A Streaming-Based Solution for Remote Visualization of 3D Graphics on Mobile Devices

Fabrizio Lamberti, *Member*, *IEEE*, and Andrea Sanna

Abstract—Mobile devices such as Personal Digital Assistants, Tablet PCs, and cellular phones have greatly enhanced user capability to connect to remote resources. Although a large set of applications are now available bridging the gap between desktop and mobile devices, visualization of complex 3D models is still a task hard to accomplish without specialized hardware. This paper proposes a system where a cluster of PCs, equipped with accelerated graphics cards managed by the Chromium software, is able to handle remote visualization sessions based on MPEG video streaming involving complex 3D models. The proposed framework allows mobile devices such as smart phones, Personal Digital Assistants (PDAs), and Tablet PCs to visualize objects consisting of millions of textured polygons and voxels at a frame rate of 30 fps or more depending on hardware resources at the server side and on multimedia capabilities at the client side. The server is able to concurrently manage multiple clients computing a video stream for each one; resolution and quality of each stream is tailored according to screen resolution and bandwidth of the client. The paper investigates in depth issues related to latency time, bit rate and quality of the generated stream, screen resolutions, as well as frames per second displayed.

Index Terms—Remote visualization, Chromium, MPEG, mobile devices, cluster-based rendering.

1 Introduction

3D graphics on mobile devices is a new and challenging task for researchers and developers. Two approaches are basically adopted to address this issue, namely, local and remote visualization. The former uses local device resources to display the 3D scene, while the latter uses remote hardware (i.e., graphics workstations or specialized clusters) to render the scene to be displayed on the screen of the mobile device; in this case, a network connection between the client (the mobile device) and the remote server has to be established in order to allow the transmission of images (from server to client) and commands (from client to server). Rendering large data sets, which can be required in a wide spectrum of disciplines including scientific simulation, virtual reality, training, and CAD, requires considerable computational power and storage resources; moreover, specialized hardware is often employed in order to allow interactive visualization of such complex scenes (in this context, the term interactive means that the user is able to smoothly navigate a 3D scene). Although nowadays mobile technologies allow the development of extremely attractive high-quality rendering applications (high resolution displays are available on many handheld devices) and some PDAs equipped with a graphics accelerator have recently appeared (i.e., the Dell Axim x51v), the smooth visualization of highly complex 3D objects modeled by millions of textured polygons and of data sets consisting of millions of voxels is still beyond the capabilities of portable devices. Moreover, situations exist in which sharing the data set with remote clients is not advisable for security reasons. Finally, often the possibility of using existing applications without any porting to another platform is a strict constraint. In all these cases, an approach based on an indirect rendering can be used. Here, remote resources are in charge of rendering the scene according to the view selected by the user at the client side; then, resulting images are transmitted to the user via a wireless channel. This solution can be defined as data independent since the data set is not moved from the server side. An additional advantage of the proposed approach is that it is also application independent, since a software porting of the existing application to the mobile device specific platform is not needed. However, this approach raises a set of issues related to channel bandwidth, manipulation latency, and resources necessary at the client side to decode compressed information.

This paper follows the remote rendering approach by proposing a three-tier architecture where a remote server (referred to below as *Remote Visualization Server* or *RVS*) is able to manage and render 3D models consisting of millions of polygons by using a cluster of Personal Computers (equipped with accelerated graphics cards managed by the Chromium software [1]). Frames to be computed are split among the graphics adapters, reassembled, and encoded into a video stream that can be simultaneously sent to multiple mobile clients via wireless channels. At the client side, users equipped with an ad hoc application (the *Mobile 3D Viewer*) can interact with the visualization interface in order to analyze the model within a cooperative session also involving the console at the remote server side. The proposed framework is able to support the visualization on "tiny" devices such as smart phones and PDAs as well as

• *The authors are with the Politecnico di Torino, Dipartimento di Automatica e Informatica, Corso Duca degli Abruzzi 24, Torino, I-10129, Italy. E-mail: {fabrizio.lamberti, andrea.sanna}@polito.it.*

Manuscript received 23 June 2005; revised 16 Jan. 2006; accepted 7 Feb. 2006; published online 10 Jan. 2007.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-0078-0605.

Tablet PCs and others clients (even not mobile) supporting larger resolutions. Roto-translation commands are coded and sent to the rendering server which is able to update the scene to be rendered in an interactive way. The server-side software can be transparently integrated within existing applications, thus enabling multiple mobile users to remotely control a variety of graphics applications based on the OpenGL libraries [2]. Results show that a 3D scene consisting of more than three million textured polygons (triangles) can be visualized at about 30 fps, at a resolution of 240 × 240 pixels, using four PCs, while eight PCs are needed to obtain the same frame rate at a double resolution. Moreover, experimental observations demonstrate that the system can scale for larger model sizes and higher client resolutions by adjusting the cluster size. The paper investigates in depth issues related to latency time (the delay between a command issued on the mobile device and the visualization of the updated scene), bitrate, and quality of the generated stream, screen resolutions, and frames per second displayed.

The paper is organized as follows: Section 3 reviews the main solutions to deliver 3D graphics on mobile devices while Sections 4 and 5 present all the components of the architecture server and client side respectively. Section 6 presents experimental results and an in-depth performance evaluation.

2 Background

2.1 3D Graphics on Mobile Devices: Local Rendering and Image-Based Remote Visualization

Some solutions were proposed to locally render 3D graphics on mobile devices. For instance, the PocketGL [3] is a 3D toolkit for PocketPC that consists of a source code containing numerous functions to manipulate a 3D display using GAPI. Although PocketGL is not identical to the popular OpenGL system for PC, there are enough similarities to allow many OpenGL tutorials to be used to assist in learning. More recently, two other technologies have been proposed: OpenGL ES (OpenGL for Embedded Systems) and M3D for the J2ME. OpenGL ES [4] is a low-level lightweight API for advanced embedded graphics using well-defined subset profiles of OpenGL. It provides a low-level applications programming interface between software applications and hardware or software graphics engines. Implementations of OpenGL ES include Vincent [5] and Hybrid's Gerbera [6]. Another open-source library similar to OpenGL is Klimt [7] (formerly known as SoftGL); Klimt is targeted for hardware independence and is available for many mobile platforms. On the other hand, M3D (Mobile 3D Graphics API) [8] for the J2ME (Java 2 Platform, Micro Edition) specifies a lightweight interactive 3D graphics API, which sits alongside J2ME and MIDP (Mobile Information Device Profile) as an optional package. The API is aimed to be flexible enough for a wide range of applications including games, animated messages, screen savers, custom user interfaces, product visualization and so on. The API is targeted at devices that typically have very little processing power and memory and no hardware support for 3D graphics or floating point math. However, the API also scales up to higher-end devices featuring a color display, a DSP, a floating point unit or even specialized 3D graphics hardware. M3G (JSR 184) [9] is built as a wrapper on top of the Hybrid OpenGL ES API implementation; M3G is a J2ME optional package that allows moderate complexity 3D graphics to be rendered at interactive frame rates on mobile devices. All the solutions mentioned above suffer from the poor computational capabilities of the considered devices. In fact, hardly any of the commercial PDAs and Tablet PCs available today can manage a million polygon or voxel dataset at interactive rates. As an example, we evaluated the rendering performance of two common PDA devices, the HP iPaq H5550 and HX4700 equipped with a 400MHz and 600MHz Intel XScale CPU and a 320 × 240 and 640 × 480 display, respectively. Models consisting of approximately five thousand polygons are rendered at a frame rate of less than six/seven fps on both devices, even though the screen resolution permits appreciation of significantly larger scenes. Among the alternative approaches, levels of details (LOD) and image-based rendering (IBR) [10] solutions should be mentioned. LOD techniques allow large models to be managed; however, scene complexity is still very limited [11]. IBR techniques are used in [12] to enhance 3D graphics on mobile devices; IBR techniques take a set of input images (key frames) and compute the in-between frames. The problem is how to place the camera in order to avoid artifacts; for this reason, IBR provides a feasible solution only for particular cases. An interesting approach which can be adopted to effectively bypass the limitations mentioned above is remote visualization.

The idea of dividing computational and visualization tasks is not new: A classification of existing techniques for rendering 3D models in client-server environments can be found in [13]. In particular, Silicon Graphics, Inc. has developed a commercial solution called Vizserver [14] that can be used to provide application transparent remote access to high-end graphics resources; moreover, Vizserver enables application-transparent collaborative visualization functionalities for multiple simultaneous users. A similar solution, called Deep Computing Visualization [15], has been proposed by IBM. Stegmaier et al. [16] take advantage of the transport mechanisms part of the X Window System [17] and they use a PocketPC version of VNC [18] in order to remotely control an X-server. This approach has two main advantages: it is application-independent and it does not require any changes in the interface; on the other hand, this strategy suffers from poor interactivity when limited bandwidth channels are used and performance is not optimized since it is not tailored for any specific application. VirtualGL [19] is an improved version; rather than composite the rendered pixels directly into an X window, it compresses them using a high-performance JPEG codec and sends them to a listener daemon running on the client machine; in [20] different image compression schemes are also evaluated. Engel et al. [21] proposed a framework for interactive remote visualization written in Java in order to provide a platform independent tool. A similar solution tailored for low bandwidth as well as low resolution devices has been presented by Beerman in [22]. A grid architecture based on Web Services is used in [23] to allow thin mobile devices to display large data sets; performance of this solution is strongly related to the bandwidth available to interconnect all rendering resources of the grid. A different approach is used in [24] where a remote server manages 3D models extracting 2D feature line primitives that are sent to the mobile device for rasterization. Lamberti et al. [25] proposed a solution able to use hardware-accelerated remote clusters where the computational task of an OpenGL application is split among graphics adapters (GPUs) by means of the Chromium architecture. The contribution of each GPU is then reassembled and sent to the PDA client as a flow of still images.

