template comparisons at each of the scanning scale values, and sums these numbers to get the total number of template comparisons needed for the image scanning task (lines 8-13). The template comparison numbers are then normalized as percentages of the totalled number. Besides the percentages array, an integral percentages array is also constructed, for each scanning scale value(lines 15-19). The classification of low, middle and high scanning scales is based on the desired speedup. According to the value of desired speedup, line 21 finds the value below which are treated as low scanning scales. Subsequently, line 22 finds the value beyond which are treated as high scanning scales. The values in-between fall into the category of middle scan scales. Once is classification is completed, the rest of the algorithm handles the processing for each class of scan scales respectively. All scanning scales in the low class are grouped to form a scan scale range. The first scanning scale in the middle class is handled by the head node. And the rest of each middle scanning scales is handled by a single node. For the processing of high scan scales, the by-region strategy is first applied and the algorithm is used recursively to construct a sub-tree for each high scanning scale (lines 34-36). Upon the creation of each tree node its workload

is defined, through the scan_region and scale_range attributes. When the algorithm finishes, the image scanning task has been partitioned to nodes across the constructed scanning tree.

Algorithm 2 Scanning tree construction and load balancing

Input:
desired speedup: P_{desire}.
width and height of the image frame: W_{frame}, H_{frame}.
width and height of the template: $W_{template}$, $H_{template}$.
scanning translation steps: X_{step}, Y_{step}.
scanning start/stop scales: S_{start}, S_{stop}.
scanning scale step: S_{step}.

Output:
scanning tree: T_{scan}.
number of nodes in the scanning tree: N_{nodes}.
estimated speedup: P_{est}.
estimated communication cost at head node: C_{head}.
estimated communication cost across scanning tree: $C_{overall}$.

$T_{scan} \leftarrow \phi$
$N_{nodes} \leftarrow 0$
$P_{est} \leftarrow 0$
$C_{head} \leftarrow 0$
$C_{overall} \leftarrow 0$

Populate $Array\_Scales$ with scale values from S_{start} to S_{stop} at interval S_{step}, record the size of $Array\_Scales$ as Len
Zeros $Array\_N_{comparison}$ of size Len
$N_{total} \leftarrow 0$
for (i = 1 to Len) **do**
    Calculate $Array\_N_{comparison}[i]$ using Equation 3, with the parameters $Array\_Scales[i]$, W_{frame}, H_{frame}, $W_{template}$, $H_{template}$, X_{step}, Y_{step}
    $N_{total} \leftarrow N_{total} + Array\_N_{comparison}[i]$
end for

Zeros $Array\_Pecentage$, $Array\_Grouped\_Pecentage$ of size Len
for (i = 1 to Len) **do**
    $Array\_Pecentage[i] = Array\_N_{comparison}[i] / N_{total}$
    $Array\_Grouped\_Pecentage[i] = \Sigma(Array\_Pecentage[1:i])$
end for

5.7 Display wall integration

A gesture interface is implemented using the OpenCV computer vision library, with KLT feature-based template comparison. The implementation uses parallel frame image. A baseline implementation that does not use the scalable computing techniques is also implemented for evaluation purposes. The gesture interface is then integrated with an

Algorithm 2 Scanning tree construction and load balancing (continued)

Find the highest index I_{group} that satisfies $Array\_Grouped\_Pecentage[I_{group}] \leq 1/P$
Find the lowest index I_{split} that satisfies $Array\_Pecentage[I_{split}] \geq$ to $1/P$

Create $Node_{head}$ with scan region to be [0,0,W_{frame},H_{frame}] and scanning scale value of $Array\_Scales[I_{group}+1]$
Add $Node_{head}$ to T_{scan} as head node

for (i_{middle} = $(I_{group}+2)$ to $(I_{split}-1)$) **do**
    Create a node $Node_{middle}$ with scan region to be [0,0,W_{frame},H_{frame}] and scanning scale value of $Array\_Scales[i_{middle}]$. Add $Node_{middle}$ to T_{scan} as a child of the head node
end for

