Pers Ubiquit Comput (2013) 17:1487–1502
DOI 10.1007/s00779-012-0598-y

ORIGINAL ARTICLE

A scalable architecture for 3D map navigation on mobile devices

José M. Noguera · Rafael J. Segura · Carlos J. Ogáyar · Robert Joan-Arinyo

Received: 23 February 2012 / Accepted: 31 July 2012 / Published online: 2 September 2012

Abstract Mobile devices such as smart phones or tablets are rapidly increasing their graphics and networking capabilities. However, real-time visualization of 3D maps is still a challenging task to accomplish on such limited devices. In this paper, we describe the principles involved in the design and development of a scalable client–server architecture for delivering 3D maps over wireless networks to mobile devices. We have developed a hybrid adaptive streaming and rendering method that distributes the 3D map rendering task between the mobile clients and a remote server. This architecture provides support for efficient delivery of 3D contents to mobile clients according to their capabilities. As a proof of concept, we have implemented a prototype and carried out exhaustive experiments considering different scenarios and hundreds of concurrent connected clients. The analysis of the server workload and the mobile clients performance show that our architecture achieves a great scalability and performance even when using low-end hardware.

J. M. Noguera (✉) · R. J. Segura · C. J. Ogáyar
Escuela Politécnica Superior, University of Jaén,
Campus Las Lagunillas, Edificio A3, 23071 Jaén, Spain
e-mail: jnoguera@ujaen.es

R. J. Segura
e-mail: rsegura@ujaen.es

C. J. Ogáyar
e-mail: cogayar@ujaen.es

R. Joan-Arinyo
Escola Tècnica Superior d'Enginyeria Industrial Barcelona,
Universitat Politècnica de Catalunya, Diagonal 647,
808028 Barcelona, Spain
e-mail: robert@lsi.upc.edu

Keywords Mobile computing · 3D graphics · Terrain rendering · Mobile map

1 Introduction

Interactive visualization of maps on mobile devices plays an important role in a number of graphics applications including mobile guides, personal navigation, and access to location-based services. According to [25], textual interfaces on mobile guides are being abandoned, and today, 2D maps are the most common approach to providing data to users.

But despite their usefulness, 2D mobile maps pose some drawbacks. Since they provide an abstract, two-dimensional representation of a 3D environment, they require cognitive resources and topological reasoning in order to read the mobile map and to relate it to the environment that surrounds the user [38]. In contrast, 3D maps combined with actual imagery (aerial/satellite) provide a directly recognizable visualization of the surrounding environment that is easier and faster to understand [39]. These 3D representations permit real-time fly-throughs and immersive first-person views of a realistic virtual representation of the geographical area where the user is physically located, as illustrated in Fig. 1.

For all these reasons, the development of new techniques that bring together location-aware ubiquitous devices and visualization of interactive 3D maps is interesting. The availability of such techniques would allow new interesting and exciting ways to deliver 3D contents in user-centric pervasive environments.

However, there exist severe technical and technological limitations that have precluded the widespread adoption of 3D maps on ubiquitous devices. Computational resources

Fig. 1 An example of the proposed solution running on an iPhone. *Top* photograph author: Wenceslao Castillo

in these devices are sparse, both main memory and secondary storage are limited, wireless networks are slow, and displays are small. In general, their computational power is an order of magnitude smaller than the hardware commonly used in today's desktop PCs. As a result, 3D visualization of large maps is still very complex to achieve on these devices.

These difficulties have forced the rebirth of research on distributed rendering techniques. Much research has been performed recently in the area of distributed rendering on mobile devices, see Sect. 2. Server-based solutions are based on an indirect rendering, where the 3D geometry is rendered in a dedicated rendering server, and the resulting images are then transmitted to the user. These techniques, although akin to very thin devices, require a powerful server and easily lead to network congestion. On the contrary, client-based techniques charge the entire rendering task to the mobile client, and the server acts as a simple file server. Therefore, a powerful client is required in order to handle large and detailed scenes at interactive rates.

Clearly, it would be interesting to design a hybridization scheme that takes advantage of both client and server-based rendering approaches, in order to overcome their individual drawbacks. In [36], we introduced a hybrid terrain rendering approach that provides tools for enhancing both the quality and the interactivity when rendering large 3D maps on mobile devices using low-bandwidth wireless networks. The clients are in charge of rendering the terrain close to the viewer, whereas the terrain in the background is portrayed as panoramic 2D images, rendered on demand by a remote server.

The main contribution of this paper is a detailed description of a novel client–server architecture that expands the hybrid approach formalized in [36] in order to support multi-client environments. Whereas in [36] each client required a dedicated server, the architecture proposed in this paper is capable of providing service to hundreds of concurrent and heterogeneous mobile clients using commodity hardware on both sides, clients and server. Figure 2 outlines the proposed architecture. This paper also provides an extensive evaluation of the hybrid rendering approach when it is deployed in an actual multi-client environment.

The architecture proposed in this paper can serve as a solid basis for the development of useful mobile 3D map-based applications, such as location-based services, 3D tourist guides [35], mobile games, and collaborative virtual environments.

The rest of the paper is organized as follows: Sect. 2 provides a necessary background on 3D graphics on mobile devices. Section 3 summarizes the principles of the hybrid rendering approach used in this paper. Section 4 presents a general overview of the proposed architecture, while Sects. 5, 6, and 7 describe its three main components. Section 8 presents and discusses experimental results. Section 9 provides a user study that completes our evaluation. Finally, Sect. 10 summarizes results of our research and gives a vision of future work.

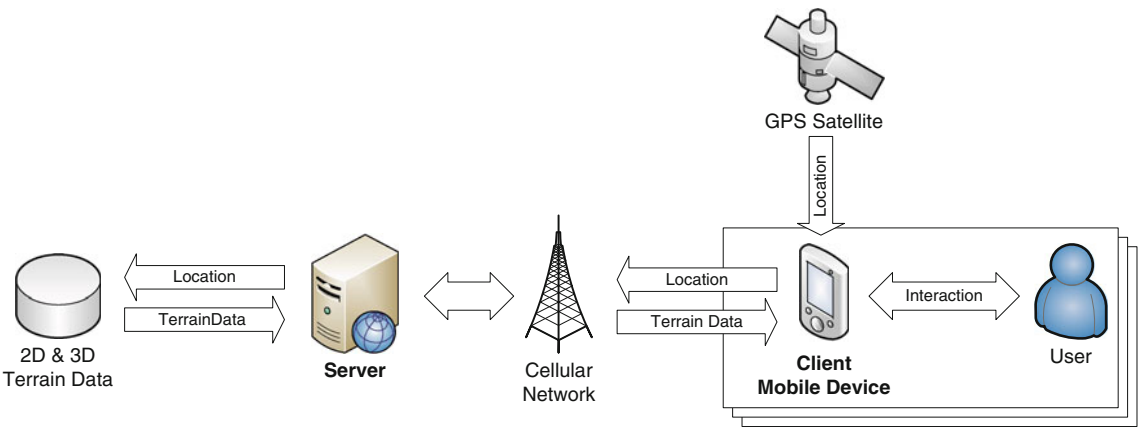

Fig. 2 General framework of the proposed architecture

2 Background

This section reviews the state of the art in the field of rendering generic tridimensional scenes on mobile devices. Special attention is paid to networked solutions. Interactive navigation through complex 3D worlds requires the ability to render the scene at an acceptable number of frames per second while keeping image quality as high as possible. Through the years, different techniques have been proposed to achieve this goal.

Local rendering methods assume that the 3D scene completely fits the device's memory, and thus, there is no need to hold any connection with remote servers. These kinds of methods are usually employed in mobile video games. Games have been the main driving force for the huge performance boost experienced by mobile *graphics processing units* (GPUs) in recent years [1, 10], and some advanced real-time rendering engines have recently arisen, for example, [14, 51].

In the scientific literature, however, most proposed local approaches for mobile devices seek to find more efficient strategies for rendering scenes than direct visualization, for example, by means of point-based [13, 17] or illustrative rendering techniques [20]. However, because of the growing inclusion of GPUs in today's mobile devices, most of these techniques are becoming unnecessary, as simple direct visualization techniques are preferred. Also, the ubiquity of mobile devices has encouraged many researchers and commercial companies to develop applications of three dimensional navigation across indoor [49], urban [3, 8, 46], and open environments [32, 53]. All these applications require the scene to be preinstalled in local memory.

Local rendering techniques are simpler to implement, and the user's experience is not reduced by network congestion or signal fades. But due to the small size of the device memory, the size and complexity of the scene become limited. Therefore, local rendering methods confine users to small virtual environments, which is a serious drawback, especially on navigation applications.

