A solution for displaying medical data models on mobile devices

FABRIZIO LAMBERTI and ANDREA SANNA
Dipartimento di Automatica e Informatica
Politecnico di Torino
Corso Duca degli Abruzzi 24, I-10129
Torino, Italy

Abstract: displaying three dimensional graphics on last generation mobile devices such as Personal Digital Assistants (PDAs) and smart phones is a new and challenging task. Moreover, several additional issues are involved when complex models have to be displayed such as in 3D medical imaging. Medical data models often result by CT (Computed Tomography) or MRI (Magnetic Resonance Image) and the obtained volumetric data sets are processed and displayed by volume rendering techniques. These rendering algorithms require a lot computational power and storage resources not available in nowadays mobile devices. The paper proposes a distributed architecture where a remote rendering server is able to manage complex models and to code a MPEG stream to be sent to the client; the user can visualize and analyze interactively the model by roto-translation commands.

Key-Words: remote visualization, mobile devices, medical imaging, distributed architecture

1 Introduction

3D graphics on mobile devices is a new and challenging task for researchers and developers. Moreover, the capability to be able to display complex 3D structures is worthwhile in several disciplines, for instance in medicine, biology, seismology, CFD, and so on. In particular, in 3D medical imaging the visualization of large 3D data-sets is used for diagnostic purposes and for planning of treatment or surgery.

Two approaches are basically adopted to address this issue. The first approach uses local device resources to display the 3D scene while the second approach 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 images (from the server to the client) and commands (from the client to the server) transmissions.

Some solutions were proposed to locally render 3D graphics on mobile devices. For instance, the PocketGL [1] is a 3D toolkit for PocketPC and consists of source code containing numerous functions which generate and 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 [2] 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. On the other hand, M3D (Mobile 3D Graphics API) [3] 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 visualizations, and so on.

Even although nowadays technologies allow to develop extremely attractive rendering applications, the visualization of complex 3D models is still out of the capabilities of mobile devices. The idea of dividing computational and visualization tasks allows to overcome limits due to local hardware and software resources. For instance, Silicon Graphics, Inc. has developed a commercial solutions named Vizserver [4] 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. Stegmaier et al. [5] use a VNC version for PocketPC in order to remotely control a X-server, while Engel et al. [6] proposed a framework for interactive remote visualization written in Java in order to provide a platform independent tool. Recently, Lamberti et al. [7] 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 [8,9]; the contribution of each GPU is reassembled and sent to the PDA client as a flow of still images.

The paper is organized as follows: Section 2 presents the proposed architecture, Sections 3 and 4 provide a detailed description of modules server-side and client-side, respectively. Finally, Section 5 shows a set of experimental results and a deep evaluation of the performance.

2 The Proposed Framework

Taking into account the results achieved by research activities carried on within the last years in the fields of mobile local and remote rendering as well as the limits of such solutions we started the development of the proposed architecture, an open framework which enables highly interactive remote visualization of OpenGL based 3D graphics applications on mobile devices over wireless communication channels. The core of the high-end server-side visualization subsystem relies on a cluster-based distributed rendering engine and allows to display extremely realistic and complex 3D models and data-sets in an interactive way on limited resource devices like PDAs, Tablet PCs and smart phones within both WLAN and WWAN environments. As side-effect, the framework we developed can be used as a core building block in the deployment of more complex collaborative visualization environments thus enabling in principle the remote management of any application requesting high-performance visualization hardware support. In areas where specialists are separated by distance the work-flow efficiency can be significantly improved by collaborative applications. For example, such applications allow users to discuss the visualized data sharing the same view. Furthermore, expensive experts can be consulted and distance education or advanced training can be held (these characteristics are extremely important in 3D medical imaging where a set of specialists can be enabled to share visualization of data obtained by CTs or MRIs). Additionally, our approach provides simultaneous access to a server application for multiple users. Thus, capabilities of expensive hardware can be simultaneously shared by low-cost client systems. The proposed framework for remote visualization of complex 3D environments on mobile devices consists in a three-tiers architecture including a server-side component (in the following referred as *Interaction Server*) controlling a distributed cluster-based hardware accelerated rendering environment and a client application (namely, the *Mobile 3D Viewer*) enabling user interaction and providing realistic visualization functionalities. The server-side software can be transparently integrated within existing applications thus enabling multiple mobile users to remotely control a variety of graphics application based on the OpenGL libraries.