Create a node $Node_{group}$ with scan region to be [0,0,W_{frame},H_{frame}] and scanning range of $[Array\_Scales[1], S_{step}, Array\_Scales[I_{group}]]$, add $Node_{group}$ to T_{scan} as a child of the head node
$P_{est} \leftarrow 1/Array\_Grouped\_Pecentage[I_{group}]$

for (i_{split} = I_{split} to Len) **do**
    Use the by-region strategy to partition the processing at $Array\_Scales[i_{split}]$ into a sub-tree, add this subtree as a child to the head node of T_{scan}
end for

$N_{nodes} \leftarrow$ total number of nodes in T_{scan}
$C_{head} \leftarrow$ number of children of the head node in T_{scan}
$C_{overall} \leftarrow$ total number of edges in T_{scan}
    return T_{scan}, N_{nodes}, P_{est}, C_{head}, $C_{overall}$

ultra-high-resolution tiled display wall. The wall has 55 LCD screen tiles, and is driven by a 32-node cluster. Each node in the cluster has a 64bit architecture with two 2GHz AMD processors and 4GB RAM. Nodes are interconnected with gigabit network interfaces. An open source high performance graphics streaming middleware SAGE (Jeong et al., 2006) is used by the display wall for frame rendering and UI management.

Hardware integration is performed as follows. The head node of the computer cluster is connected to a Dragonfly camera (Pointgrey Research Inc). The Dragonfly captures frame image at 640×480 resolution, 30 Hz rate and 8-bit gray scale. All nodes in the computer cluster participate visualization, coordinated by SAGE. A subset of them are used for the gesture interface under the management of a scanning tree. Regardless of the number of nodes in the scanning tree, it always uses the cluster head node as its head node. Communications among the scanning tree nodes are handled by the open source library QUANTA (He et al., 2003).

On the software integration side, the gesture interface runs as a standalone application, independent of the SAGE middleware. Interprocess communication using socket connection is setup between the gesture interface and SAGE. Whenever a user gesture is identified, the interface sends an interaction command to SAGE. SAGE then performs corresponding 2D window manipulation operations based on the interaction command. With this collaboration

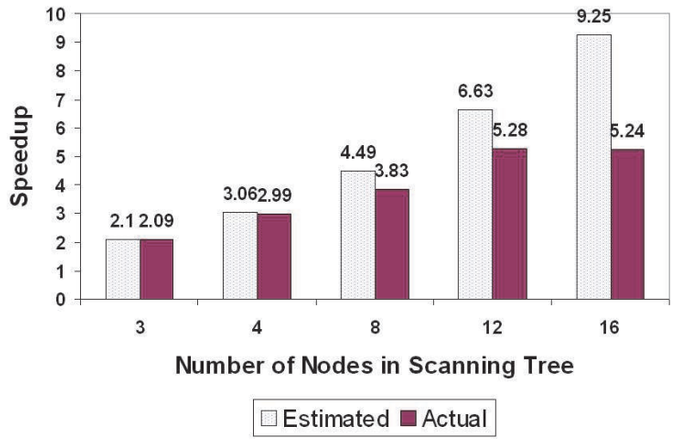


Fig. 6. Estimated vs. Actual Speedup Values.

the user can move, resize and close application windows displayed on the wall using free-hand gestures, as shown in Figure 2.

5.8 Evaluation results

A set of experiments are conducted with the gesture interface to evaluate the scalable computing techniques. These experiments measure three groups of metrics: speedup values under different scanning tree configurations, workload balance across nodes in the scanning tree, as well as performance impacts to the graphics streaming.

5.8.1 Speedup

Using the desired speedup values of 2.0, 3.0, 4.0, 6.0 and 9.0, algorithm 2 constructs five scanning trees. The node numbers in these trees are 3, 4, 8, 12 and 16 respectively. The estimated speedup values are 2.1, 3.06, 4.49, 6.63 and 9.25.

The times taken for frame image scanning during the detection and identification stages are profiled in the gesture interface. Same profiling are performed in the baseline implementation. Speedup values are obtained by calculating the ratio of averaged baseline processing times for image scanning over the counterpart in parallelized processing. Figure 6 shows the actual speedup values measured for the five scanning tree configurations described above. The measured speedup values are very close to the estimations for the 3-, 4- and 8-node scanning trees (difference is within 20%). For 12- and 16-node scanning trees the discrepancies are relatively large. This is mainly due to the increased communication cost during parallel processing. Communication costs are not taken into account in Algorithm 2. This fact leads to estimation errors.