2.2 Streaming-Based versus Image-Based Solutions

The solution proposed in this paper differs greatly from [25]. Although the work presented in [25] allowed the user to experience quite a smooth navigation (about 7 fps can be obtained on a Compaq iPaq H3630 PDA at a resolution of 120 $\times$ 120 pixels independent of scene complexity, assuming a sufficient number of cluster nodes is available) some problems still had to be tackled. In [25], a remote rendering server based on Chromium was in charge to compute the scene (and this is the only part shared with the proposed solution). After that, frames were managed as a flow of still pictures and sent to the client in a compressed or uncompressed format. Compressed images have the advantage of making better use of the bandwidth, but require extra time at the client side to be decompressed. This overhead is negligible for desktop technologies, but it can introduce an unacceptable delay when mobile devices such as PDAs and smart phones are used. Uncompressed frames can be directly displayed by a mobile device but a large bandwidth is required and this limits the application of the solution proposed in [25]. In this paper, a completely redesigned architecture is proposed and significantly improved performance is obtained. At the server side, a cluster of PCs driven by the Chromium software is used to produce an MPEG stream; at the client side, a new application capable of decoding/displaying the received stream has been implemented. This framework allows any OpenGL application to be used on the proposed system. Nevertheless, the user interface should be changed and adapted to be displayed on low resolution screens (while other solutions, like that proposed in [16], allow the output of any application to be delivered without changing the interface). Thus, applications presenting buttons, sliders, check boxes, and so on, could be managed building a sort of low level wrapper able to "translate" the position of any pen tap on the client side screen to the suitable mouse position on the remote console, but this mechanism would have to be customized for the specific interface. At the present time, the proposed framework allows OpenGL-based programs having an interface mainly based on events generated by keyboard and mouse to be used into a mobile scenario with very little effort. The architecture presented in this paper allows frame rates up to 30 fps at a 240 $\times$ 240 resolution to be obtained on a PDA even over 50 Kbit/s wireless communication channels. On the other hand, new issues related to the overall latency in scene manipulation can arise; these problems have been investigated in depth by providing an exhaustive performance analysis as well as an evaluation of the quality-related parameters such as the encoding bitrate. A first attempt to perform visualization on mobile devices through a streaming-based solution was already proposed in [26], but a rigorous analysis of system performance is missing. A rendering application, tailored to manage medical models, is remotely executed on a cluster, and the resulting images are delivered to a single client device using an encoded video stream. Nevertheless, because of the sequential approach being adopted in the rendering/encoding/streaming architecture, the solution presented in [26] is characterized by extremely poor performance that today can be achieved without resorting to a remote visualization approach. Some of the issues mentioned above have been addressed also by *mental images* in its RealityServer architecture [27]. RealityServer is a server-based platform allowing 3D contents developed using CAD/CAM packages to be navigated by remote client devices equipped with a Web browser that supports Flash or DHTML and JavaScript. Nevertheless, 3D contents imported into RealityServer are static and existing graphics applications possibly designed to manipulate the scene cannot be reused. To allow interaction and customization, developers must write new application logic using a JavaScript based API. RealityServer supports both client-side rendering (CSR) and server-side rendering (SSR). In CSR, a Web browser plugin named RealityPlayer receives and manages the rendering of automatically-created reduced representations of the original 3D content generated by the RealityServer. The system can additionally be deployed so that the server renders a photorealistic version of the original model and sends it to the client for display after scene manipulation. However, it is worth remarking that using techniques based on decimation the best visualization quality can be achieved only after 3D content manipulation and the degree of interactivity strictly depends on the graphics capabilities of the client device. Such techniques are in fact characterized by many of the drawbacks illustrated in Section 3.1 and are therefore out of the scope of this paper. In SSR, only a sequence or encoded stream of rendered images is delivered to front-end client devices. This allows in principle the integration with any kind of client device. However, even if rendering and streaming can be processed in real time in certain rendering modes, bandwidth restrictions may adversely affect interactivity, especially on mobile devices. In fact, according to [28], the visualization of a 3D content on a smart phone/PDA at a resolution of 320 $\times$ 240 at 10 frames per second requires a constant bandwidth around 360 Kbit/s that largely exceeds even the average data rate of UMTS. Moreover, a 720 Kbit/s throughput is needed to handle scene manipulations at 20 fps. This could constitute a serious bottleneck to the effective application of such an architecture in the mobile environments considered in this paper. Nevertheless, it has to be observed that when the mobility constraint is removed, the availability of large bandwidth communication channels supporting a 20 Mbit/s throughput allow to achieve interactive 3D content visualization at 20 fps even on 1,920 $\times$ 1,140 displays, thus making RealityServer a valid alternative to the solution proposed in this paper in nonmobile scenarios.

3 Server-Side Remote Rendering Subsystem

3.1 The Remote Visualization Server (RVS)

The development of the server-side remote rendering subsystem has been based on a software component, the *Remote Visualization Server* (*RVS*), which controls the communication between the 3D graphics application running in a distributed environment at the remote site and the visualization interface deployed on mobile devices. By exploiting the functionalities provided by the *RVS*, an OpenGL application can be transparently controlled by simultaneous remote users in an interactive and collaborative way. Basically, the internal operation logic driving the *RVS* can be described as follows: A newly generated scene manipulation event on the client mobile device triggers the submission of a command packet describing the event to the *RVS* over a wireless link. Once this packet reaches the *RVS*, it is translated into an application-compliant semantic and then passed on to the proper OpenGL callback functions which in turn will adjust mapping and rendering parameters. The OpenGL directives are then locally executed. Once the rendering has been completed, the frame-buffer is copied in the main memory. The *RVS* encodes the raw image data in a suitable format for distribution to mobile devices. The characteristics of the data format need to be tuned according to the specific communication channel and client device being used. Finally, encoded data are decoded and displayed to the remote user on the mobile visualization device. *RVS* is also responsible for managing multiple scene manipulation commands coming from the console as well as from mobile users. For this, a scheduling system has been developed which handles concurrent manipulation events on a first-come first-served basis. The scheduling system is also responsible for notifying the users of the actual client who is allowed to carry on operations over the scene. All these steps contribute to determining the overall latency between manipulation and display update.

3.2 Distributed Cluster-Based Rendering Environment

In deployment of the rendering subsystem, we exploited a cluster-based architecture in order to distribute the execution of OpenGL rendering commands over a cluster of PCs. In fact, despite recent advances in accelerator technology, many interactive graphics applications still cannot run at acceptable rates. As processing and memory capabilities continue to increase, so do the sizes of data being visualized. Because of memory constraints and lack of graphics power, visualizations of this magnitude are difficult or impossible to perform, even on the most powerful workstations. All these issues intrinsically impose a limit to the degree of interactivity that can be achieved on the mobile device. Recently, clusters of workstations have emerged as a viable option to alleviate this bottleneck. Thus, a cluster-based rendering subsystem has been deployed using Chromium.

The Chromium framework provides a software architecture that unifies the rendering power of a collection of graphics accelerators in cluster nodes, treating each separate frame-buffer as part of a single tiled display. Chromium provides a virtualized interface to the graphics hardware through the OpenGL API. A generic Chromium-based rendering system consists of one or more clients submitting OpenGL commands simultaneously to one or more graphics servers. Each server has its own graphics accelerator and a high-speed network connecting it to all clients, and is responsible for rendering a part of the output image which is later reassembled for visualization. Existing OpenGL applications can use a cluster with very few modifications, because Chromium relies on an industry standard graphics API that virtualizes the disjoint rendering resources present in a cluster, providing a single conceptual graphics pipeline to the clients. In this way, even large data set models, that could not be otherwise interactively rendered using a single machine, can be easily handled. The Chromium framework follows a traditional client-server paradigm in which OpenGL directives are intercepted by client nodes which locally run graphics applications and successfully distribute rendering workloads to high-end processing servers. Three-dimensional data sets are therefore decomposed into N parts which are rendered in parallel by N processors. The resulting planar images are reassembled by clients which are in turn responsible for producing the final image. Both clients and servers manage OpenGL streams by means of Stream Processing Units (SPUs), which are software modules implemented as dynamically loadable libraries responsible for supporting graphics context subdivision in tiles, distributing computing workloads to cluster nodes (Tilesort SPU), performing actual rendering on local graphics accelerators (Readback SPU), returning the result to Chromium clients (Send SPU), and reassembling image tiles (Render SPU). Native Chromium SPUs have been modified and new SPUs have been designed and integrated in the proposed framework in order to meet the requirements of remote visualization on mobile devices. In particular, a novel SPU called *RenderVideo SPU* (illustrated in Section 4.3) has been designed in order to extract from the framebuffer image frames resulting from the rendering process, and distribute such a frame-based raw video sequence to specific components which actually convert it into a suitable format. Specifically, two additional modules, referred to below as *Encode Server* and *Streaming Server* (described in Sections 5.4 and 5.5) have been deployed: these modules are responsible for image frames MPEG encoding and frame-based video sequence streaming to remote visualization clients respectively. Multiple instances of these modules can be allocated in order to manage different types of mobile devices. In Fig. 1, the overall architecture of the proposed remote visualization framework is presented. Mobile client devices and *RVS* start a visualization session through a handshake protocol. During this phase a device communicates to the *RVS* a set of parameters needed to set up the connection (resolution, bitrate and quality); the *RVS* uses these parameters to configure the encoding and streaming modules. Thus, a common collaborative visualization session can be shared among the visualization console and the remote devices.