However, and because mobile devices are usually connected to a network, the use of rendering techniques in which large 3D scenes are stored in a remote server becomes a viable solution to overcome this storage limitation. In general, methods for rendering 3D scenes in client–server environments can be classified into three major categories, according to where the rendering takes place [33]: (a) *server-side rendering methods*, (b) *client-side rendering methods*, and (c) *hybrid rendering methods*. The different client–server visualization strategies proposed in the literature will be reviewed and discussed below.

2.1 Server-side rendering

In this category, a dedicated remote rendering server is in charge of performing the geometry rendering task and streaming the resulting video stream of images to a client over a network. The client is only responsible for displaying such prerendered images. Therefore, these methods become quite appropriate for rendering geometrically complex models in thin devices with low computing or storage capacity. However, as mobile devices are becoming less computing-bounded, we expect that these approaches will progressively be discarded in favor of client-side and hybrid methods. These methods also present the following drawbacks:

1. *Interactivity* In order to achieve an interactive rate, these techniques stream a massive volume of images to the client, which can easily result in a congested network and a loss of real-time interaction with the model.
2. *Scalability* A powerful server with graphics capabilities is required. An increment in the number of concurrent clients can easily increase the response times. Also, the 3D rendering capabilities of modern mobile devices are wasted.
3. *Image quality* This usually suffers because of the lossy compression algorithms used to reduce the traffic.

The issue concerning sending images through a network and the problems related to it have been studied by many researchers, and numerous proposals can be found in the literature:

Some authors [5, 9, 54] proposed image-based remote rendering techniques. The server provides the client with a series of images (key frames and their associated z-buffer) and the client is in charge of calculating the intermediate frames using a warping technique. The problem with these techniques is to determine the situation of the camera in order to avoid artifacts, such as holes.

Aranha et al. [2], Jeong and Kaufman [24], and ImageVIS3D [23] proposed remote visualization systems in which the server generates a sequence of images by using ray-tracing; these images are compressed and sent to the client's device for its visualization. In 2007, Jeong claimed to achieve a speed of five frames per second using an IEEE 802.11b local area network on a PDA. Similarly, Hildebrandt et al. [19] employed a rendering server to project massive 3D cities onto extended cube maps, which are transmitted and visualized by thin mobile clients.

Instead of sending static images, Lamberti et al. [27, 42] and Wen et al. [52] presented remote visualization systems in which MPEG video streams are sent through the network. In [27] (2007), the authors claimed to achieve the remote visualization on a PDA with 30 frames per second

using an eight-computer cluster running a software called *Chromium* [21].

Boukerche et al. [6] and Pazzi et al. [43] have presented alternative methods for image-based approaches by using scheduling mechanisms and partial streaming of images. However, these approaches severely limit the viewer's movement and do not perform well in dynamic scenes.

2.2 Client-side rendering

In this category, 3D rendering tasks are delegated to the client and the server only provides geometric data to the client, which is responsible for rendering it locally. These methods do not involve any rendering on the part of the server, and consequently, they do not require a server with graphics capabilities. They also reduce the streaming load. However, clients must provide the computational power required to render good quality images. These methods are well suited to applications for which real-time interaction is paramount to viewing the model, assuming that the mobile client has the ability to store and render the corresponding data.

Lluch et al. [30] presented a client–server system for the visualization of multiresolution 3D models on mobile clients. The main problem is the considerable latency experienced when the user interacts with the model and it needs to be redrawn. In [37], Nurminen describes a complete client–server solution for virtual browsing in urban environments through mobile devices. *NaviGenie* [34], on the other hand, is a commercial application that provides procedurally generated cities for urban 3D navigation.

Terrain rendering is an application that usually benefits from a client-side rendering paradigm. Digital 3D terrain representations (usually in the order of gigabytes or terabytes) easily exceed the storage capacity of any desktop computer. This fact has caused several out-of-core rendering techniques for PCs to be developed, see [41] for a detailed study. Yet, the interactive visualization of large 3D terrains on mobile devices is still an unexplored field in the scientific literature.

Pouderoux and Marvie [44] presented in 2005 one of the first attempts at rendering large 3D maps on mobile devices. Their proposal consisted of a very simple paging approach based on a grid of tiles that managed to render a scene of 3,744 triangles at 7 frames per second using an USB 2.0 network. More recently, Suárez et al. [50] presented *Glob3*, an open-source framework for rendering virtual globes on mobile devices and WebGL compliant browsers. However, Google Earth [16] is still the best-known commercial application that provides 3D maps on mobile devices. Google's approach to achieving interactive frame rates consists of using low geometry terrain models and focusing on providing high-quality textures.

2.3 Hybrid rendering

Hybrid methods aim at distributing the calculation between the server and the client in order to improve the performance.

Some authors [12, 18, 45] have proposed client–server hybrid techniques that perform an expressive visualization of the scene. The server carries out image processing techniques on the 3D models in order to extract simple primitives in run-time, such as lines or silhouettes. The use of these primitives, instead of actual geometry or textures, allows for a reduction of the bandwidth needed for its transmission to the client, while increasing the visualization speed. We should bear in mind that these techniques show monochromatic and/or nonphotorealistic images, which makes scene comprehension difficult for the user.

There exist other kinds of hybrid methods whose goal is to partition the scene into parts that are rendered on a server and parts that are downloaded and rendered on the client. Such methods have the advantage that they reduce the geometric complexity of the scene rendered by the mobile client by replacing parts of it with images. However, determining whether a part of a scene should be rendered on the server or on the client is not a trivial task [33].

Noguera et al. [36] presented a client–server hybrid rendering technique following this scheme. This work focussed on navigating large terrains using mobile devices. This approach distributes the 3D rendering workload between a client and a server and manages to achieve an interactive frame rate using a single Nokia N95 smart phone connected to a server via cellular networks.

3 The hybrid terrain rendering approach

For the sake of completeness, we briefly recall our hybrid rendering approach [36] here. This approach distributes the 3D rendering workload between a mobile client, usually with very limited resources, and a remote server, generally featuring high-end hardware and software resources. The server stores the complete dataset and is responsible for providing the client with small chunks of 3D terrain close to the user's position and also for rendering and sending the client impostors for the terrain in the background. The client is in charge of rendering the map close to the user's position, displaying the impostor that replaces the distant terrain and requesting from the server updates of the data when the user moves.

In Computer Graphics, the term impostor refers to a 2D image that is used instead of actual 3D geometry. Since images are faster to render, this technique aims at improving the rendering performance on the mobile device.

3.1 Terrain representation

Since available CPU and memory resources in mobile devices are limited, adaptively streaming and rendering large-scale terrains on mobile devices requires the use of specifically adapted algorithms and data structures.

In *Geographic Information Systems* (GISs), 3D terrains are commonly represented by *Digital Elevation Models* (DEMs) [11, 31]. Although there exist several types of them, in this paper, we are mostly interested in rasterized DEMs, also known as height maps. A height map is a two-dimensional grid of regularly spaced sample points, each one representing an elevation value. Realism is further enhanced by adding photo textures, consisting of actual aerial/satellite imagery.

We organize the height map according to two different levels. The first level subdivides the complete terrain height map into a regular grid of equal size tiles, each tile covering a squared area of the height map. The second level consists of a set of restricted quadtrees [40, 47], each quadtree associated with one terrain tile, see Fig. 3. Textures associated with the terrain are also structured according to a grid of quadtrees defined as before. Figure 5a shows the part of the scene that is rendered locally by the mobile device.

This structure is suitable for progressive data transmission [28, 40] over wireless links. It is also optimized for fast rendering on mobile GPUs using indexed triangle strips, [36].

3.2 Panoramas

In our approach, view-dependent impostors are used to portray the terrain located far from the viewer, rendered by the server on demand and streamed to the client. These impostors consist of two-dimensional synthetic images that simulate a wide view of a physical terrain placed in the background far from the viewer. These impostors are called panoramic impostors, or simply *panoramas* [7].