5.8.2 Load balancing

Figure 7 illustrates the workload distribution estimated by the load balancing algorithm and the actual measured numbers, over a an 8-node scanning tree. Note the difference in units used by the top and bottom plots of figure 7. The estimated numbers are represented as the percentages of processing amount at all nodes against the processing amount of baseline

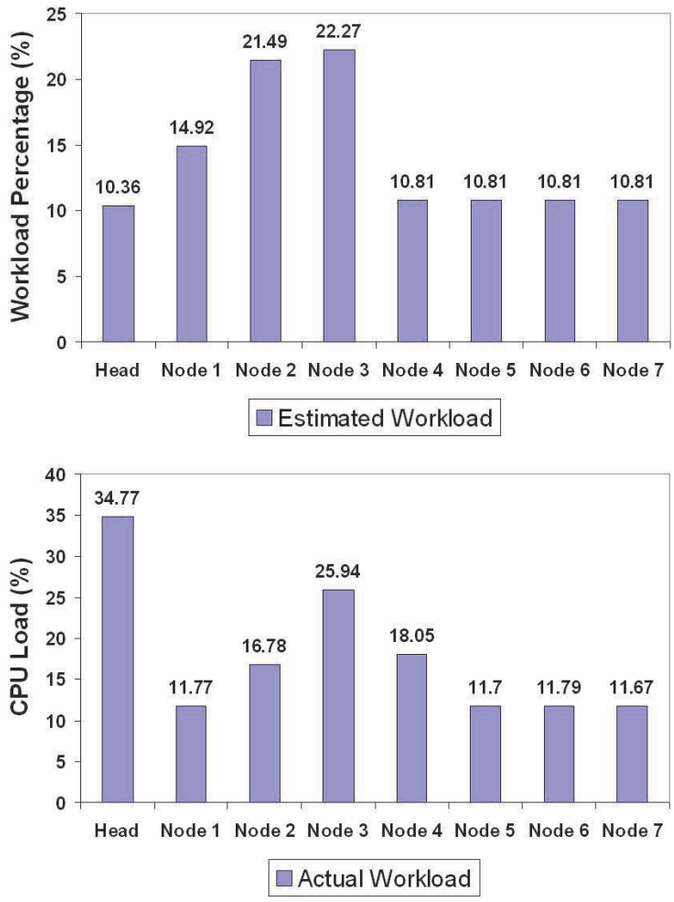


Fig. 7. Estimated and actual workload distributions across a 8-node scanning tree. Note that the units used by top and bottom plots are different.

implementation. In the mean time, the actual workload values are measured as average CPU load recorded at each of the nodes. Although the numbers in the top and bottom plots are not directly comparable, it can be seen that the actual workload distribution roughly follows the estimated outcome, with the exception at the head node. The larger readings at the head node is because that CPU load is measured as a whole for the gesture interface. Besides the work of image scanning, the head node is also in charge of video capturing, whose CPU usage is not easily separated when doing application-level profiling.

5.8.3 Performance impacts

Because the gesture interface shares the same computing resources with the SAGE middleware, its impact on the graphics streaming performance of SAGE is of interest. This experiment is conducted by running the OpenGL "Atlantis" application over SAGE, with the application window covering 20 display tiles. The profiling tools that come with SAGE is used to measure its display bandwidth and display frame rate. The display bandwidth reflects SAGE data throughput, which is calculated as the product of number of frames streamed in unit time and the data size of each frame. The display frame rate indicates the update rate

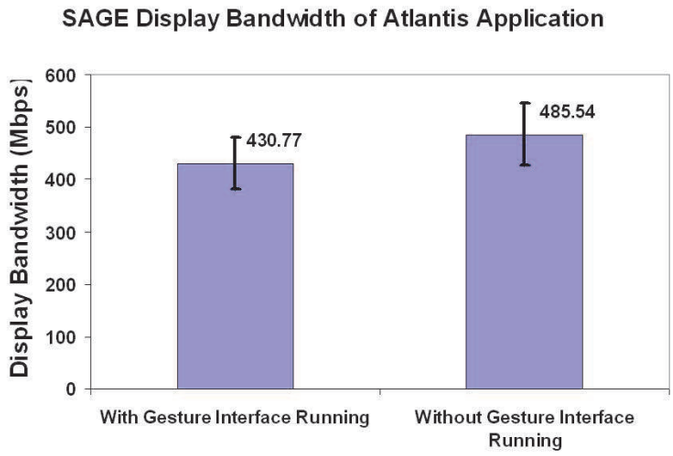


Fig. 8. Display bandwidths of the SAGE middleware, with and without the gesture interface running.