The need to handle multiple user interaction involves a problem related to scene management: only one user at a

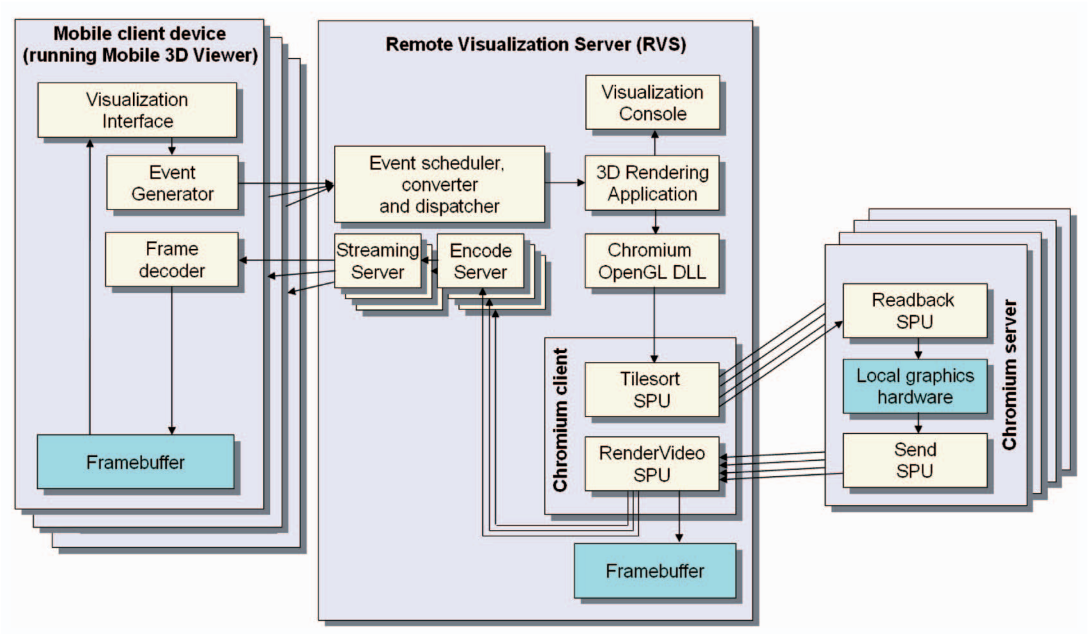

Fig. 1. Layout of the proposed three-tier architecture. One or more mobile client devices can remotely control a 3D graphics application by interacting with a middleware layer (based on the RVS) that is responsible for handling distributed rendering on a cluster of PCs using Chromium. Image frames generated by the RenderVideo SPU are encoded into multiple video sequences (using different encoding parameters) by the Encode Server components and finally streamed to heterogeneous remote clients over multicast wireless channels by the Streaming Server components.

time has to be able to send commands to the rendering system, while events generated by the other clients have to be ignored. A simple "token" protocol has been implemented. Only the commands received from the user getting the token are processed while the others will be discarded. The first client starting a visualization session obtains the token and all the other clients joining the session will only be able to display the stream coming from the *Streaming Server* related to their own device type. Each client can send two commands labeled "ask for the token" and "release the token." When the client getting the token sends to the *RVS* the release token command, the *RVS* gives the token to the first client that requires the session control. A precomputed stream is used to inform clients that the token is available and an online computed stream is sent to communicate the new token assignment. At the moment, a priority client management is not implemented; moreover, all token requests arriving at the *RVS* before the token is released are lost.

3.3 Generation of a Rendering-Based Raw Video Sequence for Distribution: The RenderVideo SPU

A specific Chromium SPU called *RenderVideo SPU* has been designed in order to manage the generation of 2D contents suitable for effective interactive distribution in a remote visualization environment. The newly developed SPU exploits all the main functionalities of the original Chromium Render SPU which is responsible for reassembling tiles received from Chromium server nodes as well as for writing reassembled frames into the framebuffer (thus, enabling the visualization on the console). Additionally, the *RenderVideo SPU* is capable of extracting image data resulting from the rendering of current frames from the framebuffer using `glReadPixel()`, and of serially distributing multiple copies of the newly obtained frame-based raw video sequence to specialized components, which in turn perform video encoding and streaming.

3.4 Encoding of Frame-Based Raw Video Sequence: The Encode Server Component

The *Encode Server* component incorporates MPEG video processing capabilities and provides the requested support for efficient compression of the rendering-based frame sequence. This module receives a frame-based raw video sequence from the *RenderVideo SPU* and it is responsible for scaling frames to a specific resolution, converting resized frames into the YUV 4:2:0 format suitable for MPEG processing, performing MPEG encoding of incoming frames—thus, generating an MPEG ES (Elementary Stream) bitstream [29]—and, finally, passing encoded MPEG video to a streaming application (the *Streaming Server*). Scaling is necessary since a high resolution rendering has to be produced to manage the visualization on the remote console. The *Encode Server* has been developed as a separate application which receives contents to be encoded over a TCP connection. This allows for the application to be run on a different machine, thus limiting the overhead on the Chromium client. Currently, the *Encode Server* makes use of the open-source `libavcodec` library [30] in order to generate an MPEG ES stream. In the encoding process, all typical MPEG parameters can be adjusted in order to precisely control resolution, bitrate and quality of the generated stream. Thus, multiple instances of the *Encode Server* can be allocated, each generating a video stream using different parameters to fit networking and visualization capabilities of the remote device. The additional advantage of the modular approach introduced above is that the *Render Video SPU* and the *Encode Server* can work in parallel and a pipeline effect can be obtained. In this way,

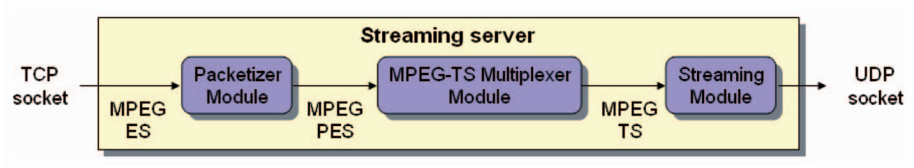

Fig. 2. Structure of the Streaming Server component.

during the encoding of a frame, the next rendering cycle is being carried out by *Render Video SPU*. For this, in computation of the frame rate which can be achieved in the video stream generation process, only the duration of the longest phase has to be considered.

3.5 Streaming Rendering-Based Encoded Video to Mobile Client: The Streaming Server Component

Streaming of MPEG encoded frames generated by an *Encode Server* module is delegated to its associate *Streaming Server* component. The *Streaming Server* is responsible both for performing several preprocessing tasks on the encoded bitstream, and streaming the resulting MPEG video sequence to remote clients. By keeping frames encoding and video sequence streaming separated, it has been possible to design a lighter *Encode Server* component only responsible for MPEG encoding of rendered frames and a highly specialized unit for remote visualization oriented streaming. Encoded video frames are streamed to remote clients using MPEG TS (Transport Stream). MPEG TS is defined in MPEG-2 specifications [31] and provides effective support for interactive transmission and visualization of multimedia contents over unreliable computer networks. In MPEG TS, essential synchronization information usually needed in multimedia streaming applications are passed by means of time stamps; a sample of the encoder's local time is included in the bitstream to allow synchronization of the decoder. In this way, streamed data incorporate sufficient information to carry out synchronization between encoder and decoder, and data can be streamed directly on a UDP socket without introducing the overhead of additional protocols. Nevertheless, in the future, the effects of the introduction of alternative synchronization protocols (like RTP, for example) are expected to be evaluated. Synchronization relies on timing information inserted into each video data unit header including PCR (Program Clock Reference), PTS (Presentation Time Stamp), and DTS (Decoding Time Stamp). Periodical transmissions of PCRs enable continuous synchronization of the receiver System Time Clock (STC) with the transmitter STC. PTS indicates the time at which an encoded picture should be removed from the receiver buffer, instantaneously decoded and presented for display. DTS indicates the time at which an encoded picture should be instantaneously removed from the receiver buffer and decoded (it differs from the PTS only when picture reordering is used for bidirectional predicted MPEG pictures). The synchronization scheme provided by MPEG TS allows for correct handling of out-of-order or delayed and possibly corrupted data delivery at the receiver.