3 Serverside Remote Rendering Subsystem

3.1 The interaction server

In a remote visualization scenario, rendering is actually performed using hardware resources of a high-end graphics server that hosts the 3D graphics application and is responsible for streaming visualization frames generated by the rendering process. We based the development of our server-side remote rendering subsystem on a software component, the Interaction Server, 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 Interaction Server, an OpenGL application can be transparently controlled by simultaneous remote users in an interactive and collaborative way.

Basically, the internal operation logic driving the Interaction Server can be described as follows (see Fig. 1). A newly generated scene manipulation event on the client mobile device triggers the submission of a command packet describing the event to the Interaction Server over a wireless communication link. Once this packet reaches the Interaction Server, it is translated into a compliant semantic and then passed on to the proper OpenGL callback functions which in turn will adjust mapping and rendering parameters. Then, the OpenGL directives are locally executed, exploiting the processing capabilities available at the rendering site. Once the rendering has been completed, the frame-buffer is copied in the main memory. The Interaction Server 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.

3.2 Distributed cluster-based rendering environment

In the 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 of recent advances in accelerator technology, many real-time 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 on even 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/PCs have emerged as a viable option to alleviate this bottleneck. The necessary components for scalable graphics on clusters of PCs have matured sufficiently to allow exploration of clusters as a reasonable alternative to multiprocessor servers for high-end visualization. In addition to graphics accelerators and processor power, memory and I/O controllers have reached a level of sophistication that permits high-speed memory, network, disk, and graphics I/O to all occur simultaneously, and high-speed general purpose networks are now fast enough to handle the demanding task of routing streams of graphics primitives. The cluster-based rendering sub-system that has been deployed is based on the Chromium architecture [8,9].

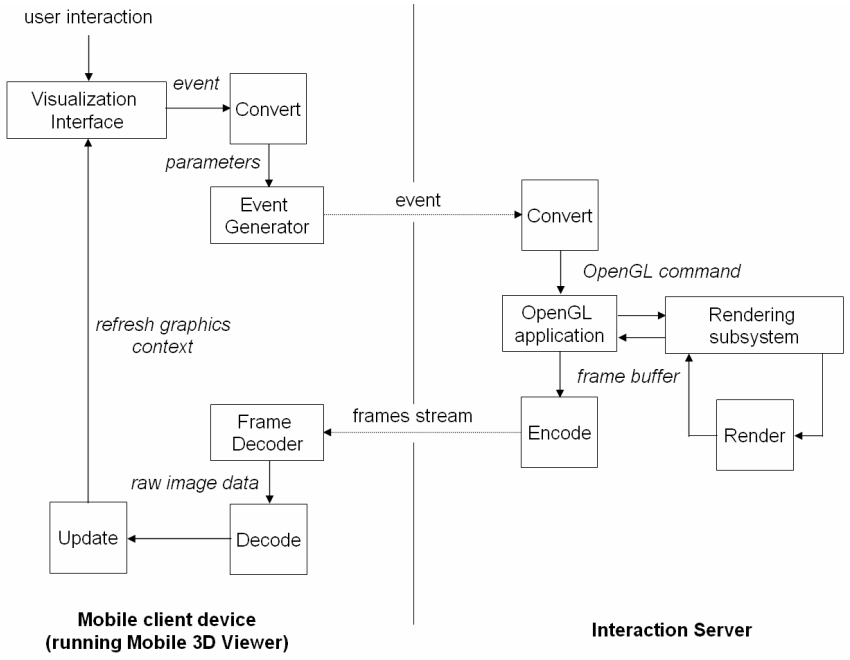

Fig. 1 Sequence of operations in a remote rendering session.

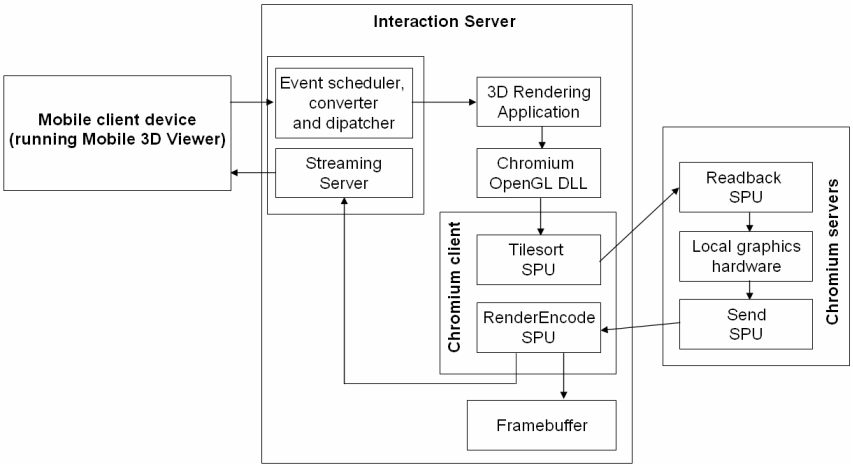

Fig. 2 Layout of the proposed three-tiers architecture.

The Chromium framework takes advantage of the aforementioned opportunities by providing 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 it is responsible for rendering a part of the output image which is later reassembled for visualization. Existing OpenGL applications can use a cluster with a 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 (i.e. medical data), 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 manage to distribute rendering workload to high-end processing servers. 3D 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. In Fig. 2 the overall architecture of the proposed remote visualization framework is presented.

3.3 The RenderEncode module

A specific Chromium module named RenderEncode has been designed in order to manage framebuffer data compression which allows the generation of 2D contents suitable for effective real-time distribution in a remote visualization environment. The newly developed module is responsible for reassembling tiles received from Chromium server nodes as well as for writing reassembled frames into the framebuffer. Additionally, the RenderEncode incorporates MPEG video processing capabilities, thus providing the requested support for efficient compression of framebuffer content. In particular, by overriding the OpenGL swapBuffers() function, the RenderEncode is capable of extracting image data resulting from the rendering of current frame from the framebuffer, scaling grabbed frame to a specific resolution, converting resized frame into YUV 4:2:0 format suitable for MPEG processing, performing MPEG encoding of current frame generating an MPEG ES (Elementary Stream) [10] and, finally, passing encoded MPEG bitstream to a streaming application (the Streaming Server). The streaming application has been developed as a separate application which receives contents to be streamed over a TCP connection. This allows for the streaming application to be run on a different machine thus limiting the overhead on the Chromium client. In the encoding process, all typical MPEG parameters can be adjusted in order to precisely control bitrate and quality of the generated stream.

3.4 The Streaming Server component

Streaming of MPEG encoded frames generated by the RenderEncode has been delegated to a specific component in the overall architecture, named the Streaming Server, which is responsible both for performing several pre-processing tasks on the encoded bitstream and for streaming the resulting MPEG video sequence to remote clients. By keeping framebuffer encoding and video sequence streaming separated, it has been possible to design a Chromium module 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 [11] and provides effective support for real-time 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, 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 decoders and data can be streamed directly on a UDP socket without introducing the overhead of additional protocols. Nevertheless, in the future we expect to evaluate the effects of the introduction of alternative synchronization protocols (like, for example, RTP). 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. 3), namely the Packetiz*er* module, the MPEG-TS Multiplexer module and the Streaming module. The first two modules deals with pre-processing 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.

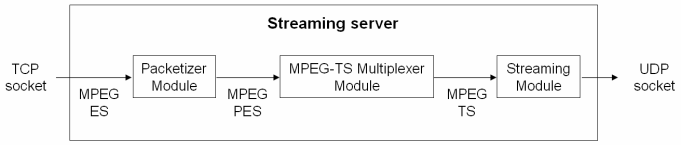

Fig. 3 Structure of the Streaming Server component.

4 Client-side Visualization Application

In the proposed remote rendering scenario, mobile devices running a dedicated application named Mobile 3D Viewer only act as visualization front-end supporting user interaction with a virtual 3D scene which is actually produced at distance. We evaluated the remote control of distributed 3D graphics applications on traditional, off-the-shelf portable computers. In particular, we selected a HP iPaq H5550 PDA and a Compaq TC1000 Tablet PC.

The user can interact with the actual device by tapping a pen over the display. On the PDA, a directional pad is available, which implements the traditional keyboard arrow-keys functionalities. Moreover, several application buttons are available on both the devices, which can be completely customized via software. Tablet PC device has been selected for its higher screen resolution with respect to common handheld devices. In this way, usability and performance on the next generation high resolution PDAs can be estimated. Concerning the practical implementation of the Mobile 3D Viever visualization application, we investigated the feasibility of the exploitation of existing off-the-shelf software modules. However, such general-purpose technologies may degrade the overall performances of the system. For this reason, we decided to develop an ad-hoc application to provide special support for accelerated remote visualization on portable devices. At the same time, we performed an evaluation of the programming environment best suitable for the deployment of the client side software. In other works, the Java programming language was chosen to develop different frameworks for interactive remote visualization thanks to its high portability across different platforms. However, in this case we preferred to adopt the native language of the platform under consideration in order to optimize the 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. 2): 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 (i.e. pen tapping, navigation pad, application buttons depending on the particular device being considered) and passes them to the Event Generator module. A series of control buttons below the rendering area allows the user to interactively manage a subset of the functionalities offered by the remote OpenGL application. In the future, complete support for application-specific commands will be provided. Furthermore, information related to the current scene including polygons count, volume data size and frame rate are displayed to the user see Fig. 5 and 6). 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. 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. Mobile 3D Viewer MPEG video processing capabilities rely on a general purpose open-source multimedia player developed within the Videolan project [12] which is capable of displaying MPEG TS video streams at thirty frames per seconds. Other video-coding schemes (MPEG-2, MPEG-4) and streaming protocol (RTP) can be used to replace current MPEG TS based solution.

4.3 The Event Generator

Finally, the Event Generator software component is responsible for receiving information concerning the events generated at the user interface and converting them in suitable commands for the remote application. Such commands are then encoded in a format suitable for transmission through a TCP connection over a low bandwidth channel.

5 Experimental Results

In this Section we present two different usage scenarios which would benefit of the proposed architecture: a parallel volume renderer based on a commercial application programming interface (API) used to interactively explore 3D texture based volumetric data-sets [13] and a general purpose surface renderer for generic 3D scene navigation developed at Politecnico di Torino university.

We evaluated the effectiveness of the proposed framework in remote 3D rendering scenarios enabling mobile users endowed with PDA and Tablet PC devices to simultaneously interact with the aforementioned 3D rendering applications running on a remote hardware accelerated high-end graphics server. Our 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 3D visualization sessions among distributed mobile users regardless of the complexity of the 3D data-set being considered. In other words, mobile users can perceive a degree of interactivity comparable to that they would experience in a typical local visualization based environment.

Since visualization frame rate and the overall latency experienced at the remote client constitute the main limitations of existing remote rendering architectures, especially when considering collaborative visualization session, an event-driven analysis system has been designed in order to accurately quantify critical parameters of the remote visualization system thus providing an effective measure of the interactivity of the proposed architecture. Therefore, for each rendered frame, a distributed measurement environment which involves almost all the components of the designed architecture allows to record the amount of time needed for:

- rendering current frame (t_{render});
- producing a MPEG encoded frame (t_{MPEG});
- displaying a MPEG encoded frame to the mobile device ($t_{streaming}$).

To provide a fully comprehensive measure of the overall latency ($t_{server\text{-}client}$), 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 ($bitrate_{client}$).

Various experiments have been conducted by varying the size of the geometric 3D model and of the 3D texture volume data-set. Table 1 reports measured t_{render}, t_{MPEG}, fps_{server}, and fps_{client} in a centralized geometric rendering based session (that is, without exploiting Chromium acceleration) by varying the complexity of the scene. It can be observed that the average t_{render} strictly depends on the polygon count of the particular rendered scene. On the contrary, mean t_{MPEG} is not significantly affected by the particular scene being considered. Moreover, it has to be noticed that, except for very low rendering times, one can observe the same frate rate both at the rendering server (fps_{server}) and at the remote visualization site (fps_{client}).

Nevertheless, a limitation on the number of frames per second to be encoded by the RenderEncode has to be imposed for limited complexity scenes in order to avoid transmission channel overload and to keep the latency low for lower rendering times. Selected threshold enables a very high degree of interactivity in collaborative visualization sessions.

	t_{render} (ms)	t_{MPEG} (ms)	fps_{server}	fps_{client}
12500	**16**	**16**	**31.3**	**30.0**
25000	**25**	**17**	**23.8**	**23.8**
50000	**40**	**18**	**17.2**	**17.2**
100000	**64**	**17**	**12.3**	**12.3**
200000	**91**	**17**	**9.2**	**9.2**

Table 1 Rendering time (t_{render}), encoding time (t_{MPEG}) and frame rate of the rendering process on the high-end graphics server (fps_{server}) and of the video stream presented to the mobile user (fps_{client}) for different geometries with increasing polygon count.

From Table 1 it can be noticed that, as expected, the rendering time per frame which is strictly coupled with the complexity of the scene being rendered heavily contributes in degrading interactivity of the system since frame rate at the client side is constrained by the limited graphics performances of the rendering host: medium and high complexity scenes exhibit poor rendering performances (fps_{server}) which translate in unsatisfactory frame rates at the remote client (fps_{client}). We therefore repeated the same experiments on a cluster of eight nodes each running RedHat Linux 8.0. The nodes contain a Pentium IV 2 GHz, a nVidia GeForce2 MX 440 graphics accelerator with 64 MB of video memory, 256 MB of main memory and a Gigabit Ethernet network card. Our goal is to demonstrate that interactive visualization at the remote host can be achieved by simply removing the bottleneck on the frame rate experienced at the rendering server (supposed available bandwidth supports video streaming). The graph presented in Fig. 4 shows the average frame rate for a given fixed rendering resolution as we increase the complexity of the geometry, that is the number of polygons per frame, from 12500 to 100000. Four curves are shown, corresponding to a cluster of 1, 2, 4, and 8 nodes. Performances in the single server-based Chromium configuration are slightly lower than those presented in Table 1. That is mainly due to the fact that the use of the distributed rendering middleware provided by Chromium introduces a worsening of the performances due to the overhead related to tiles extraction, transmission and reassembly. Nevertheless, by increasing the number of rendering nodes, one can observe interactive rendering rates even with significantly complex geometries. In particular, using 8 servers, the 100000 polygons scenes can be rendered at about fifty frames per second on the server side thus enabling a 30 frames per second visualization at the mobile client. In this way, a local-like interaction with highly complex 3D scenes can be experienced by remote mobile users.

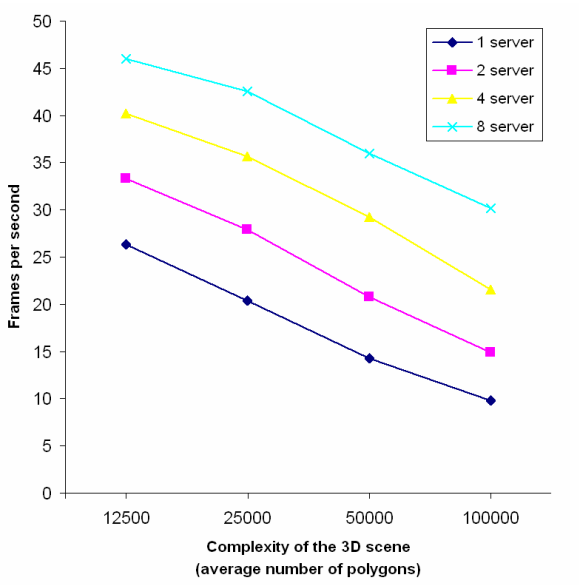

Fig. 4 Each curve shows the relationship between scene complexity and performance for a given number of rendering servers.

Fig. 5 and 6 show two visualization session examples by using a PDA as remote client. A medical model is inspected by the user in Fig. 5, where it is shown both the visualization on the handheld device and the rendering on the console of the remote cluster. Control buttons below the rendering area are visible in Fig. 6 as well as information concerning the frame rate and the number of polygons.

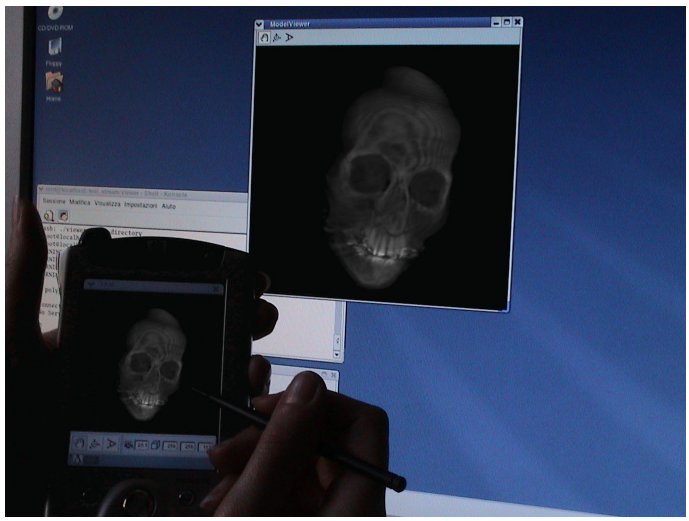

Fig. 5 A medical model is displayed both on the mobile client and on the console of the remote rendering cluster; a GPRS wireless connection is used in this case.

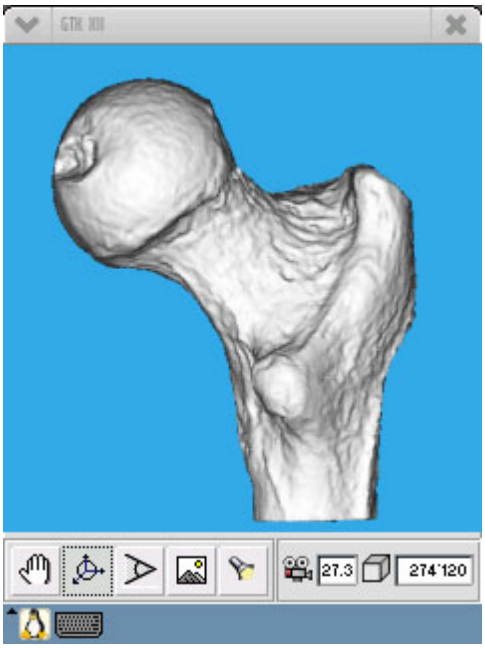

Fig. 6 A set of icons help the user during visualization sessions. It can be noticed the frame rate value (about 27.3 fps) and the number of polygons of the analysed object [14] (over 200.000 in this case).

6 Conclusion and Future Work

Real-time visualization of extremely complex 3D scenes on portable devices is still considered a challenging task.

It is predictable that interactive rendering in mobile environments will evolve over the time and ever more sophisticated 3D graphics hardware acceleration supports 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 high-quality interactive 3D visualization in mobile environments since today. The ambitious application framework we have presented in this paper represents a crucial step in our 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.

The current platform allows to display extremely realistic and complex 3D data-sets in an interactive way on limited resource devices such as PDAs and TabletPCs and requires only minimal efforts to port any existing OpenGL 3D rendering program 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 it enables collaborative real-time 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 and synchronization protocols in order to achieve higher compression ratios and lower latencies. This will translate in the possibility of providing real-time performances even on very low bandwidth transmission channels setting practically no limits to the complexity of 3Ds scene to be visualized in mobile environments.

7 Acknowledgements

This project is supported by the Ministero dell'Istruzione dell'Università e della Ricerca in the frame of the PRIN 2004 project (Prot. 2004095094): “Studio e sviluppo di un sistema per il controllo e il monitoraggio in tempo reale del territorio per la prevenzione degli incendi”.

References:

1. PocktGL website: *http://www.pocketgear.com*
2. OpenGL ES website: *http://www.khronos.org/opengles/*
3. M3D website: *http://www.jcp.org/en/jsr*
4. Silicon Graphics, Inc OpenGL Vizserver website: *http://www.sgi.com/software/vizserver/*
5. Stegmaier, S., M. Magallón and T. Ertl, T. *A Generic Solution for Hardware-Accelerated Remote Visualization*, Joint Eurographics - IEEE TCVG Symposium on Visualization, 2002, pp. 87-94.
6. Engel, K., O. Sommer and T. Ertl. *A Framework for Interactive Hardware Accelerated Remote 3D-Visualization*, Joint Eurographics - IEEE TCVG Symposium on Visualization, 2000, pp. 167–177.
7. Lamberti, F., C. Zunino, A. Sanna, A. Fiume and M. Maniezzo. *An Accelerated Remote Graphics Architecture for PDAs*, Proc. of ACM/SIGGRAPH Web3D 2003 Symp., 2003, pp. 55-61.
8. Humphreys, G., M. Houston, R. Ng, R. Frank, S. Ahern, P. Kirchner and J.T. Klosowki. *Chromium: a stream-processing framework for interactive rendering on clusters*, In Proceedings Siggraph, 2002, pp.693–702.
9. Chromium: *http://sourceforge.net/projects/chromium/*
10. MPEG-1 - ISO/IEC 11172 - *Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s*.
11. MPEG-2 - ISO/IEC 13818 - *Generic coding of moving pictures and associated audio information*.
12. Videolan project website: *http://www.videolan.org*
13. Silicon Graphics, Inc OpenGL Volumizer website: *http://www.sgi.com/software/vizserver/ Volumizer*
14. Cyberware website: *http://www.cyberware.com*