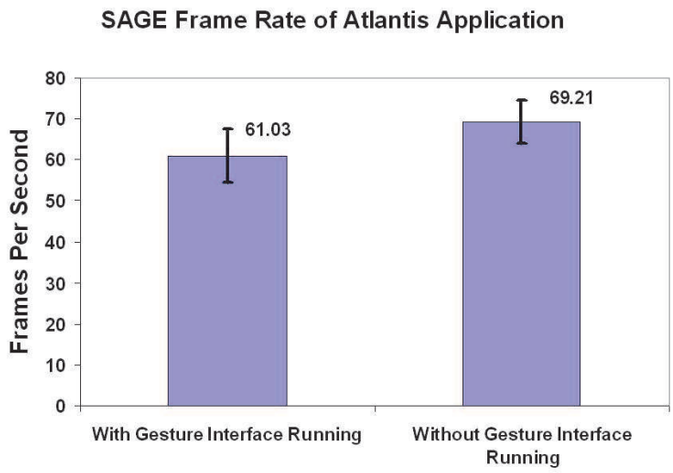


Fig. 9. Frame Rates of the SAGE middleware, with and without the gesture interface running.

SAGE could maintain across the display wall. Two conditions are tested, i.e. with and without the gesture interface running. For the "with gesture interface running" condition, a 12-node scanning tree is used for parallel processing. The nodes of the scanning tree belong to a subset of the nodes that drive the 20 LCD tiles where "Atlantis" is displayed.

Figure 8 shows the side-by-side comparison results of SAGE display bandwidth under two conditions. Without gesture interface in place, the display bandwidth is $485.54 \pm 59.74 Mbps$. While with gesture interface running, the display bandwidth is $430.77 \pm 49.84 Mbps$. Display bandwidth drops by 11% when the gesture interface is active. Similar results on display frame rates can be seen in figure 9. Without the gesture interface active the frame rate is 69.21 ± 5.26. With gesture interface active, frame rate becomes 61.03 ± 6.44. The frame rate decrease is about 12%. Considering the gain of a five-fold speedup for gesture detection and recognition, the performance impacts to graphics streaming are relatively insignificant.

5.9 Results discussion

This section explores the techniques to utilize the computing power of cluster-driven large displays to alleviate the performance needs for vision-based gesture interaction. It makes three contributions: 1) an analysis of the task partitioning choices, which leads to the proposition of a hybrid strategy, 2) the scanning tree data structure together with the load balancing algorithm, and 3) implementation and evaluation of the integration with a real system. Experimental results show that the proposed techniques are effective and exhibits promising speedup, estimation accuracy and performance impact metrics. To our best knowledge, this section is the first to address scalable gesture interaction for large display systems.

The solution presented can be applied to a broader range of computer vision interfaces beyond gesture-based systems, such as silhouette tracking. Being at the application-level, it is also feasible to be stacked on programming environment-level parallelization techniques and achieve further performance gains.

Presently communication factors, such as latency, overhead and bandwidth are not explicitly addressed in system modeling. As can be seen in section 5.8, this limitation can cause considerable estimation inaccuracy when data distribution is intensive. One extension could be the addition of several communication parameters in the LogP model (Culler et al., 1993) into the solution design. We leave it as a future work.

6. Conclusions and discussions

In this chapter we presented our work to enhance mobile AR system performance with a novel computing paradigm, named Cloud-Mobile Convergence (CMC). The design principle and philosophy of CMC is introduced, and several sample scenarios to put this paradigm into practical use are discussed. We also described in detail a real life application which adopts CMC paradigm. As a meta computing method, CMC exhibits potentials to be used in a wide range of mobile AR systems.