The *Streaming Server* is based on open-source components and comprises three modules (see Fig. 2), namely, the *Packetizer* module, the *MPEG-TS Multiplexer* module, and the *Streaming* module. The first two modules deal with preprocessing operations on MPEG encoded frames which are necessary in order to obtain MPEG TS data suitable for transmission, while the latter one deals with video data transmission to remote clients. In particular, the *Packetizer* module receives an MPEG ES (Elementary Stream) from an *Encode Server* module (possibly running on a different machine) and is responsible for building an MPEG PES (Packetized Elementary Stream) [29]. This operation consists in splitting a single ES into variable length packets including the DTS and PTS fields. In the *MPEG-TS Multiplexer* module, the MPEG PES stream is finally converted into MPEG TS transport units. PES packets are split into MPEG TS fixed length data units ready for transmission. PCR information is periodically generated and inserted into MPEG TS packets. Resulting TS packets are placed in a FIFO output buffer. The *Streaming* module is then responsible for extracting TS packets from such a buffer and for writing them on a multicast UDP socket, thus implementing simultaneous streaming to mobile clients.

4 Client-Side Visualization Application: Mobile 3D Viewer

In the proposed remote rendering scenario, mobile devices like PDAs and Tablet PCs running a dedicated application called *Mobile 3D Viewer* only act as visualization front-ends supporting user interaction with a virtual 3D scene which is actually produced (and possibly displayed) at a distance. The user can interact with the actual device by tapping a pen over the display and using a device-specific directional pad and/or customizable buttons. Tablet PC devices have been selected for their higher screen resolution in comparison to common handheld devices. In this way, usability and performance on the next generation high resolution PDAs allowing the user to appreciate ever more complex geometries can be estimated. *Mobile 3D Viewer* has been deployed as an ad hoc application using Gtk+ 1.2 and X11 graphics libraries under Linux, and has been extensively tested on HP iPaq H5500 PDAs running Familiar Linux 0.6.1, and on a Compaq T1000 Tablet PC running Linux RedHat 8.0. *Mobile 3D Viewer* has been also evaluated in a Microsoft Windows XP-based native installation on Tablet PC. Native languages of the platforms under consideration have been used (giving up of the portability guaranteed, for example, by the Java language) in order to optimize use of the limited available resources, and speed up performances. Following a highly modular approach, the developed mobile application is based on three software components (see Fig. 1): the *Visualization Interface*, the *Frame Decoder*, and the *Event Generator*.

4.1 The Visualization Interface

The *Visualization Interface* is responsible for managing user interaction with the scene currently displayed on the mobile device as well as with the remote 3D application: It handles basic events generated by user input devices and passes them to the *Event Generator* module. A series of control buttons below the render area allows the user to interactively manage a subset of the functionalities offered by the remote OpenGL application. In the future, a complete support for application-specific commands will be provided. Furthermore, information related to the current scene, including polygon count, volume data size, and frame rate, are displayed to the user. Finally, the *Visualization Interface* is also responsible for supervising the render area for framebuffer content received from the remote server (which actually performs the rendering operations).

4.2 The Frame Decoder

The *Frame Decoder* receives an encoded MPEG TS video stream from the remote *Streaming Server* over a multicast UDP channel established on a wireless link. Experiments have been carried out in both local (IEEE 802.11b) and geographic (2.5G/3G) wireless communication environments. The *Frame Decoder* extracts synchronization information embedded in the incoming bitstream, decodes compressed MPEG frame according to the DTS time stamp, writes decoded images into a visualization buffer, and displays ready images according to the PTS time stamp. The *Frame Decoder* is also responsible for correctly handling packet losses, which commonly occur in wireless networking environments. This task is accomplished by replacing the missing picture with the most recently decoded MPEG frame. *Mobile 3D Viewer* MPEG video processing capabilities rely on a general purpose open-source multimedia player developed within the Videolan project [32].

4.3 The Event Generator

The *Event Generator* is responsible for receiving information concerning the events generated at the user interface and converting them into suitable commands for the remote application. Events generated at the user interface concern roto-translation commands, in order allow the user to navigate the scene, and commands for token management. Such commands are then encoded in a format suitable for transmission through a TCP connection over low bandwidth communication links.

5 Experimental Results

Two different usage scenarios have been used to test the proposed architecture: a volume renderer used to interactively explore 3D texture-based volumetric data sets, and a general purpose surface renderer for generic 3D scene navigation developed at the Politecnico di Torino University. The volume rendering application is XMLViewer, a tool included in the SGI Volumizer package [33]; this software provides a direct volume rendering method based on 3D textures. For the sake of completeness, it is worth outlining that XML Viewer is not optimized to be used with Chromium.

We evaluated the effectiveness of the proposed framework in remote visualization scenarios enabling mobile users endowed with PDA and Tablet PC devices to simultaneously interact with the above 3D rendering applications running on an accelerated graphics back-end. The main aim is to prove that remote visualization based architectures, like the one presented in this paper, allow for highly interactive collaborative sharing of realistic virtual worlds, between a high-end visualization console and multiple mobile users, regardless of the complexity of the scene being considered.

Since visualization frame rate and overall latency experienced at the mobile client both constitute the main limitations of existing remote rendering architectures, especially when considering collaborative visualization sessions, an event-driven analysis system has been designed in order to accurately quantify critical parameters of our remote visualization system, thus providing an effective measure of the interactivity of the proposed architecture. For each rendering cycle, a distributed measurement environment that involves almost all the components of the designed architecture allows recording of the amount of time needed for: rendering current frame (t_{render}), producing an MPEG encoded frame (t_{MPEG}) and displaying an MPEG encoded frame on the mobile device ($t_{streaming}$). It should be noted that, when cluster-based rendering is considered, t_{render} includes the time needed for subdividing graphics context into tiles, distributing workload to cluster nodes, performing rendering of tiles on cluster nodes graphics hardware, sending back rendering results and reassembling image tiles before encoding. Furthermore, to provide a fully comprehensive measure of the overall latency $t_{client-server}$, which is strictly coupled with the previous variables, the designed performance analysis system is also responsible for recording the amount of time needed for a *3D Mobile Viewer* generated command to reach the server side ($t_{command}$). Other critical parameters are recorded, including the frame rate at the rendering site (fps_{server}), the frame rate at the remote client (fps_{client}), and the average bitrate ($bitrate_{client}$). A detailed description of the meaning of all the variables used in this section is reported in Table 1.

5.1 Surface Rendering

The experiments have been carried out by using Chromium and distributing the rendering subsystem on a cluster of up to eight nodes, each running RedHat Linux Fedora 3 operating system. Each node contains a Pentium IV 2 GHz, an nVidia Quadro FX-1100 AGP graphics accelerator with 128 MB of video memory, 256 MB of main memory, and a GB Ethernet network card. The goal is to demonstrate that, for a given t_{MPEG} (that is, also for a given visualization window), interactive visualization at the remote host can be achieved by simply removing the bottleneck on the frame rate experienced at the rendering server, assuming that the available bandwidth is capable of supporting video streaming requirements.

Measurements of t_{render}, t_{MPEG}, and fps_{server} for a particular rendering resolution and streaming video size

TABLE 1
Quick Reference to the Variables Mentioned in This Section

Variable	*Description*
t_{render}	Amount of time required to perform the rendering of the current frame. In a distributed rendering scenario, this time includes the overhead due to graphics context tile distribution to cluster nodes and reassembly. In particular, by referring to actual Chromium based rendering environment, t_{render} takes into account the execution time of the Tilesort, Readback, Send and Render SPUs.
t_{MPEG}	Amount of time required to extract image data resulting from the rendering of current frame from the frame buffer and performing data conversion, resampling and MPEG encoding.
$t_{streaming}$	Amount of time required for a newly generated MPEG frame to be transmitted to the mobile device. It takes into account the time required for an MPEG frame to be processed by the *Packetizer* and *MPEG-TS Multiplexer* modules in the *Streaming Server* (where a buffering delay of approximately 150 ms is also introduced in order to allow for a stable encoding phase), and it considers the transmission time over a specific communication channel. Finally, it includes the time required by the *Mobile 3D Viewer* application for graphics context update. Here, a buffering delay due to a decoding buffer of 200 ms (obtained by shifting DTS and PTS with respect to PCR) is introduced.
$t_{command}$	Amount of time needed for a *Mobile 3D Viewer* generated command to reach the server side.
$t_{client\text{-}server}$	Overall latency in scene manipulation, that is, the time needed for a command issued on the mobile device to reach the server, and for the first newly generated frame to be displayed by *Mobile 3D Viewer*. During remote manipulation, it corresponds to the sum of $t_{command}$, t_{render}, t_{MPEG} and $t_{streaming}$, while when the scene is manipulated on the server side, $t_{command}$ does not have to be taken into account.
$size_{server}$	Resolution of the rendering window. It also corresponds to the size of the visualization area on the display of the console at the graphics server back-end.
$size_{client}$	Resolution of the visualization window on the mobile device. It corresponds to the size of the MPEG video stream generated by the *Encode Server* module.
fps_{server}	Frame rate experienced at the hardware accelerated high-end graphics server (and console).
fps_{client}	Frame rate experienced on the mobile client running *Mobile 3D Viewer*. It usually corresponds to fps_{server}. A limitation on the number of frames per second to be encoded by the *Encode Server* module for limited complexity scenes constrains fps_{client} to thirty fps even for larger values of fps_{server}. The threshold of thirty fps corresponds, approximately, to the maximum frame rate displayable by the selected player. Also, this allows to avoid transmission channel overload and to keep the latency low.
$bitrate_{MPEG}$	Parameter that controls the video encoder establishing the requested bitrate of the MPEG stream.
$bitrate_{client}$	Actual bitrate of the MPEG video streamed to the mobile device over the wireless link.

of 240×240 pixels, as the complexity of the geometry as well as the number of rendering nodes is increased, are reported in Table 2. Values of fps_{client} are not tabulated since they correspond to the frame rate of the server side when fps_{server} is lower than the threshold, and to approximately thirty frames per second when fps_{server} is larger than the threshold. Let's consider the single rendering server-based Chromium configuration. First, it should be noted that the values of t_{render} are slightly lower than in a system where Chromium is not used. This is mainly due to the fact that the use of the distributed rendering middleware provided by Chromium introduces a worsening in the performance because of the overhead related to tile extraction, transmission and reassembly.