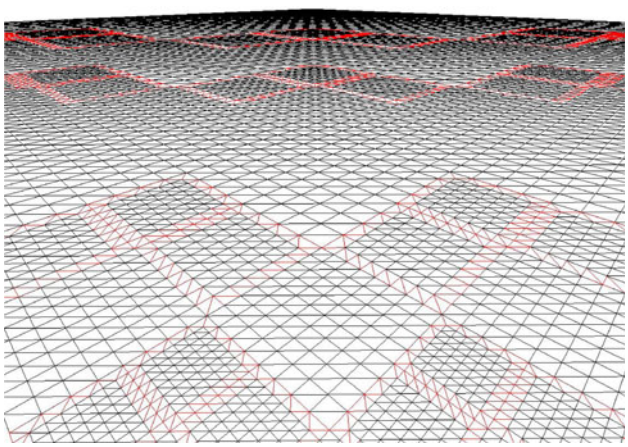

Fig. 3 Restricted quadtree triangulated mesh, used by the mobile client to render locally the nearby 3D terrain

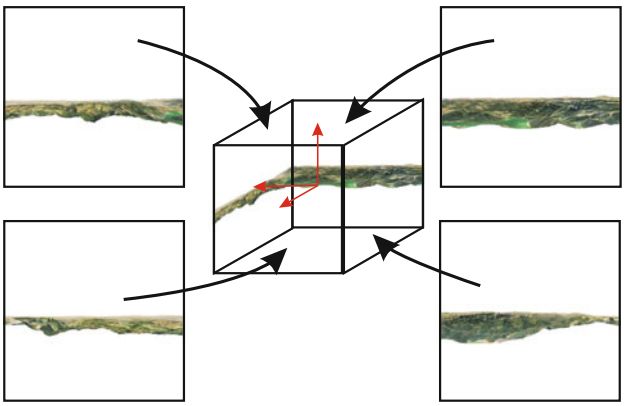

Fig. 4 A panorama, generated by the server on demand

In order to visualize a panorama, it is first projected on the inner six faces of a cube centered at the viewer, see Fig. 4. Panoramic images projected onto a cube are usually referred to as a *skybox* [48] or *environment map* [4]. The resulting image is composed by the client by merging the terrain and the panorama as illustrated in Fig. 5b.

Figure 5b, c compare the same scene rendered, respectively, by our hybrid method and a pure client-side method (*Google Earth* for iOS [16]). Clearly, to achieve a good enough viewing distance without incurring loss of interactivity, the latter method must use a coarsened representation of the terrain at middle and large distances, resulting in a flat horizon that lacks distinctive details.

The panorama, rendered on demand by the remote server, and the close-range geometry, rendered locally by the client, should be correctly matched in order to avoid visible discontinuities and artifacts. Therefore, we split the terrain into nearby terrain and panorama as follows: Let the view volume [15] in the client be limited by the front and back clipping planes placed, respectively, at $zfront_c$ and $zback_c$ distance from the viewing point. Similarly, let the view volume in the server be limited by the clipping planes placed at distances $zfront_s$ and $zback_s$. Then, we require that $zfront_s = zback_c$, that is, the front and back culling planes in the server and client respectively, are coincident. See Fig. 6. Clearly, the client renders the close terrain whereas the distant terrain is culled. On the contrary, the server only uses the distant terrain to render the panorama.

As long as the viewer does not move, the panorama remains valid. Under a perspective projection, a small movement of the viewer causes a small displacement of the projection of distant parts of the scene. Given this large temporal coincidence, it is wasteful to update the panorama for every small movement of the client. Nonetheless, if the viewer moves and the panorama is not properly updated,

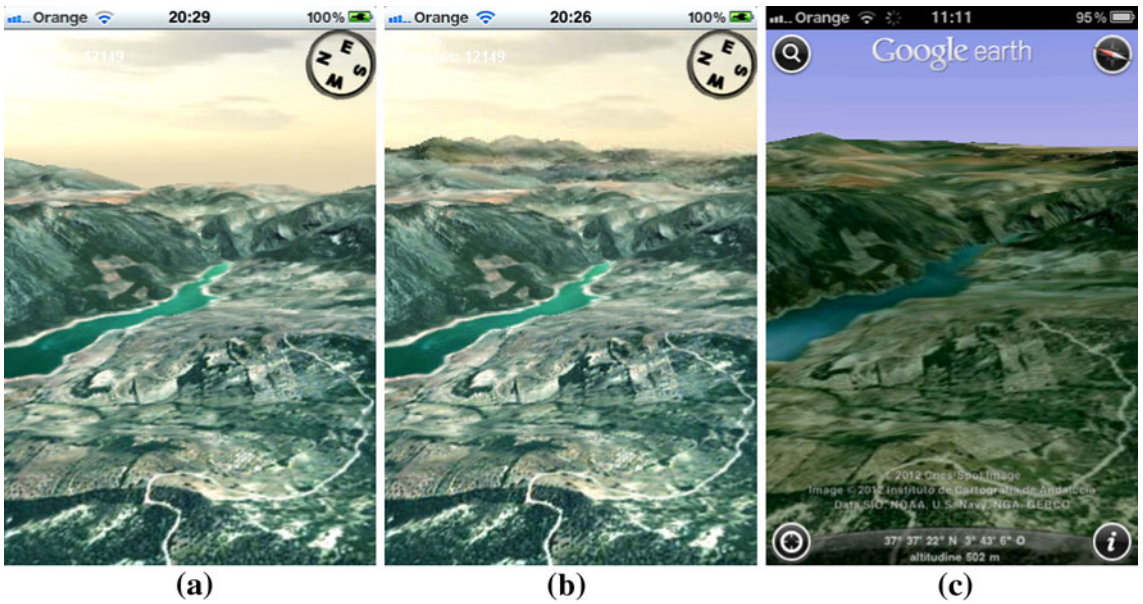

Fig. 5 **a** Nearby terrain rendered by the client at 60 frames per second (iPhone 3GS). **b** Synthesis of the previous image and the panorama. **c** The same scene on Google Earth 6.1.0 (Jan. 2012) running on the same device

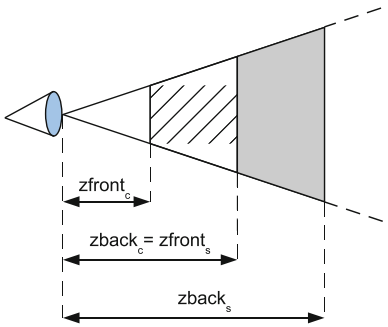

Fig. 6 Splitting the view volume as terrain to be rendered by the client and panorama. The hatched area is the view volume rendered by the client. The *gray area* is the view volume rendered by the server

the displayed image is no longer correct. In [36], a criteria for assessing the error committed when the viewpoint varies but the panorama is not updated is defined. This approach is based on estimating the error after each user movement and updating the panorama whenever this error exceeds a predefined threshold.

4 The framework

This section introduces a novel software architecture for the hybrid rendering approach able to allow a variable number of clients to be connected simultaneously to the server. The higher the number of clients that can be served, the better the system's scalability.

The architecture developed is illustrated in Fig. 7. This architecture allows location-aware interactive rendering of open 3D virtual environments on mobile devices. Specifically, it consists of three software components:

- *The Main Server* runs in the server and is in charge of handling all the requests of the clients.
- *The Panorama Server* also runs in the server and provides compressed panoramas on request, which are streamed to the client.
- *The Client Application* runs in the mobile devices. It manages the user interface and displays the map.

Since each component has been designed as an independent application, the system works even if no Panorama Server is present. In this case, the system behaves like a standard client-side rendering architecture.

A typical scenario of user interaction might as be as follows. Once a user launches the mobile client application on his mobile device, a network connection to the Main Server is established. This connection remains open until the application is closed. Then, the user's current location is obtained via GPS and provided to the Main Server. In return, an interactive 3D map is progressively streamed to the client based on this geographical position. The user can then roam freely across the 3D environment, exploring any area of his/her interest. New parts of the 3D map are requested and downloaded from the Main Server as needed. Periodically, the mobile application also assesses the error of the current panorama as mentioned in Sect. 3.2 Whenever this error exceeds a predefined threshold value, a new updated panorama is requested and downloaded from the server.

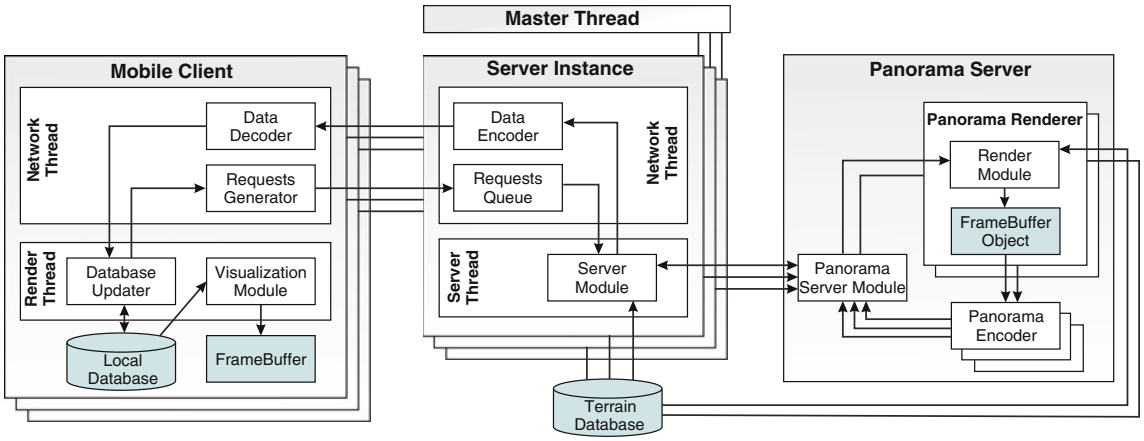

Fig. 7 Architecture for the hybrid, client–server-based rendering system

In the following sections, we describe in detail each component of our system and how they interrelate.