It should be pointed out that the research of using CMC for AR applications is still in its growing age. Solutions to some interesting problems remain largely unexplored. One of these problems is how to discover the cloud-based resources and configure the collaboration relationship between the mobile device and its cloud environment in an automated fashion. Up to now we always assumed that the mobile device had obtained information about its environment before conducting computation using CMC paradigm. We also assumed that the cloud-based resources were ready to be utilized as soon as requested by the mobile device. Both assumptions were simplified and might not be true under real world circumstances. For this, some well-established discovery and auto-configuration protocols, software implementations and standards may be integrated into a CMC application. Examples of such protocols, implementations and standards include Apple Inc's Bonjour (Apple, 2005), UPnP Forum's UPnP (UPnP, 2011), and Qualcomm Inc's AllJoyn (Qualcomm, 2011), to name just a few.

Another problem is inter-operability across heterogenous systems. When mobile device is collaborating with its cloud environment, they need to work out a solution to address possible heterogeneities at the levels of operating system, application programming interface, data format, and many others. To deal with the OS and API discrepancies, inter-process

communication methods such as RPC (Remote Procedure Call) may be useful. While data format differences need to be solved by techniques like interpretation middleware and application gateway. The importance of unification and standardization has been realized by the community, but to be able to get some real progresses there are still long roads to go.

In a broader sense, CMC paradigm belongs to the field of collaboration computing, and people had split views on collaborative computing. User experience had long been a controversial topic in the debates. Proponents argued that the performance gains collaborative computing brought to the table helped to achieve better user satisfaction. While the opponents believed that the discontinuity in service quality actually hampered user experience and so that collaborative computing would not be preferred by the mass users. With all these opinions said, there haven't been decisive and convincing evidences for either side found in the literature. To concretely evaluate the users' preference of CMC paradigm, extensive user study and human factors experiments should be conducted. We leave this as a potential future direction of the follow-up work.

Optimization using hybrid approaches, involving the CMC paradigm and methods discussed in section 2 would also be an interesting direction to explore. What if we reduce the computing problem complexity on the mobile device to get coarse tracking results, and refine it with the reminder of the computing task done outside of the device? What if we unify the parameter tuning both internal and external of the device, and make the cloud-based resources scheduling part of the parameter tuning process? Such hybrid approaches might very likely open new avenues to the performance improvement challenges. We will keep polishing the CMC paradigm to reflect the new findings and improvements.

7. References

Allard, J., Ménier, C., Raffin, B., Boyer, E. & Faure, F. (2007). Grimage: Markerless 3d interactions, *Proceedings of ACM SIGGRAPH 07*, San Diego, USA. Emerging Technology.

Allard, J. & Raffin, B. (2006). Distributed physical based simulations for large vr applications, *IEEE Virtual Reality Conference*, Alexandria, USA.

Apple (2005). Bonjour, http://www.apple.com/support/bonjour/, last accessed June 2011.

Arita, D., Hamada, Y., Yonemoto, S. & ichiro Taniguchi, R. (2000). Rpv: A programming environment for real-time parallel vision - specification and programming methodology, *IPDPS '00: Proceedings of the 15 IPDPS 2000 Workshops on Parallel and Distributed Processing*, Springer-Verlag, London, UK, pp. 218–225.

Arita, D. & ichiro Taniguchi, R. (2001). Rpv-ii: A stream-based real-time parallel vision system and its application to real-time volume reconstruction, *ICVS '01: Proceedings of the Second International Workshop on Computer Vision Systems*, Springer-Verlag, London, UK, pp. 174–189.

Baudel, T. & Beaudouin-lafon, M. (1993). Charade: Remote control of objects using free-hand gestures, *Communications of the ACM* 36: 28–35.

Bay, H., Tuytelaars, T. & Gool, L. V. (2006). Surf: Speeded up robust features, *In ECCV*, pp. 404–417.

Bolt, R. A. (1980). Put-that-there: Voice and gesture at the graphics interface, *SIGGRAPH '80: Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, pp. 262–270.

Cao, X. & Balakrishnan, R. (2003). Visionwand: interaction techniques for large displays using a passive wand tracked in 3d, *UIST '03: Proceedings of the 16th annual ACM symposium on User interface software and technology*, ACM, New York, NY, USA, pp. 173–182.