Nevertheless, it can be observed that, for low complexity scenes, t_{MPEG} is predominant on t_{render}, and since the encoding task is performed at more than thirty frames per second, interactive frame rates can still be achieved both on the server side and on the mobile device. However, for higher complexity scenes t_{render} becomes predominant. When the complexity of the scene goes beyond one million polygons, rendering and visualization frame rates go below the 30 fps threshold, thus reducing the degree of interactivity (see for example the last two rows in Table 2). It can be easily seen, however, that by increasing the number of rendering nodes, the value of t_{render} can be reduced, thus observing interactive rendering rates even with significantly complex geometries. In particular, using only four servers, all the selected scenes can be rendered at more than thirty frames per second on the server side, thus enabling visualization on the mobile client with a similar frame rate.

The same experiments for a rendering resolution and streaming video size of 512×512 pixels have been repeated. In this configuration, without introducing a rendering distribution layer, even scenes with less than one million polygons cannot be visualized at a satisfactory frame rate. On the contrary, when Chromium is used and a suitable number of rendering nodes is added, ever more complex geometries can be handled. This is clearly illustrated in Table 3.

Using eight rendering nodes, t_{render} is always lower than t_{MPEG} and interactive remote visualization of the most complex scene under investigation (made up by more than three million polygons) can be experienced. It is worth noting that, in this configuration, it appears that it is sufficient to keep the value of t_{render} lower than t_{MPEG} to guarantee the maximum frame rate at the remote site. This assumption is not true in general, since it based on the fact that the video sizes selected for the devices under consideration allow for a continuous encoding at an average rate of more than 30 fps. If larger visualization resolutions

TABLE 2
Performance with a (Surface) Rendering Resolution and Streaming Video Size of 240 × 240 Pixels for Increasing Number of Polygons

Number of polygons (triangles)	1 server			2 servers			4 servers			8 servers		
	t_{render} (s)	t_{MPEG} (s)	fps_{server}	t_{render} (s)	t_{MPEG} (s)	fps_{server}	t_{render} (s)	t_{MPEG} (s)	fps_{server}	t_{render} (s)	t_{MPEG} (s)	fps_{server}
21108	0.00058	0.00715	**139.864**	0.00030	0.00723	**138.317**	0.00015	0.00719	**139.086**	0.00008	0.00727	**137.556**
37380	0.00094	0.00697	**143.454**	0.00049	0.00705	**141.887**	0.00024	0.00701	**142.666**	0.00012	0.00698	**143.361**
57667	0.00134	0.00709	**141.104**	0.00071	0.00701	**142.715**	0.00035	0.00705	**141.905**	0.00017	0.00713	**140.312**
80020	0.00189	0.00693	**144.293**	0.00101	0.00701	**142.647**	0.00049	0.00710	**140.837**	0.00025	0.00721	**138.611**
224574	0.00506	0.00702	**142.375**	0.00267	0.00710	**140.772**	0.00133	0.00706	**141.569**	0.00066	0.00710	**140.837**
277935	0.00634	0.00716	**139.650**	0.00335	0.00708	**141.228**	0.00166	0.00712	**140.434**	0.00083	0.00704	**142.030**
357759	0.00862	0.00721	**115.977**	0.00452	0.00714	**140.067**	0.00224	0.00718	**139.335**	0.00112	0.00694	**144.030**
654666	0.01606	0.00709	**62.252**	0.00868	0.00717	**115.167**	0.00422	0.00694	**144.030**	0.00211	0.00687	**145.645**
995137	0.02485	0.00699	**40.246**	0.01322	0.00707	**75.662**	0.00652	0.00703	**142.264**	0.00324	0.00701	**142.628**
1765388	0.04509	0.00704	22.179	0.02381	0.00696	**41.993**	0.01183	0.00700	**84.503**	0.00588	0.00698	**143.291**
3168455	0.08318	0.00708	12.023	0.04310	0.00717	23.203	0.02200	0.00712	**45.445**	0.01092	0.00714	**91.612**

Frame rate at the client side (fps_{client}) has not been tabulated since it generally assumes a value close to the minimum between the frame rate at the rendering site (fps_{server}) and the selected threshold of 30 frames per second, which allows for local-like interaction. Gray boxes indicate the maximum between t_{render} and t_{MPEG}, that is, the delay actually timing the concurrency of the rendering and encoding phases. Frame rates over the threshold are reported in bold. Nonbold frame rates indicate configurations which provide a frame rate lower than 30 fps.

TABLE 3
Performance with a (Surface) Rendering Resolution and Streaming Video Size of 512 × 512 Pixels

Number of polygons (triangles)	1 server			2 servers			4 servers			8 servers		
	t_{render} (s)	t_{MPEG} (s)	fps_{server}	t_{render} (s)	t_{MPEG} (s)	fps_{server}	t_{render} (s)	t_{MPEG} (s)	fps_{server}	t_{render} (s)	t_{MPEG} (s)	fps_{server}
21108	0.00138	0.02925	**34.189**	0.00072	0.02933	**34.096**	0.00036	0.02929	**34.142**	0.00018	0.02937	**34.049**
37380	0.00219	0.02958	**33.804**	0.00114	0.02966	**33.716**	0.00057	0.02962	**33.760**	0.00029	0.02966	**33.716**
57667	0.00314	0.03080	**32.463**	0.00164	0.03072	**32.548**	0.00082	0.03076	**32.506**	0.00041	0.03007	**33.256**
80020	0.00445	0.03173	**31.521**	0.00232	0.02985	**33.502**	0.00116	0.03079	**32.481**	0.00059	0.03095	**32.310**
224574	0.01193	0.03052	**32.763**	0.00621	0.03060	**32.678**	0.00310	0.03056	**32.720**	0.00157	0.03058	**32.699**
277935	0.01497	0.03092	**32.343**	0.00779	0.03084	**32.427**	0.00389	0.03088	**32.385**	0.00197	0.03080	**32.469**
357759	0.02049	0.02975	**33.614**	0.01067	0.02967	**33.699**	0.00532	0.02975	**33.614**	0.00270	0.02958	**33.804**
654666	0.03819	0.03099	26.186	0.01989	0.03107	**32.186**	0.00992	0.03015	**33.171**	0.00502	0.02933	**34.100**
995137	0.05902	0.03117	16.944	0.03074	0.03063	**32.533**	0.01533	0.03090	**32.362**	0.00777	0.03099	**32.272**
1765388	0.10629	0.03015	9.408	0.05536	0.03007	18.063	0.02761	0.02937	**34.051**	0.01399	0.02964	**33.741**
3168455	0.19807	0.03086	5.049	0.10316	0.03095	9.694	0.05145	0.03090	19.438	0.02606	0.03093	**32.334**

are requested, the delay of the encoding phase becomes a true bottleneck. Even if this issue is beyond the scope of this paper, we expect the introduction of a parallelization scheme in the encoding process to be advisable in the future.

To conclude the evaluation of system behavior in a surface rendering environment, we carried out several experiments in order to evaluate the effect of the main configuration parameters of the *RenderVideo SPU* and *Encode Server* module as well as of the communication channel being used on the performance experienced at the mobile side. In these experiments, a four rendering server based Chromium configuration was used for rendering a high complexity 3D scene (a million polygons per frame) in a five minute long collaborative session. Several configurations of *Render Video SPU* and *Encode Server* parameters including rendering size, visualization size and encoding bitrate have been evaluated. Table 4 summarizes results obtained for each configuration. In addition, detailed diagrams for one of these configurations are presented in Fig. 3.