5 Main Server architecture

For the Main Server, we propose a multi-threaded architecture as illustrated in Fig. 7. The data flow is managed by a master thread that listens to a network socket, waiting for incoming clients. When a client connects to the server, a new *Server Instance* is created with an associated network socket and a connection is established with the client. The Server Instance stays alive until the connection is closed or the server application dies. Therefore, multiple clients can be connected to the server at the same time with one dedicated Server Instance per client.

The Server Instance has also been designed following a multithreaded paradigm, in which communication and processing are performed in different threads. See Fig. 7. The first thread deals with network transmission, while the second drives the internal logic of the server.

Due to the fact that cellular networks usually suffer from low bandwidth and high latencies, we use a simple binary request–response protocol built over TCP/IP to efficiently communicate the client devices and the server. A client request can query either a quadtree node or a panorama from the server. The server then issues a response message, which provides the requested data to the client:

- *Quadtree node requests* When the client needs to download terrain data, it sends a quadtree node request to the server. In response, the Server Instance retrieves the height values and the associated texture from a terrain database and sends them back to the client. The sequence diagram in Fig. 8 illustrates these steps.
- *Panorama requests* These requests receive a slightly different treatment. Once the request reaches the Server Instance, it is passed to the Panorama Server, which will in turn return a new panorama according to the current client's geographic position. See sequence diagram in Fig. 9.

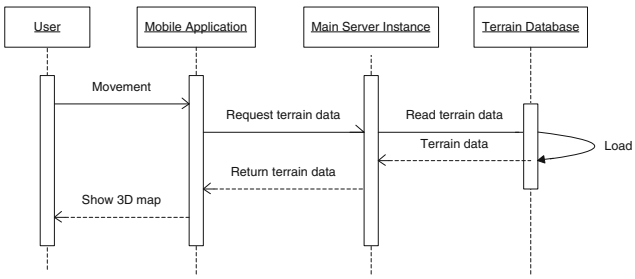

Fig. 8 Sequence diagram of processing a quadtree node request

6 Panorama Server architecture

The Panorama Server is responsible for rendering and encoding all the panoramas that are requested by the multiple Server Instances. Figure 9 shows the sequential diagram corresponding to the panorama generation process. In brief, this process can be described as follows. Incoming panorama requests are handled by a first-in first-out scheduling system. A panorama is then built by the *Panorama Renderer module*, see Fig. 7, by projecting the distant terrain on a frame buffer. Once it is synthesized, the resulting raw images are compressed by the *Panorama Encoder module* using any standard image compression algorithm. The encoded panorama is then delivered to the Server Instance that requested it and finally sent to the mobile client through a wireless link. In following subsections, the two modules that compose the Panorama Server will be described in greater detail.

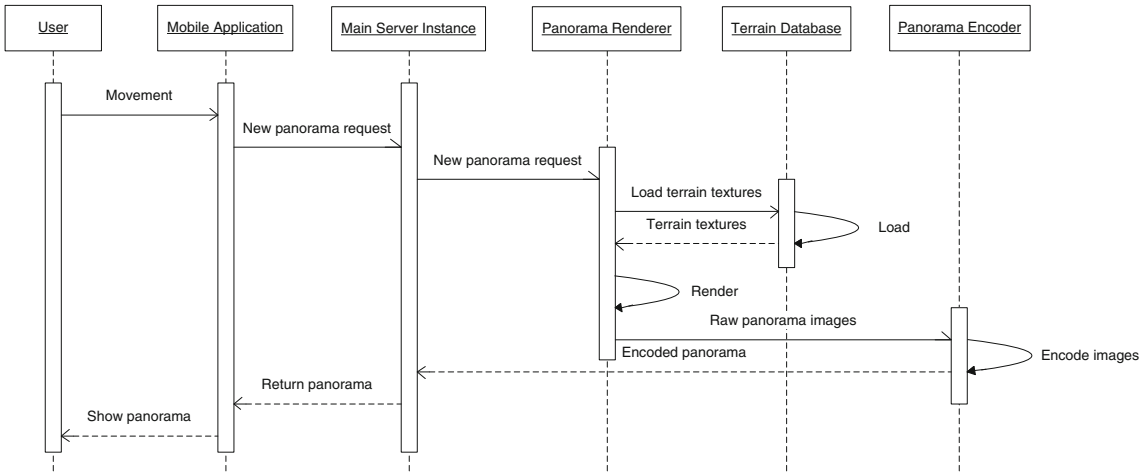

Fig. 9 Sequence diagram of processing a panorama request

6.1 The Panorama Renderer

As stated in Sect. 3, the hybrid rendering approach splits the rendering workload between the mobile devices and the remote server. The Panorama Renderer is responsible for carrying out the rendering workload of the server.

The construction of a cubic panorama is straightforward [6]. Each face of the cube covers 90 degrees of view both horizontally and vertically, see Fig. 4. The panorama is built by the Panorama Renderer by placing the camera referred to the viewer's geographical coordinates in the mobile client and making use of the terrain nearby. Then, six orthogonal images are rendered. Finally, the resulting images allocated in the frame buffer are copied from video memory to main memory and placed in a queue waiting for their turn to be encoded by the Panorama Encoder module.

The ubiquitous and multi-client nature of our solution raises an unprecedented challenge that must be addressed. Most 3D terrain rendering techniques employ frame-to-frame coherence to avoid complex re-meshing and re-transmission of the terrain to the graphics hardware [41]. However, the rendering task performed by the server to generate panoramas does not present this coherence. Given the multi-client nature of our solution, subsequent requests of panoramas are likely to belong to different users, who might be navigating over different geographical areas far from each other. Therefore, standard terrain rendering approaches do not apply here.

To the best of our knowledge, the issue of rendering 3D terrains from different viewpoints in every frame has not been yet addressed in the literature. Thus, we must define a new data structure that should be capable of:

- Computing the terrain triangulation in a fast way, regardless of the viewer position in the last rendered frame.
- Sharing terrain data used for rendering panoramas requested by different clients, avoiding data redundancy.

We will describe our solution below, which successfully overcomes the aforementioned problems. First, the complete height and texture maps are partitioned into a regular grid of squared size tiles. Second, the height and the texture maps of the area covered by each tile are stored, respectively, as a grayscale and a color texture. Next, these textures are uploaded to the GPU's texture memory.

However, since current terrain datasets often exceed the capacity of a typical GPU's memory, we cannot assume that the complete set of tiles will fit in the server's GPU memory. Therefore, we employ a texture memory manager that always maintains in GPU memory the tiles needed for rendering the panorama according to the locations provided by the clients. That is, in order to render a panorama from a given viewpoint, the texture memory manager uploads to the GPU the textures corresponding to the tiles centered at the viewpoint, see Fig. 10a. Cached tiles can be re-used to render panoramas for additional clients without incurring extra costs, see Fig. 10b. When the GPU memory is full,

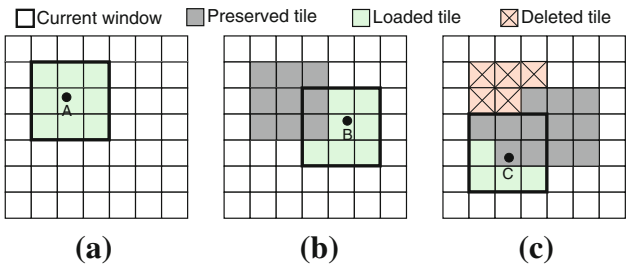

Fig. 10 LRU paging scheme used in the server to allocate terrain tiles in GPU memory. In the example, a GPU memory limit of 16 tiles is assumed

unused tiles are discarded according to a least recently used (LRU) replacement algorithm, as depicted in Fig. 10c.

Moreover, we can expect that most panorama requests will come from users physically located in densely populated areas, for example, urban areas or motorways. Thus, tiles will likely be reused, drastically reducing the number of disk accesses and data transfers.

Once the set of tiles that are visible from the current viewpoint are available in GPU memory, we build on the fly a restricted quadtree hierarchy similar to [29] for each selected tile. These structures are used by the GPU to draw the panorama.

6.2 The Panorama Encoder

The Panorama Encoder module is fed with the raw panoramas generated by the Panorama Renderer. Here, these panoramas are encoded in a compressed format suitable for both network transmission and fast decoding by mobile clients. In our implementation, we use the JPEG format. In order to reduce the overall encoding time, multiple instances of this module can be run in parallel on different threads.

7 Client-side architecture

In our architecture, users should install and run a dedicated application on their mobile devices. In our implementation, the client application has been developed as a plain C++ native program using the industry-standard 3D graphics library OpenGL ES [26]. Figures 1 and 11 show some snapshots of this application.

Fig. 11 Some snapshots on an iPhone

7.1 Interface

As our goal was to provide an immersive experience to the user, our application tries to match the virtual view offered on the screen with the user's current view in the physical world, see Fig. 1. This is accomplished by obtaining the user's geographical position and the view direction from, respectively, the mobile built-in GPS receiver and the electronic compass.

The values obtained from these sensors automatically drive the user's viewpoint in the virtual world. This automatic movement scheme reduces and simplifies the user interaction required to use the system. However, an unrestricted maneuvering mode following the flying metaphor [39] is also provided, allowing users to freely locate areas of their interest. In this mode, users explicitly control the navigation around their geographical space by using the device's keyboard or touch screen [22].

7.2 Client architecture

The client-side application has been designed as a modular application as depicted in Fig. 7. The *Local Database* stores the scene, the *Visualization Module* manages the user interface and renders the scene, and the *Database Updater Module* processes the 3D map and panoramas provided by the server. Also, to reduce the CPU load in the main thread, networking tasks are moved to a second thread that manages the communication with the server and which, in parallel, decodes JPEG textures and panoramas. These elements will be described below.

The client Local Database, unlike its server's counterpart, resides in the main memory of the client. It serves as a temporal repository where those components of the scene needed for rendering are stored. The Local Database maintains a very small subset of the complete terrain dataset, consisting of a small grid of incomplete quadtrees centered on the viewer and the panorama currently being displayed.

The information stored in the Local Database is used by the Visualization Module to render the scene according to the current viewer position. This module also handles the user interaction.

Finally, the Database Updater Module of the client takes care of updating the Local Database dynamically, according to the current needs of the application. This module is in charge of the following tasks:

1. It determines whether new terrain data should be downloaded from the server, issuing a request if needed. It also discards unneeded parts of the terrain.

2. It assesses the error of the current panorama, see Sect. 3.2, and requests a new one whenever it must be updated.
3. It adds to the Local Database the information coming from the server.
4. It constructs a triangulated mesh that approximates the 3D terrain according to the current view and the quadtree data structure, see [36]. The level of detail is computed based on the distance to the viewpoint.

8 Performance evaluation

In order to evaluate the effectiveness of the proposed architecture, we have implemented a prototype and carried out an exhaustive analysis of its performance and scalability. The aim is to prove that the proposed architecture allows for highly interactive visualization of photorealistic 3D maps on mobile devices connected through low-bandwidth wireless networks.

In Sect. 1, we affirmed that the proposed architecture does not require any expensive hardware on the server side. Therefore, we ran our experiments on an ordinary desktop PC equipped with an Intel Core-2 Duo CPU, 4 GB system memory, an NVIDIA GeForce 8800 GPU, and a commodity S-ATA hard disk. In our tests, both the Main Server and the Panorama Server components were run in the same computer.

We used in our experiments the Puget Sound terrain,[1] a typical dataset used in terrain rendering benchmarking. It is made up of 16,536 × 16,536 elevation samples with a horizontal resolution of 10 m and a vertical resolution of 0.1 m. The texture map included 16,536 × 16,536 pixels, with a resolution of 10 m per pixel.

Section 8.1 studies the performance and interactivity of the client, as well as the impact of the network on the navigation. Section 8.2 focuses on studying the general performance of the server. Finally, Sect. 8.3 analyzes the scalability of the architecture.

8.1 Client performance

Our experiments were carried out on an Apple iPhone 3GS mobile phone connected to the server through two popular real-world cellular networks: *UMTS* (Universal Mobile Telecommunication System) and *GPRS* (General Packet Radio Service). These networks are usually known as 3G and 2G, respectively. While 3G provides better bandwidth and latencies, 2G is usually the only available network in large rural areas.

[1] Available at http://www.cc.gatech.edu/projects/large_models/ps.html [accessed 28 May 2012].

To study the performance of the mobile device, we connected it to the server and simulated an user navigating across the virtual environment. The simulation consisted of performing a rectilinear flyover at a constant speed of 150 km/h and a constant height of 200 m over the terrain. The experiments were performed using both networks, 3G and 2G. In both cases, the same trajectory was followed. To avoid false results, the terrain boundaries were never reached. The minimum viewing distance was 30 km, the panoramas were placed 7.5 km away from the client, and their resolution was 256^2 pixels per skybox face. We used the panorama updating criteria reported in [36] with a maximum allowed error of 5% pixels. Each test lasted 300 s.

The goal of this experiment was to measure a set of objective parameters during the flyovers. Figure 12a compares the performance over time obtained when using 3G and 2G. The studied parameters are the evolution over time of (from top to bottom): the amount of triangles rendered in each frame, the frame rate, and the downloading measured in kB.

During the tests, the system was configured to maintain a target rendering speed of 30 frames per second. As shown in Fig. 12a, the tested device was able to render around 30k triangles per frame while guaranteeing this frame rate. This translates as a smooth navigation and a constant image quality during the whole test. The upper plot in Fig. 12a also shows that, at the beginning of the tests, the iPhone required a longer time to achieve 30k triangles when the 2G network was used. But apart from that, the curves are constant and almost coincident in both cases, 2G and 3G, proving that our architecture is capable of providing a smooth user experience regardless of the network used.

Another important practical consideration that was investigated was the impact of the network reliability on the performance and image quality. Apart from the low bandwidth and high latencies, probably the largest problem with cellular networks is their high level of unreliability. Hence, developing fault tolerant networked applications becomes an important issue.

In order to evaluate this, we repeated the previous tests under the same conditions, but a network stall of 60 s was simulated at second 100. Results are shown in Fig. 12b. During this stall, no data were transmitted from or to the server, see bottom diagram in Fig. 12b, precluding the download of terrain and panorama updates. Since the local database of the client stores the area close to the user, it provides an effective tool to mitigate the impact of occasional network failures. Consequently, during the network stall, the application still provides a smooth navigation to the user across the virtual environment.

As can be seen in the upper diagram of Fig. 12b, during the stall, the number of triangles presented a logarithmic

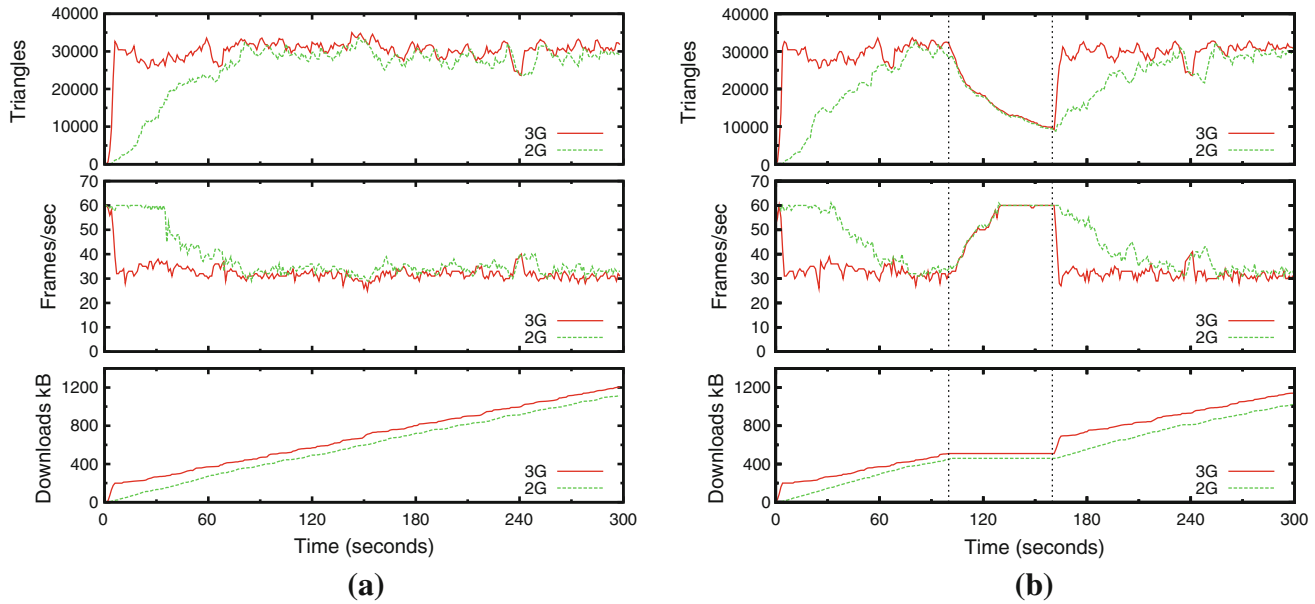

Fig. 12 Client performance using UMTS (3G) versus GPRS (2G) when moving at 150 km/h. In **b**, a network stall of 60 s is introduced at second 100. *Top* number of triangles rendered. *Middle* frame rate achieved. *Bottom* data transferred (kiloBytes)

decay. This stems from the fact that the user is moving forward but no new terrain data can be fetched from the server. Therefore, the terrain stored in the local database gradually gets behind the viewer. This also explains the frame rate increase. It was also impossible to update the panorama used to portray distant terrain. At second 160, the number of triangles used to render the scene had dropped from 30k to 10k, which still offers adequate rendering quality. As soon as the network link is restored at second 160, the system rapidly updates the panorama and streams the missing terrain, thus restoring the prior image quality. Again, the test with 2G required a longer time to achieve the quality provided by 3G.

Note that in this situation, server-side rendering techniques described in Sect. 2.1 would cause the whole application to stall for 60 s, probably causing the user to give up the application.

8.2 Server performance

The server-side part of the architecture was also evaluated with empirical studies. As explained in Sect. 4, the server was composed of two components: the Main Server and the Panorama Server. Since the performance of the Main Server simply depends on the server's hard disk speed, we focus here on analyzing the Panorama Server.

We used the following methodology: we connected one client to the server and performed a fly-over at a constant height of 200 m over the terrain. We generated 100 panoramas with different viewer positions and recorded several measurements along the process. Table 1 shows the average performance values yielded by the server during the experiment. From left to right, Table 1 lists the resolution of the panorama defined as the resolution of the panorama skybox faces, the JPEG compression quality used, the time needed by the Panorama Renderer module (Sect. 6.1) to generate all the images in one panorama, the time required by the Panorama Encoder module (Sect. 6.2) to perform the JPEG encoding for one panorama, and the compressed panorama size.

Table 1 Experimental values for different panorama resolutions and JPEG encoding quality

Panorama resolution	JPEG quality	Rendering time (s)	Encoding time (s)	Panorama size (kB)
256^2	60	0.00544	0.00996	10.64
256^2	80	0.00536	0.01076	12.48
512^2	60	0.00952	0.02932	30.40
512^2	80	0.00976	0.02972	35.80
$1{,}024^2$	60	0.02528	0.11484	101.92
$1{,}024^2$	80	0.02484	0.11528	118.76

We observe from Table 1 that the time needed to render and encode a panorama increases with the resolution. However, the JPEG compression quality required does not seem to have an effect on the processing time. It just affects the size of the resulting panorama. On the other hand, encoding time is always significantly longer than rendering time. This suggests that the encoding phase is the most expensive phase of the panorama generation process. Differences become larger as the panorama resolution increases.

In our tests, encoding time ranged between 0.00996 s for the 256^2 resolution and 60 JPEG quality scenario, and 0.11528 s, for $1{,}024^2$ resolution and 80 JPEG quality scenario. The first scenario would allow an encoding rate of 100.40 panoramas per second, while the second would allow an encoding rate of 8.67 panoramas per second. These figures give us an approximate upper limit for the number of clients per second that can be provided with panoramas.

8.3 Scalability

In order to assess the scalability of our architecture, we carried out a set of experiments with an increasing number of connected clients. For each test, clients simultaneously established a connection to the server and performed a rectilinear flyover using the same conditions described in Sect. 8.1. The starting point and the flight direction of each client were random values. The navigation speed was also a random value in the range, 100–150 km/h.

As shown in Fig. 13, mobile clients were simulated by using a cluster of up to 32 PCs, each one running 8 instances of the client application. Each client was locally rendering around 10k triangles. Note that, from the server point of view, there is no practical difference between an actual and a simulated mobile client.

For each test, we recorded a set of measures in the server and in the clients. These results will be discussed below.

8.3.1 Scalability measured from the server side

In what follows, consider that in our experiments a panorama response consisted of a message containing six JPEG textures of a panorama at a given resolution. Similarly, a quadtree node response contained four brother quadtree nodes, each consisting of a 9×9 height map (2-bytes per height value) and a 64×64 pixels JPEG texture. These messages were generated by the server in response to, respectively, a panorama request and a quadtree node request.

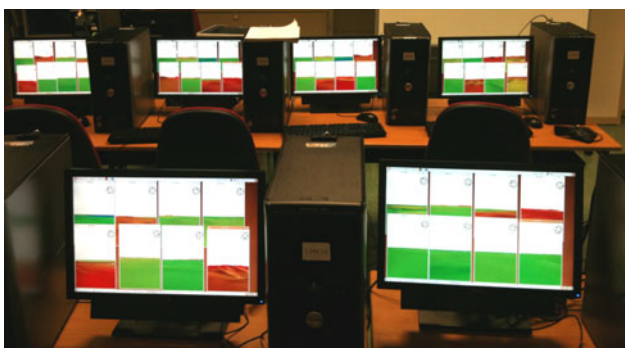

Fig. 13 Some PCs of the 32-node cluster used in our experiments. Each node was simulating 8 mobile clients

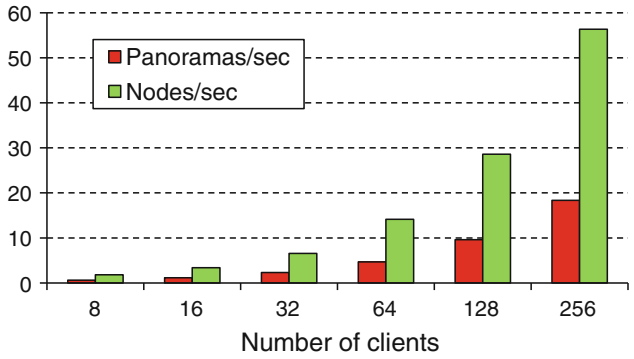

Fig. 14 Average number of requests received per second by the server for an increasing number of clients

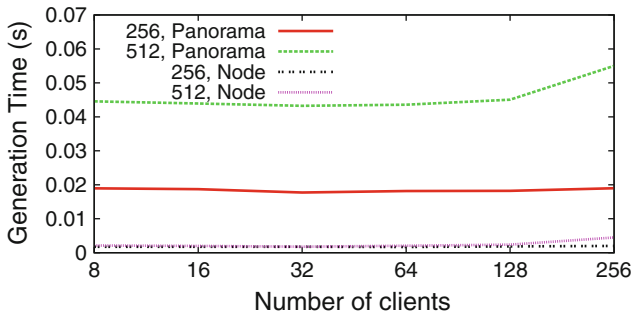

Fig. 15 Average time in seconds needed by the server to generate a response to a panorama or quadtree node request for an increasing number of clients

First, we studied the relation between the number of connected mobile clients and the number of requests received by the server. This relation can provide an idea of the server workload under different scenarios. The diagram in Fig. 14 depicts the average number of panorama and quadtree node requests received per second by the server for an increasing number of clients. The bars in this diagram show that, as expected, the server workload is directly linear with the number of clients.

Second, the average time required by the server to compute responses to the client's requests was measured. The aim was to evaluate the impact of the server's workload on its response times. The diagram in Fig. 15 shows the average time in seconds needed by the server to generate a response to a quadtree node or panorama request for an increasing number of clients. This last measure includes loading the height and texture map from disk. Two alternative scenarios are compared. In the first, all clients requested panoramas with a resolution of 256^2 pixels per skybox face, whereas in the second, all clients requested 512^2 panoramas.

As can be clearly seen in Fig. 15, our measurement shows that all the response generation curves are almost horizontal, which proves that our architecture manages to

provide a constant performance and response times regardless of the number of connected clients. This is achieved mainly due to the low rate of panorama requests received by the server reported in Fig. 14. At automobile-like speeds, a client only issues a panorama requests at intervals of a few seconds, which can be easily managed by a commodity PC acting as server, even for hundreds of concurrent clients.

8.3.2 *Scalability measured from the client side*

Finally, we wished to demonstrate that increasing the number of connected mobile clients to the same server does not reduce the quality of the scene rendered by the clients. Therefore, we measured the performance on the client side for an increasing number of clients connected to the same server. Here, we used the same conditions as in the previous test. Figure 16 illustrates the average values obtained by all the clients. The left *Y*-axis shows the average number of triangles rendered per frame whereas the right *Y*-axis shows the average number of panoramas downloaded from the server during the whole flyover by each client. The panorama resolution used in this test was 512^2.

In all cases, the sustained number of rendered triangles per frame was about 10k triangles. The almost horizontal curves in Fig. 16 indicate that our architecture manages to provide a constant image quality and a constant navigating experience regardless of the number of connected clients. Up to 256 clients connected to the same server did not decrease the quality of the scene rendered by each client. The same rationale also applies to the number of panoramas streamed from the server.

9 User study

The assessment performed in the previous section was mainly based on objective parameters, and the system effectiveness was measured in terms of performance and scalability. In this section, we follow a second approach, and we have measured the subjective user satisfaction for the system. This approach allows us to complete our evaluation and to study subjective parameters such as visual quality.

We recruited 22 subjects (14 males and 8 females) with ages ranging from 22 to 44 years, averaging at 29. They were all smart phone users. The study was carried out on an Apple iPhone 3GS, and we used the same terrain dataset described in the previous section. Users had to navigate the 3D map from their starting position to a specific location (around 10 km) using the touch screen as input and at a maximum speed of 400 km/h and a constant height of 200 m over the terrain. Evaluators performed this task two times, each one using a different network connection (3G and 2G) in random order. After finishing an experiment, the Local Database of the mobile application was flushed, that is, each experiment required downloading the 3D map from scratch.

After completing both experiments, evaluators filled out a usability questionnaire, one per network used. This questionnaire contained seven predefined questions, which were answered using a seven-point Likert scale where 1 means "strongly disagree" and 7 means "strongly agree". The questionnaire is reproduced below.

- Q1 Loading times are low.
- Q2 The 3D map updates adequately as I move.
- Q3 The application has a good performance and runs smoothly.
- Q4 I do not notice important changes nor discontinuities in the distant terrain.
- Q5 The application provides a good viewing distance.
- Q6 The 3D map is realistic and very similar to the actual world.
- Q7 Overall, I'm satisfied with this system.

Figure 17 collects the subjective ratings obtained from our usability test.

The purposes of Q1 and Q2 were to study the effect of the network on the usability of the application. As expected, the system received a considerably better evaluation

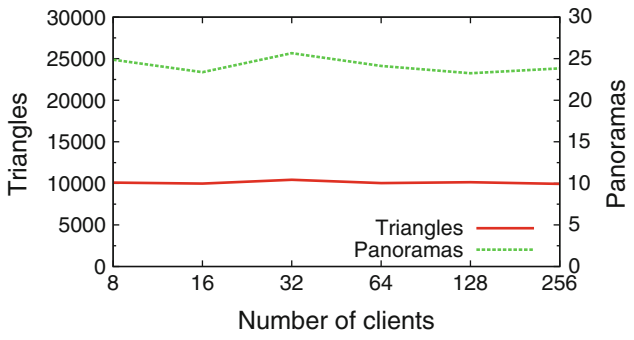

Fig. 16 Averaged client performance for an increasing number of clients connected to the same server

Fig. 17 Usability questionnaire. Average replies to the questions

when the 3G network was used. In this case, loading times were almost nonexistent, and map updates when users moved around were difficult to notice. However, when the 2G network was used, the application needed a start-up time of several seconds to provide sufficient visual quality, as shown in Fig. 12. Nevertheless, evaluations were still positive (above 4), which we consider a good result considering the extremely low performance of 2G networks.

The users' evaluation also demonstrates that the performance of the system is very high (Q3), regardless of the network being used. This result confirms the objective study performed in Sect. 8.1. The achieved frame rate of 30 fps translates to a fluid and consistent user experience.

Questions Q4, Q5, and Q6 are mainly related to visual quality, which is difficult to evaluate with objective studies. The purpose of Q4 was to check whether the panorama provides enough quality to effectively replace actual geometry for distant scenery. It also allowed us to determine whether transitions between consecutive panoramas were easy to notice. The results were very positive, proving that most evaluators did not notice anything unusual in the distant mountains. In general, users' attention was focused on nearby parts of the scene, and subtle transitions between consecutive panoramas usually went unnoticed. Only some users were able to detect such transitions under certain situations, specifically after a fast vertical movement caused by going down a steep mountain at high speed. Apart from that, the 3D map was found to be very appealing by the evaluators, who uniformly praised the high quality and fidelity of the map when compared to the real world (Q5, Q6).

Finally, when the participants were requested to directly evaluate the system (Q7), they gave a very favorable evaluation even in the 2G network scenario. In conclusion, although the number of evaluators was limited, this evaluation provided clear evidence that the proposed architecture delivers a pleasant user experience and good image quality.

10 Summary and future work

Due to the limited computing resources and restricted bandwidth available in current mobile device technologies, designing systems for adaptive streaming and rendering of large terrains over wireless networks for mobile devices is a challenging task. In this paper, we have described a complete and scalable client–server architecture that successfully overcomes these limitations. The architecture is based on a hybrid rendering technique that splits the rendering workload between a remote server and the mobile clients.

In order to assess scalability and performance robustness, we carried out an exhaustive analysis of the client and server performance with respect to different network scenarios and the number of simultaneously connected clients. Contrarily to most server-based rendering approaches found in the literature, our results show that a commodity PC is capable of providing a smooth navigation to a large number of concurrent clients.

Future work includes putting this technology into practice in order to develop applications such as context-based mobile 3D guides and collaborative virtual environments. We also plan to investigate effective ways to incorporate additional data layers from GIS databases into our 3D environment, for example, points of interest, roads, cadastral maps and the like.

Acknowledgments This work has been partially supported by the Consejería de Innovación, Ciencia y Empresa of the Junta de Andalucía and the European Union (via ERDF funds) through the research project P07-TIC-02773.

References

1. Akenine-Möller T, Ström J (2008) Graphics processing units for handhelds. Proc IEEE 96(5):779–789 doi:10.1109/JPROC.2008.917719
2. Aranha M, Dubla P, Debattista K, Bashford-Rogers T, Chalmers A (2007) A physically-based client–server rendering solution for mobile devices. In: MUM '07: proceedings of the 6th international conference on mobile and ubiquitous multimedia. ACM, New York, NY, USA, pp 149–154. doi:10.1145/1329469.1329489
3. Arikawa M, Konomi S, Ohnishi K (2007) Navitime: supporting pedestrian navigation in the real world. Pervasive Comput IEEE 6(3):21–29. doi:10.1109/MPRV.2007.61
4. Blinn JF, Newell ME (1976) Texture and reflection in computer generated images. Commun ACM 19(10):542–547. doi:10.1145/360349.360353
5. Bouatouch K, Point G, Thomas G (2005) A client–server Approach to image-based rendering on mobile terminals. Research report RR-5447, INRIA. http://hal.inria.fr/inria-00000127/en/
6. Boukerche A, Jarrar R, Pazzi R (2009) A novel interactive streaming protocol for image-based 3D virtual environment navigation. In: IEEE international conference on communications, 2009. ICC '09, pp 1–6. doi:10.1109/ICC.2009.5198649
7. Boukerche A, Jarrar R, Pazzi RW (2008) An efficient protocol for remote virtual environment exploration on wireless mobile devices. In: WMuNeP '08: proceedings of the 4th ACM workshop on Wireless multimedia networking and performance modeling. ACM, New York, USA, pp 45–52. doi:10.1145/1454573.1454584
8. Burigat S, Chittaro L (2005) Location-aware visualization of vrml models in gps-based mobile guides. In: Web3D '05: proceedings of the tenth international conference on 3D Web technology. ACM, New York, NY, USA, pp 57–64. doi:10.1145/1050491.1050499
9. Chang CF, Ger SH (2002) Enhancing 3d graphics on mobile devices by image-based rendering. In: PCM '02: proceedings of the third IEEE Pacific Rim conference on multimedia. Springer, London, UK, pp 1105–1111
10. Chehimi F, Coulton P, Edwards R (2008) Evolution of 3D mobile games development. Pers Ubiquit Comput 12:19–25. doi:10.1007/s00779-006-0129-9