Crowley, J. L., Coutaz, J. & Berard, F. (2000). Perceptual user interfaces: things that see, *ACM Communications* 43(3): 54–58.

Culler, D., Karp, R., Patterson, D., Sahay, A., Schauser, K. E., Santos, E., Subramonian, R. & von Eicken, T. (1993). Logp: towards a realistic model of parallel computation, *SIGPLAN Not.* 28(7): 1–12.

DiVerdi, S., Wither, J. & Hollerei, T. (2008). Envisor: Online environment map construction for mixed reality, *Virtual Reality Conference, 2008. VR '08. IEEE*, pp. 19 –26.

Fragoso, V., Gauglitz, S., Zamora, S., Kleban, J. & Turk, M. (2011). Translatar: A mobile augmented reality translator, *IEEE Workshop on Applications of Computer Vision (WACV'11)*, Kona, Hawaii.

Freeman, W. T. & Weissman, C. D. (1995). Television control by hand gestures, *International Workshop on Automatic Face and Gesture Recognition*, pp. 179–183.

Harris, C. & Stephens, M. (1988). A combined corner and edge detector, *Proceedings of the 4th Alvey Vision Conference*, pp. 147–151.

He, E., Alimohideen, J., Eliason, J., Krishnaprasad, N. K., Leigh, J., Yu, O. & DeFanti, T. A. (2003). Quanta: a toolkit for high performance data delivery over photonic networks, *Future Generation Computer Systems* 19(6): 919–933.

Jeong, B., Renambot, L., Jagodic, R., Singh, R., Aguilera, J., Johnson, A. & Leigh, J. (2006). High-performance dynamic graphics streaming for scalable adaptive graphics environment, *Supercomputing '06*, IEEE Computer Society, Los Alamitos, CA, USA, pp. 24–32.

Joseph J. LaViola, J. (1999). A survey of hand posture and gesture recognition techniques and technology, *Technical report*, Providence, RI, USA.

Kage, H., Tanaka, K., Kyuma, K., Weissman, C. D., Freeman, W. T., Freeman, W. T., Beardsley, P. A. & Beardsley, P. A. (1999). Computer vision for computer interaction, *ACM SIGGRAPH Computer Graphics* 33: 65–68.

Kolsch, M., Turk, M., HÃllerer, T. & Chainey, J. (2004). Vision-based interfaces for mobility, *In Intl. Conference on Mobile and Ubiquitous Systems (MobiQuitous)*.

Krumbholz, C., Leigh, J., Johnson, A., Renambot, L. & Kooima, R. (2005). Lambda table: High resolution tiled display table for interacting with large visualizations, *Proceedings of Workshop for Advanced Collaborative Environments (WACE) 2005*.

Lee, T. & Hollerer, T. (2007). Handy ar: Markerless inspection of augmented reality objects using fingertip tracking, *Proceedings of the 2007 11th IEEE International Symposium on Wearable Computers*, IEEE Computer Society, Washington, DC, USA, pp. 1–8.
URL: *http://dl.acm.org/citation.cfm?id=1524303.1524850*

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints, *Int. J. Comput. Vision* 60: 91–110.

Luo, X. & Kenyon, R. (2009). Scalable vision-based gesture interaction for cluster-driven high resolution display systems, *Virtual Reality Conference, 2009. VR 2009. IEEE*, pp. 231 –232.

Malik, S., Ranjan, A. & Balakrishnan, R. (2006). Interacting with large displays from a distance with vision-tracked multi-finger gestural input, *SIGGRAPH '06: ACM SIGGRAPH 2006 Sketches*, ACM, New York, NY, USA, p. 5.

Mechdyne (2008). Powerwall, http://www.mechdyne.com/, last accessed September 2008.

Ni, T., Schmidt, G., Staadt, O., Livingston, M., Ball, R. & May, R. (2006). A survey of large high-resolution display technologies, techniques, and applications, *Proceedings of IEEE Virtual Reality 2006*, pp. 223–236.

Ozuysal, M., Fua, P. & Lepetit, V. (2007). Fast keypoint recognition in ten lines of code, *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pp. 1 –8.