It should be immediately pointed out that for a remote user, system performance can be evaluated mainly by considering the latency experienced in scene manipulation, the visualization frame rate and the quality of the video stream. Let's analyze first the contribution of each parameter on the overall latency, that is on $t_{client-server}$. From the definition reported in Table 1, it can be observed that the value of $t_{client-server}$ is first influenced by $t_{command}$, that is, the amount of time needed for a *Mobile 3D Viewer* generated command to reach the server side. This time is almost constant for a given communication channel, and it roughly corresponds to its latency since only few bytes have to be transmitted. It has been experienced that in the worst case its value varies between 0.21205 s on a GPRS connection, 0.20786 s over a UMTS channel and 0.00170 s in a WLAN environment. Then, in the computation of the overall latency, the sum of t_{render} and t_{MPEG} has to be considered. In

TABLE 4
Performance Analysis for Different RenderVideo SPU and Encode Server Related Parameters in a Four Server-Based Cluster Configuration Rendering a High Complexity 3D Scene (a Million Polygons per Frame)

$size_{server}$	$size_{client}$	$bitrate_{MPEG}$ (Kbit/s)	Communication Channel	$t_{command}$ (s)	t_{render} (s)	t_{MPEG} (s)	$t_{streaming}$ (s)	$t_{client\text{-}server}$ (s)	fps_{server}	fps_{client}
240×240	240×240	50	GPRS	0.21205	0.00643	0.00705	0.60432	0.82985	141.843	30.103
512×512	240×240	50	GPRS	0.21943	0.01545	0.02683	0.62644	0.88815	37.271	30.081
240×240	240×240	100	UMTS	0.20627	0.00656	0.00701	0.58503	0.80487	142.653	30.062
512×512	240×240	100	UMTS	0.19985	0.01601	0.02691	0.57759	0.82036	37.160	30.067
240×240	240×240	250	802.11b	0.00159	0.00659	0.00705	0.35327	0.36850	141.843	30.103
512×512	240×240	250	802.11b	0.00160	0.01517	0.02687	0.35287	0.39651	37.216	30.204
240×240	240×240	500	802.11b	0.00170	0.00652	0.00703	0.35319	0.36844	142.247	30.053
512×512	240×240	500	802.11b	0.00152	0.01538	0.02673	0.35386	0.39749	37.411	30.098
512×512	512×512	100	UMTS	0.20786	0.01534	0.03078	0.58141	0.83539	32.488	30.394
512×512	512×512	250	802.11b	0.00163	0.01498	0.03006	0.35403	0.40070	32.287	30.145
512×512	512×512	500	802.11b	0.00168	0.01529	0.03090	0.35374	0.40161	32.362	30.024
1024×1024	512×512	1000	802.11b	0.00164	0.03620	0.09048	0.37649	0.50481	11.052	10.996
1024×1024	1024×1024	1000	802.11b	0.00158	0.03612	0.12381	0.37563	0.53714	8.076	8.032

The effect of the rendering size ($size_{server}$), of the visualization size ($size_{client}$), of the video stream bitrate ($bitrate_{MPEG}$), and of the communication channel being selected (GPRS, UMTS, 802.11b) on the degree of usability perceived by the remote user measured in terms of overall latency, frame rate, and visualization quality is reported.

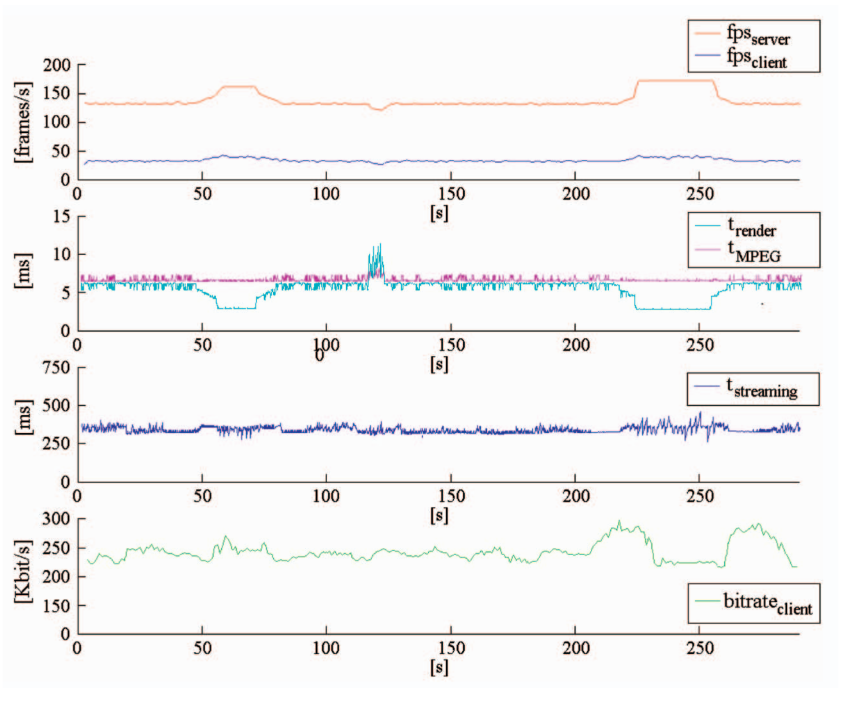

Fig. 3. Measured fps_{server}, fps_{client}, t_{render}, t_{MPEG}, $t_{streaming}$, and $bitrate_{client}$ within a five minute long remote visualization session of a million polygon scene at 240 $\times$ 240 pixels generated using four rendering nodes. The Encode Server module produces a 250 Kbit/s video content at a resolution 240 $\times$ 240 pixels which is streamed over a 802.11b wireless network to a PDA device. During the session, the 3D model is translated out of the viewport area. When this condition occurs, t_{render} and t_{MPEG} decrease and as a consequence fps_{server} increases. However, it should be observed that even if fps_{server} reaches significantly high values, fps_{client} is not allowed to exceed the average threshold of 30 fps.

fact, while in estimating the experienced frame rate only the largest of the two values has to be considered because of the parallelization scheme (see Section 5), in computation of the latency, rendering and encoding of a single frame have to be considered as a sequential process. For this, it can be assumed that the latency strictly depends on the complexity of the scene manipulated as well as on the size of the rendering and visualization windows. On the other hand, it has been experienced that the encoding engine is not significantly sensitive to the value of the selected MPEG bitrate. Furthermore, $t_{streaming}$, the amount of time required for a newly generated MPEG frame to be transmitted to the mobile device, has to be considered. As already stated, this time already takes into account the processing time of *Packetizer* and *MPEG-TS Multiplexer* modules in the *Streaming Server* component, where a buffering delay of approximately 0.150 s is

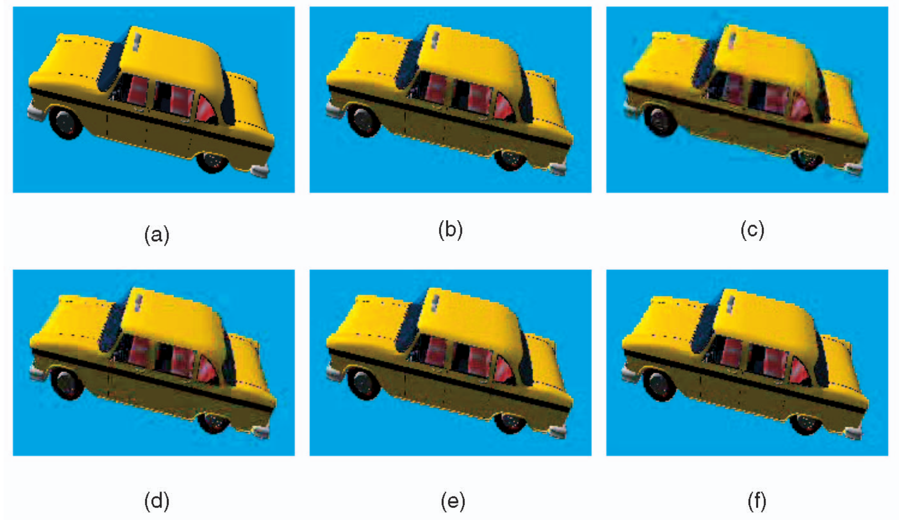

Fig. 4. 235 × 155 snapshots from a 240 × 240 video stream encoded at different bitrates showing in motion and in no-motion models. (a) No MPEG (remote application). (b) 50 Kbps, no motion. (c) 50 Kbps, motion. (d) 100 Kbps, motion. (e) 250 Kbps, motion. (f) 500 Kbps, motion.

introduced, which allows for a stable encoding phase. It also considers the latency and transmission delay over the particular communication channel being adopted. The latency is comparable to $t_{command}$ and as a consequence the same considerations can be applied. On the other hand, the transmission delay depends on the selected bitrate and on the available bandwidth and, in the worst case (corresponding to the largest bitrate), varies between 0.25000 s for a GPRS connection, 0.24500 s for a UMTS channel and 0.00313 s for an IEEE 802.11b link. Finally, $t_{client\text{-}server}$ includes the time required by the *Mobile 3D Viewer* application for graphics context update. In this phase, a buffering delay due to a decoding buffer of 0.200 s obtained by shifting DTS and PTS with respect to PCR is introduced. For all the configurations considered in Table 4, $t_{streaming}$ is always significantly larger than the other contributes. Thus, we can conclude that for a particular communication channel, the reduction of the overall latency requires additional work in order to further limit the buffering delay both on the server and on the client side. However, latency experienced in WLAN environment already allows for a highly interactive scene manipulation with an average response time of 0.4219 s. Considering GPRS and UMTS based communications, a mean delay of 0.85900 s and 0.82021 s has been measured, respectively. Even if, under these conditions, the response time cannot allow for the deployment of applications requiring fast interaction like video games, we believe that this latency can be certainly accepted in all those professional environments where the possibility of navigating complex remote virtual worlds over large geographical distances is of primary concern.

Table 4 also allows the effect of alternative system parameter configurations to be evaluated. In particular, performance for large rendering sizes (useful to enable visualization sessions involving an operator at the graphics server site requesting a higher resolution rendering) as well as for high visualization resolutions (i.e., 1,024 × 1,024 or greater, needed when clients are Tablet PCs or similar devices) are analyzed. With respect to the first case, it should be observed that the encoding time t_{MPEG} is strictly coupled to both $size_{server}$ and $size_{client}$. This is due to the fact that when $size_{client}$ is smaller than $size_{server}$, a resize of the content of the framebuffer has to be performed before beginning the encoding step. This means that under these conditions, a decrease in the performance experienced at the mobile site for a given visualization size has to be expected. Concerning the second case, it can be observed that when t_{MPEG} becomes predominant on t_{render}, only limited interactivity can be achieved at the remote site. In this case, the introduction of additional cluster nodes is of no help. The only feasible solutions could consist in increasing the computational power of the machine hosting the *Encode Server* via a hardware upgrade or via the introduction of a parallelization scheme in the encoding phase.

Finally, it has been observed that the quality of the video stream received at the remote site is largely influenced by the selected bitrate of the MPEG stream, as expected. Results for different encoding bitrates ranging from 50 to 1,000 Kbit/s are therefore reported. Clearly, the choice of a particular bitrate is driven by the available bandwidth of the communication channel being considered. Very low values for $bitrate_{MPEG}$ allows a limited bitrate stream to be produced useful for managing remote visualization sessions over low bandwidth wireless links. It should be pointed out that even if video quality during scene manipulation is largely reduced for very low bitrates (as in the case, for example, of GPRS at 50Kbit/s), quality worsening of still pictures for static scenes is almost negligible thanks to the intrinsic properties of MPEG encoding (see Fig. 4). However, it should be noted that the variations in the communication channel latency experienced during performed tests never affected the user interactivity. Also, the percentage of packet loss was around 0.3 percent in wireless LAN and 0.5 percent in GPRS/UMTS environments (all measurements were performed in a laboratory environment without experiencing handovers, signal losses, and signal strength deteriorations). If latency variations are not negligible or the percentage of packet loss increases, many frames would arrive after the time indicated in their DTS and PTS (and would have to be ignored by the client) or would not arrive at all, thus decreasing the frame rate and quality of the scene visualized at the client side. There is no doubt that additional work will be necessary in the future to arrive at a quantitative evaluation of the effect of the requested bitrate and network latency on the perceived quality and degree of interactivity in the specific field of video-based 3D visualization (especially in nonlaboratory environments). Nevertheless, many works in the literature have already addressed the effect of the bitrate and of the loss and delay of packets on the quality of a video stream in wireless network environments for general purpose applications and proposed possible solutions mainly based on adaptive coding schemes. We therefore believe that the presented framework would significantly benefit from the application of such existing methodologies [34]. In conclusion, all the considered parameters can be effectively used to adjust overall latency, perceived frame rate and visualization quality. It is worth noticing that suitable values have to be selected depending also on the particular communication scenario being considered in order to properly handle trade off between latency and frame rate, and to get the expected level of interactivity at the mobile side.

For the sake of completeness, some brief considerations about system performance during multiuser visualization sessions are presented. In this case, it is important to establish whether all the mobile clients receive a copy of the same video stream or whether multiple streams at different resolutions have to be provided. In the first case, system performance does not change since stream duplication is accomplished by the underlying network. In the second case, system behavior depends on the characteristics of the host running the *Encode Server* modules as well as on the requested resolutions. As an example, it has been experienced that the machine used in this work is not capable of supporting simultaneous encoding of two different streams at 240 × 240 and 512 × 512 pixels, respectively. For this, a distributed solution can be adopted since each *Encode Server* is a logically independent module and two machines each running an instance of the encoder have to be used. In Fig. 5a, a multiuser remote visualization session involving both Tablet PC and PDA-equipped users as well as a console operator is presented.

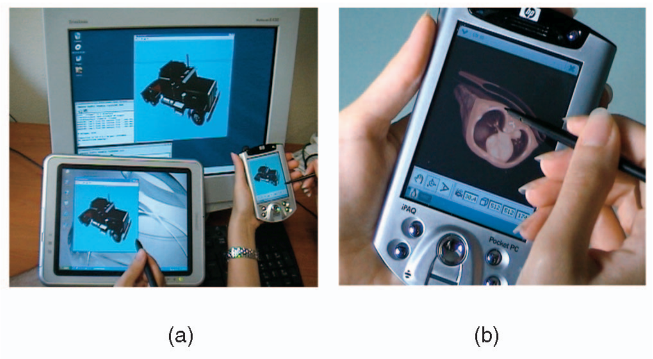

Fig. 5. (a) Different types of mobile devices participating in a collaborative remote visualization session. The console visualization (and rendering) resolution is 1,024 × 1,024. Tablet PC and PDA devices receive a 512 × 512 and a 240 × 240 video sequence, respectively. Video streams are generated by two instances of the Encode Server module running on separate hardware. (b) A 512 × 512 × 174 volume is inspected on a PDA at a resolution of 240 × 240 pixels.

5.2 Volume Rendering

The proposed architecture was then used to prove the feasibility of interactive navigation of high complexity volume datasets which can be found at *www.volvis.org*. For this, various experiments were carried out by enabling remote visualization of the XMLViewer tool.

Results obtained in a Chromium-based distributed rendering scenario for a rendering resolution and streaming video size of 240 × 240 pixels and increasing data set size are reported in Table 5. Let's consider first the single rendering node-based Chromium configuration. It can be observed that t_{render} is always predominant on t_{MPEG} and interactive frame rates can be achieved only for volume sizes up to a million voxels (that is, $256 \times 256 \times 128$). However, for a given fixed volume size, the value of t_{render} can be approximately halved as the number of cluster nodes is progressively doubled. System scales as expected and using eight rendering nodes, fps_{server} is always larger than the maximum frame rate which can be provided at the mobile site. Thus, even the $512 \times 512 \times 512$ data set can be displayed at roughly 30 frames per second on the portable device.

TABLE 5
Performance with a (Volume) Rendering Resolution and Streaming Video Size of 240 × 240 Pixels for Increasing Volume Sizes

Volume size	1 server			2 servers			4 servers			8 servers		
	t_{render} (s)	t_{MPEG} (s)	fps_{server}	t_{render} (s)	t_{MPEG} (s)	fps_{server}	t_{render} (s)	t_{MPEG} (s)	fps_{server}	t_{render} (s)	t_{MPEG} (s)	fps_{server}
128×128×128	0.01611	0.00709	**62.063**	0.00885	0.00701	**112.955**	0.00458	0.00702	**142.375**	0.00235	0.00696	**143.668**
256×256×128	0.02489	0.00709	**40.180**	0.01367	0.00688	**73.145**	0.00702	0.00711	**140.623**	0.00362	0.00705	**141.905**
256×256×256	0.03899	0.00705	25.645	0.02142	0.00705	**46.687**	0.01102	0.00710	**90.785**	0.00568	0.00709	**141.003**
512×512×174	0.05721	0.00717	17.480	0.03126	0.00705	**31.989**	0.01616	0.00696	**61.880**	0.00835	0.00709	**119.739**
512×512×373	0.18601	0.00701	5.376	0.10334	0.00701	9.677	0.05263	0.00690	19.002	0.02711	0.00710	**36.880**
512×512×512	0.23160	0.00723	4.318	0.12654	0.00708	7.903	0.06636	0.00714	15.069	0.03328	0.00717	**30.052**

Frame rate at the client side (fps_{client}) has not been tabulated since it generally assumes a value close to the minimum between the frame rate at the rendering site (fps_{server}) and the selected threshold of 30 fps.

TABLE 6
Performance with a (Volume) Rendering Resolution and Streaming Video Size of 512 × 512 Pixels

Volume size	1 server			2 servers			4 servers			8 servers		
	t_{render} (s)	t_{MPEG} (s)	fps_{server}	t_{render} (s)	t_{MPEG} (s)	fps_{server}	t_{render} (s)	t_{MPEG} (s)	fps_{server}	t_{render} (s)	t_{MPEG} (s)	fps_{server}
128×128×128	0.05780	0.02925	17.302	0.03174	0.03093	**31.507**	0.01631	0.03084	**32.425**	0.00841	0.02942	**32.541**
256×256×128	0.08756	0.02931	11.421	0.04815	0.03117	20.771	0.02467	0.03172	**31.526**	0.01280	0.02952	**33.750**
256×256×256	0.11815	0.02968	8.464	0.06499	0.02976	15.387	0.03319	0.03104	**30.132**	0.01717	0.02954	**33.875**
512×512×174	0.14513	0.02977	6.891	0.07976	0.03032	12.538	0.04077	0.03085	24.530	0.02122	0.02968	**33.990**
512×512×373	0.47775	0.03079	2.093	0.26351	0.03053	3.795	0.13458	0.03028	7.431	0.06944	0.02955	14.401
512×512×512	0.61396	0.03091	1.629	0.33671	0.02895	2.970	0.17426	0.03095	5.739	0.08963	0.02907	11.157

Let's now consider a rendering resolution and streaming video size of 512 × 512 pixels. Without using a distributed rendering environment, none of the considered datasets can be visualized at a satisfactory frame rate. Since the average encoding time value for a single frame using the selected hardware is around 30 ms, which would correspond to a frame rate of approximately 33 frames per second (which is already close to the above threshold), values of fps_{server} over the threshold can be obtained only when t_{MPEG} is dominant or at least very close with respect to t_{render}. When Chromium framework is used, values of t_{render} can be constrained as needed by simply adding a suitable number of cluster nodes (see Table 6). In particular, with an eight rendering server configuration, high interactivity can be guaranteed for volumes up to 4.5 million voxels (corresponding to a $512 \times 512 \times 174$ data set). Nevertheless, it should be observed that using the largest number of nodes available in our cluster, the proposed framework already enables the manipulation of a $512 \times 512 \times 512$ data set on a limited resources portable device, at a resolution of 512 × 512 pixels at more than 10 frames per second. Results for different video bitrates and communication channels are not reported since the same considerations for surface-based rendering apply. Only minor variations have been experienced concerning the rendering time. Fig. 5b shows a volume rendering visualization session involving a PDA as mobile interacting device.

6 Conclusions and Future Work

Interactive visualization of complex 3D scenes on portable devices is still considered a challenging task. It is predictable that interactive rendering in mobile environments will evolve over time and ever more sophisticated 3D graphics hardware acceleration support will be introduced in next generation mobile appliances, allowing users to navigate ever more complex and realistic 3D environments. Nevertheless, remote visualization represents a viable alternative for large data set interactive 3D visualization in mobile environments as of today. The ambitious application framework presented in this paper represents a crucial step in an ongoing effort to build a comprehensive client-server 3D rendering framework enabling simultaneous mobile users to interact with graphics intensive OpenGL-based applications without the user noticing that most of the processing is actually done on a remote and possibly distributed server. Chromium allows complex 3D geometries to be rendered at high frame rates; by increasing the number of rendering nodes, medium level GPUs can be used to manage scenes modeled by millions of textured polygons and voxels, therefore obtaining MPEG streams displayable at 30 fps at the client side. The current platform allows realistic and complex 3D data sets to be displayed in an interactive way on limited resource devices such as PDAs and Tablet PCs allowing OpenGL-based programs to be ported into a mobile scenario. The proposed implementation uses MPEG and MPEG TS video technologies to distribute a video stream, embedding 3D frames generated by surface and volume rendering applications on a high-end rendering server and enables collaborative visualization at interactive frame rates over both local and geographic wireless networks on mobile devices. Future work will be devoted to the evaluation of alternative video codecs, rendering schedulers and synchronization protocols in order to achieve higher compression ratios and lower latencies. This will translate into the possibility of providing interactive performances even on very low bandwidth channels, setting practically no limits to the complexity of 3D scenes to be visualized in mobile environments.

Acknowledgments

The authors wish to thank Joachim Diepstraten and Simon Stegmaier from the VIS group of the University of Stuttgart who helped in discussing technical details and improving the manuscript. In addition, special thanks to Professor Thomas Ertl, head of the VIS group, for his collaboration.

References

[1] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. Kirchner, and J.T. Klosowski, "Chromium: A Stream-Processing Framework for Interactive Rendering on Clusters," *Proc. SIGGRAPH,* pp. 693-702, 2002.
[2] M. Woo, J. Neider, T. Davis, and D. Shreiner, *OpenGL Programming Guide: The Official Guide to Learning OpenGL—Version 1.2,* third ed. Addison-Wesley, 1999.
[3] *PocketGL,* http://www.pocketgear.com, 2001.
[4] *OpenGL ES,* http://www.khronos.org/opengles/, 2002.
[5] *Vincent,* http://ogl-es.sourceforge.net/, 2003.
[6] *Hybrid's Gerbera,* http://www.hybrid.fi/, 2004.
[7] *Klimt,* http://studierstube.org/klimt/, 2003.
[8] *M3D,* http://www.jcp.org/en/jsr/, 2005.
[9] *M3G (JSR 184),* http://www.hybrid.fi/main/m3g/, 2004.

[10] L. McMillan, "Image-Based Rendering: A New Interface between Computer Vision and Computer Graphics," *SIGGRAPH Computer Graphics Newsletter—Applications of Computer Vision to Computer Graphics,* vol. 33, no. 4, 1999.
[11] C. Zunino, F. Lamberti, and A. Sanna, "A 3D Multiresolution Rendering Engine for PDA Devices," *Proc. World Multiconf. Systemics, Cybernetics, and Informatics (SCI '03),* vol. 5, pp. 538-542, 2003.
[12] C.F. Chang and S.H. Ger, "Enhancing 3D Graphics on Mobile Devices by Image-Based Rendering," *Proc. IEEE Third Pacific-Rim Conf. Multimedia,* pp. 1105-1111, 2002.
[13] I.M. Martin, "Adaptive Rendering of 3D Models over Networks Using Multiple Modalities," technical report, IBM T.J. Watson Research Center, 2000.
[14] *Silicon Graphics, Inc. OpenGL Vizserver,* http://www.sgi.com/software/vizserver/, 1999.
[15] *IBM, Deep Computing Visualization,* http://www-03.ibm.com/servers/deepcomputing/, 2004.
[16] S. Stegmaier, M. Magallón, and T. Ertl, "A Generic Solution for Hardware-Accelerated Remote Visualization," *Proc. Eurographics—IEEE TCVG Symp. Visualization,* pp. 87-94, 2002.
[17] A. Nye, *X Protocol Reference Manual, XWindow System Series.* O'Reilly & Assoc., 1995.
[18] T. Richardson, Q. Stafford-Fraser, K.R. Wood, and A. Hopper, "Virtual Network Computing," *IEEE Internet Computing,* vol. 2, no. 1, pp. 33-38, 1998.
[19] *VirtuaGL Project,* http://virtualgl.sourceforge.net/, 2004.
[20] S. Stegmaier, J. Diepstraten, M. Weiler, and T. Ertl, "Widening the Remote Visualization Bottleneck," *Proc. Third Int'l Symp. Image and Signal Processing and Analysis,* pp. 174-179, 2003.
[21] K. Engel, O. Sommer, and T. Ertl, "A Framework for Interactive Hardware Accelerated Remote 3D-Visualization," *Proc. Eurographics—IEEE TCVG Symp. Visualization,* pp. 167-177, 2000.
[22] D. Beerman, "CRUT: Event Distribution and Image Delivery for Cluster-Based Remote Visualization," *Proc. IEEE Visualization 2003—Workshop Parallel Visualization and Graphics,* 2003.
[23] I.J. Grimstead, N.J. Avis, and D.W. Walker, "Visualization Across the Pond: How a Wireless PDA Can Collaborate with Million-Polygon Datasets via 9,000km of Cable," *Proc. ACM/SIGGRAPH Web3D Symp.,* 2005.
[24] J. Diepstraten, M. Görke, and T. Ertl, "Remote Line Rendering for Mobile Devices," *Proc. IEEE Computer Graphics Int'l (CGI '04),* pp. 454-461, 2004.
[25] F. Lamberti, C. Zunino, A. Sanna, A. Fiume, and M. Maniezzo, "An Accelerated Remote Graphics Architecture for PDAs," *Proc. ACM/SIGGRAPH Web3D Symp.,* pp. 55-61, 2003.
[26] F. Lamberti and A. Sanna, "A Solution for Displaying Medical Data Models on Mobile Devices," *WSEAS Trans. Information Science & Applications,* vol. 2, pp. 258-264, 2005.
[27] *Mental Images RealityServer,* http://www.mentalimages.com/2_3_realityserver/, 2005.
[28] "Mental Images White Paper," *RealityServer Functional Overview,* RS-T-005-01, 2005, http://www.mentalimages.com/cgi-bin/rs-whitepaper.cgi, 2005.
[29] *Coding of Moving Pictures and Associated Audio for Digital Storage Media at Up to About 1.5 Mbit/s,* MPEG1—ISO/IEC 11172, 1992.
[30] *Libavcodec Library,* http://ffmpeg.sourceforge.net/, 2005.
[31] *Generic Coding of Moving Pictures and Associated Audio Information,* MPEG2-ISO/IEC 13818, 1994.
[32] *Videolan Project,* http://www.videolan.org, 2002.
[33] *SGI Volumizer,* 2001, http://www.sgi.com/products/software/volumizer/.
[34] H. Liu and M. El Zarki, "Adaptive Source Rate Control for Real-Time Wireless Video Transmission," *Mobile Networks and Applications,* vol. 3, pp. 49-60, 1998.

Fabrizio Lamberti received the degree in computer engineering and the PhD degree in software engineering from the Politecnico di Torino, Italy, in 2000 and 2005, respectively. He has published a number of technical papers in international journal and conferences in the areas of mobile and distributed computing, wireless networking, image processing, and visualization. He has served as a reviewer and program or organization committee member for several conferences. He is member of the Editorial Advisory Board of international journals. He is member of the IEEE and the IEEE Computer Society.

Andrea Sanna graduated in electronic engineering in 1993, and received the PhD degree in computer engineering in 1997, both from Politecnico di Torino, Italy. Currently, he has an assistant professor position at the Second Engineering Faculty. He has authored and coauthored several papers in the areas of computer graphics, virtual reality, parallel and distributed computing, scientific visualization, and computational geometry. He is currently involved in several national and international projects concerning grid, peer-to-peer, and distributed technologies. He is a member of ACM and serves as reviewer for a number of international conferences and journals.