11. Demers M (2008) Fundamentals of geographic information systems. Wiley, London
12. Diepstraten J, Gorke M, Ertl T (2004) Remote line rendering for mobile devices. In: CGI '04: proceedings of the computer graphics international. IEEE Comput Soc, Washington, DC, USA, pp 454–461. doi:10.1109/CGI.2004.86
13. Duguet F, Drettakis G (2004) Flexible point-based rendering on mobile devices. IEEE Comput Graph Appl 24(4):57–63. doi:10.1109/MCG.2004.5
14. Epic Games, Inc. (2012) Unreal engine. http://udk.com/mobile. Accessed 14 June 2012
15. Foley J, van Dam A, Feiner S, Hughes J (1990) Computer graphics: principles and practice, 2nd edn. Addison-Wesley Longman, Boston
16. Google (2012) Google Earth for mobile devices. http://www.google.com/mobile/earth/. Accessed 14 June 2012
17. He Z, Liang X (2007) A multiresolution object space point-based rendering approach for mobile devices. In: AFRIGRAPH '07: proceedings of the 5th international conference on computer graphics, virtual reality, visualisation and interaction in Africa. ACM, New York, NY, USA, pp 7–13. doi:10.1145/1294685.1294687
18. Hekmatzada D, Meseth J, Klein R (2002) Non-photorealistic rendering of complex 3d models on mobile devices. In: 8th annual conference of the international association for mathematical geology, vol 2. Alfred-Wegener-Stiftung, pp 93–98
19. Hildebrandt D, Klimke J, Hagedorn B, Döllner J (2011) Service-oriented interactive 3d visualization of massive 3d city models on thin clients. In: Proceedings of the 2nd international conference on computing for geospatial research and applications, COM.Geo '11. ACM, New York, NY, USA, p 6:1. doi:10.1145/1999320.1999326
20. Huang J, Bue B, Pattath A, Ebert DS, Thomas KM (2007) Interactive illustrative rendering on mobile devices. IEEE Comput Graph Appl 27:48–56. doi:10.1109/MCG.2007.63
21. Humphreys G, Houston M, Ng R, Frank R, Ahern S, Kirchner PD, Klosowski JT (2002) Chromium: a stream-processing framework for interactive rendering on clusters. ACM Trans Graph 21(3):693–702. doi:10.1145/566654.566639
22. Hürst W, Helder M (2011) Mobile 3D graphics and virtual reality interaction. In: Proceedings of the 8th international conference on advances in computer entertainment technology, ACE '11. ACM, New York, NY, USA, pp 28:1–28:8. doi:10.1145/2071423.2071458
23. ImageVis3D: ImageVis3D: A real-time volume rendering tool for large data. Scientific computing and imaging institute (SCI). http://www.imagevis3d.org (2011). URL http://www.imagevis3d.org. Accessed 20 Feb 2012
24. Jeong S, Kaufman AE (2007) Interactive wireless virtual colonoscopy. Vis Comput 23(8):545–557. doi:10.1007/s00371-007-0117-8
25. Kenteris M, Gavalas D, Economou D (2011) Electronic mobile guides: a survey. Pers Ubiquit Comput 15:97–111. doi:10.1007/s00779-010-0295-7
26. Khronos Group (2010) OpenGL ES—The standard for embedded accelerated 3D graphics. http://www.khronos.org/. Accessed 24 Mar 2010
27. Lamberti F, Sanna A (2007) A streaming-based solution for remote visualization of 3D graphics on mobile devices. IEEE Trans Vis Comput Graph 13(2):247–260. doi:10.1109/TVCG.2007.29
28. Lerbour R, Marvie JE, Gautron P (2009) Adaptive streaming and rendering of large terrains: a generic solution. In: 17th WSCG international conference on computer graphics, visualization and computer vision
29. Livny Y, Kogan Z, El-Sana J (2009) Seamless patches for GPU-based terrain rendering. Vis Comput 25(3):197–208. doi:10.1007/s00371-008-0214-3
30. Lluch J, Gaitán R, Escrivá M, Camahort E (2006) Multiresolution 3d rendering on mobile devices. In: Alexandrov V, van Albada G, Sloot P, Dongarra J (eds) Computational science—ICCS 2006. Lecture notes in computer science, vol 3992. Springer, Berlin, pp 287–294
31. Longley PA, Goodchild MF, Maguire DJ, Rhind DW (2005) Geographic information systems and science. Wiley, London
32. Luley P, Perko R, Weinzerl J, Paletta L, Almer A (2012) Mobile augmented reality for tourists marft. In: Gartner G, Ortag F (eds) Advances in location-based services, lecture notes in geoinformation and cartography. Springer, Berlin, pp 21–36. doi:10.1007/978-3-642-24198-7_2
33. Martin IM (2000) Adaptive rendering of 3D models over networks using multiple modalities. Tech Rep RC 21722, IBM T.J. Watson Research Center
34. NaviGenie (2012) NaviGenie 2.0. http://www.navigenie.com/. Accessed 14 June 2012
35. Noguera JM, Barranco M, Segura RJ, Martínez L (2012) A mobile 3D-GIS hybrid recommender system for tourism. Inf Sci 215(0):37–52. doi:10.1016/j.ins.2012.05.010
36. Noguera JM, Segura RJ, Ogáyar CJ, Joan-Arinyo R (2011) Navigating large terrains using commodity mobile devices. Comput Geosci 37(9):1218–1233. doi:10.1016/j.cageo.2010.08.007
37. Nurminen A (2008) Mobile 3d city maps. IEEE Comput Graph Appl 28:20–31. doi:10.1109/MCG.2008.75
38. Nurminen A, Oulasvirta A (2008) Designing interactions for navigation in 3d mobile maps. In: Meng L, Zipf A, Winter S (eds) Map-based mobile services. Lecture notes in geoinformation and cartography. Springer, Berlin, pp 198–227
39. Oulasvirta A, Estlander S, Nurminen A (2009) Embodied interaction with a 3D versus 2D mobile map. Pers Ubiquit Comput 13:303–320. doi:10.1007/s00779-008-0209-0
40. Pajarola R (1998) Large scale terrain visualization using the restricted quadtree triangulation. In: VIS '98: proceedings of the conference on visualization '98. IEEE Computer Society Press, Los Alamitos, CA, USA, pp 19–26
41. Pajarola R, Gobbetti E (2007) Survey of semi-regular multiresolution models for interactive terrain rendering. Vis Comput 23(8):583–605. doi:10.1007/s00371-007-0163-2
42. Paravati G, Sanna A, Lamberti F, Ciminiera L (2011) An open and scalable architecture for delivering 3D shared visualization services to heterogeneous devices. Concurr Comput Pract Exp 23(11):1179–1195. doi:10.1002/cpe.1695
43. Pazzi R, Boukerche A, Huang T (2008) Implementation, measurement, and analysis of an image-based virtual environment streaming protocol for wireless mobile devices. Instrum Meas IEEE Trans 57(9):1894–1907. doi:10.1109/TIM.2008.919901
44. Pouderoux J, Marvie J (2005) Adaptive streaming and rendering of large terrains using strip masks. In: GRAPHITE '05: proceedings of the 3rd international conference on computer graphics and interactive techniques in Australasia and South East Asia. ACM, New York, USA, pp 299–306. doi:10.1145/1101389.1101452
45. Quillet JC, Thomas G, Granier X, Guitton P, Marvie JE (2006) Using expressive rendering for remote visualization of large city models. In: Web3D '06: proceedings of the eleventh international conference on 3D web technology. ACM, New York, NY, USA, pp. 27–35. doi:10.1145/1122591.1122595
46. Rakkolainen I, Vainio T (2001) A 3d city info for mobile users. Comput Graph 25(4):619–625. doi:10.1016/S0097-8493(01)00090-5. (Intelligent interactive assistance and mobile multimedia computing)
47. Samet HJ (1989) Design and analysis of spatial data structures: quadtrees, octrees, and other hierarchical methods. Addison–Wesley, Redding

48. Shankel J (2001) Game programming gems 2. Charles River Media Inc., Rockland
49. Silva WB, Rodrigues MAF (2009) A lightweight 3D visualization and navigation system on handheld devices. In: SAC '09: proceedings of the 2009 ACM symposium on applied Computing. ACM, New York, NY, USA, pp 162–166. doi:10.1145/1529282.1529318
50. Suárez JP, Trujillo A, de la Calle M, Gómez DD, Santana JM (2012) An open source virtual globe framework for iOS, Android and WebGL compliant browser. In: Proceedings of the 3rd international conference on computing for geospatial research and applications, COM.Geo '12. ACM, New York, NY, USA
51. Unity Technologies (2012) Unity. http://unity3d.com/. Accessed 14 June 2012
52. Wen J, Wu Y, Wang F (2009) An approach for navigation in 3D models on mobile devices. In: CMRT09: city models, roads and traffic. Paris, France, pp 109–114
53. Wen J, Zhu B, Wang F (2008) Real-time rendering of large terrain on mobile device. In: The international archives of the photogrammetry, remote sensing and spatial information sciences, vol XXXVII. Part B5. Beijing, pp 693–697
54. Yoo W, Shi S, Jeon W, Nahrstedt K, Campbell R (2010) Real-time parallel remote rendering for mobile devices using graphics processing units. In: 2010 IEEE international conference on multimedia and Expo (ICME), pp 902–907. doi:10.1109/ICME.2010.5583022