Qualcomm (2011). Alljoyn, https://www.alljoyn.org/, last accessed June 2011.

Ringel, M. (2001). Barehands: implement-free interaction with a wall-mounted display, *ACM CHI*, Press, pp. 367–368.

Rosten, E. & Drummond, T. (2003). Rapid rendering of apparent contours of implicit surfaces for realtime tracking, *British Machine Vision Conference*, pp. 719–728.

Rosten, E. & Drummond, T. (2005). Fusing points and lines for high performance tracking., *IEEE International Conference on Computer Vision*, Vol. 2, pp. 1508–1511.

Sandin, D. J., Margolis, T., Ge, J., Girado, J., Peterka, T. & DeFanti, T. A. (2005). The varrier autostereoscopic virtual reality display, *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, ACM, New York, NY, USA, pp. 894–903.

Sandstrom, T. A., Henze, C. & Levit, C. (2003). The hyperwall, *CMV '03: Proceedings of the conference on Coordinated and Multiple Views In Exploratory Visualization*, IEEE Computer Society, Washington, DC, USA, p. 124.

Segen, J. & Kumar, S. (1998). Gesture vr: vision-based 3d hand interace for spatial interaction, *MULTIMEDIA '98: Proceedings of the sixth ACM international conference on Multimedia*, ACM, New York, NY, USA, pp. 455–464.

Segen, J. & Kumar, S. (2000). Look ma, no mouse!, *ACM Communications* 43(7): 102–109.

Shi, J. & Tomasi, C. (1994). Good features to track, *1994 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*, pp. 593 – 600.

Taylor, C. & Pasquale, J. (2010). Towards a Proximal Resource-based Architecture to Support Augmented Reality Applications, *Proceedings of the Workshop on Cloud-Mobile Convergence for Virtual Reality (CMCVR)*, Waltham, MA.

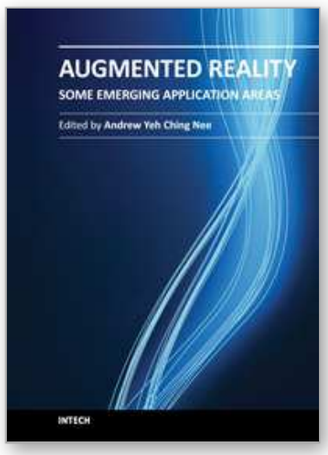
Terriberry, T. B., French, L. M. & Helmsen, J. (2008). Gpu accelerating speeded-up robust features.

UPnP (2011). Upnp, http://www.upnp.org/, last accessed June 2011.

Wagner, D., Schmalstieg, D. & Bischof, H. (2009). Multiple target detection and tracking with guaranteed framerates on mobile phones, *ISMAR*, pp. 57–64.

Wallace, G., Anshus, O. J., Bi, P., Chen, H., Chen, Y., Clark, D., Cook, P., Finkelstein, A., Funkhouser, T., Gupta, A., Hibbs, M., Li, K., Liu, Z., Samanta, R., Sukthankar, R. & Troyanskaya, O. (2005). Tools and applications for large-scale display walls, *IEEE Computer Graphics and Applications* 25(4): 24–33.

Wu, C. (2007). SiftGPU: A GPU implementation of scale invariant feature transform (SIFT), `http://cs.unc.edu/~ccwu/siftgpu`.



Augmented Reality - Some Emerging Application Areas

Edited by Dr. Andrew Yeh Ching Nee

ISBN 978-953-307-422-1
Hard cover, 266 pages
Publisher InTech
Published online 09, December, 2011
Published in print edition December, 2011

Augmented Reality (AR) is a natural development from virtual reality (VR), which was developed several decades earlier. AR complements VR in many ways. Due to the advantages of the user being able to see both the real and virtual objects simultaneously, AR is far more intuitive, but it's not completely detached from human factors and other restrictions. AR doesn't consume as much time and effort in the applications because it's not required to construct the entire virtual scene and the environment. In this book, several new and emerging application areas of AR are presented and divided into three sections. The first section contains applications in outdoor and mobile AR, such as construction, restoration, security and surveillance. The second section deals with AR in medical, biological, and human bodies. The third and final section contains a number of new and useful applications in daily living and learning.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen