A Hybrid Client-Server Based Technique for Navigation in Large Terrains Using Mobile Devices

José M. Noguera, Rafael J. Segura, Carlos J. Ogáyar
Grupo de Gráficos y Geomática
Universidad de Jaén

Robert Joan-Arinyo
Grup d'Informàtica a l'Enginyeria
Universitat Politècnica de Catalunya

January 29, 2010

Abstract

We describe a hybrid client-server technique for remote adaptive streaming and rendering of large terrains in resource-limited mobile devices. The technique has been designed to achieve an interactive rendering performance on a mobile device connected to a low-bandwidth wireless network. The rendering workload is split between the client and the server. The terrain area close to the viewer is rendered in real-time by the client using a hierarchical multiresolution scheme. The terrain located far from the viewer is portrayed as view-dependent impostors, rendered by the server on demand and, then sent to the client. The hybrid technique provides tools to dynamically balance the rendering workload according to the resources available at the client side and to the saturation of the network and server.

A prototype has been built and an exhaustive set of experiments covering several platforms, wireless networks and a wide range of viewer velocities has been conducted. Results show that the approach is feasible, effective and robust.

Keywords: Terrain navigation, Terrain rendering, Mobile computing, 3D graphics, Software Portability, Adaptive streaming.

1 Introduction

Nowadays, mobile devices with computational capabilities like mobile phones and Personal Digital Assistants (PDA) are ubiquitous. Following the general framework depicted in Figure 1, they are used as interactive guides to real environments, offer features such as Global Positioning Systems (GPS), access to location-based data and, visualization of maps and, terrain rendering.

Terrain rendering is arguably a central technology in a number of fields. In personal navigation and Geographic Information Systems (GIS) terrain rendering plays an outstanding role. Here, real time terrain rendering is a must.

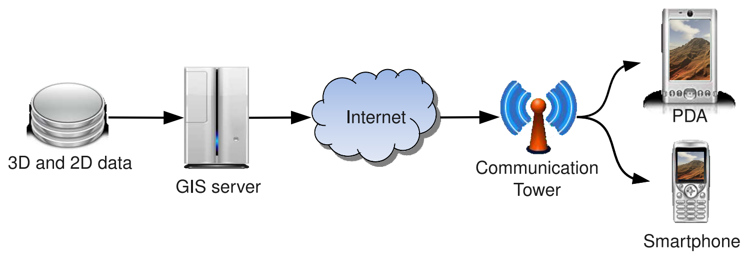


Figure 1: Mobile computing devices in a general data-system framework.

Real-time adaptive streaming and rendering of large terrains has always been considered a complex task that needs a large amount of computational power, random-access memory and network bandwidth.

Many visualization and streaming techniques have been developed for desktop computers and workstations. However, there are several technical limitations that preclude using them in mobile devices. Mobile devices lack of the CPU power required to properly handle complex data structures and the huge amount of data involved in terrain represenation. Besades, mobile devices in general do not include 3D graphics hardware thus GPU-based solutions cannot be considered yet. Moreover, the limited random-access memory featured along with slow wireless commercial networks pose great difficulties to process large amount of data on mobile devices.

Some techniques move geometry-rendering tasks to a dedicated remote rendering server that streams images to a client over a network to be displayed locally. Although these techniques are more akin to mobile devices, they easily lead to network congestion and poor performance.

In this paper we describe a technique for wireless remote streaming and rendering of terrains on mobile phones and PDAs using low bandwidth networks, such as General Packet Radio Service (GPRS) and Universal Mobile Telecommunications System (UMTS), in the general framework illustrated in Figure 1. Preliminary versions of this work can be found in [NSOJA09, SRNF08].

We propose a client-server hybrid rendering approach. The client renders the geometry of the terrain close to the viewer, whereas the distant terrain is portrayed as impostors which are rendered by the server and streamed to the client. Since impostors represent terrain distant from the viewer, they do not need to be updated unless the user's position in the virtual environment changes beyond a given threshold. The approach provides tools to dinamically split rendering task between the server and the client according to the resources available in the client and the network congestion.

To cope with the limited local random-access memory available in mobile devices at the client end, the terrain is subdivided into patches organized into a hierarchical data structure which provides an increasing level of detail. The smallest possible amount of patches are streamed to the client on demand. Cracks in terrain points shared by patches are avoided by rendering in a seamless and adaptive way precomputed triangle strips. This rendering method is simple, fast and fully compatible with the progressive streaming nature of our application.

As a proof of concept, we have implemented a prototype and an exhaustive set of experiments

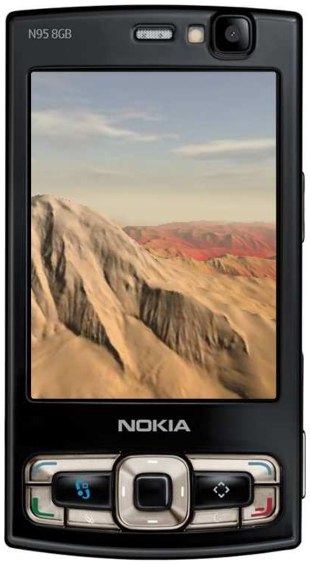


Figure 2: Streaming and rendering of the Puget Sound dataset on a Nokia N95 at 31 fps, using 12120 triangles and connected through an UMTS wireless network.

has been conducted. Experiments included several platforms, wireless networks and viewer velocities ranging from humane walking to airplane jet speed. The prototype applies progressive downloading of data over low bandwidth wireless network, such as GPRS and UMTS. Optimized database management, fast visualization algorithms, impostors, a lightweight network scheme and a smart cache management, have been used in the implementation. The prototype achieves terrain visualization at interactive rates on mobile devices. Results show that the approach is feasible, effective and robust. Figure 2 shows an actual display generated by the prototype.

The paper contents is organized as follows. Related and relevant work published in the literature is reviewed in Section 2. Section 3 reviews the mobile phone technologies and 3D graphics standards existing in the market. We also reports a preliminary assessment of mobile devices and available libraries. Section 4 recalls principles concerning the software layer model to decouple applications from specific platforms and, describes our specific layout. In Section 5 we recall basic concepts of viewing in 3D that will be used in the following sections. The main components of the hybrid approach are described in Section 6. In Section 7 we define the policy to manage updating the scene rendered on the mobile devices and outline the visualization algorithm. Section 8 is devoted to describe the multi-threading architecture at the server and at the client, networking and, system scalability. Results are reported and discussed in Section 9. We close with Section 10 offering a summary and outlining future work.

2 Previous Work

According to where the geometry-rendering task is performed, current networked rendering methods can be divided into two categories In one category a dedicated remote rendering server is in charge of performing the geometry-rendering task and streaming images to a client over a network. Altholugh these techniques are appropriate for thin mobile devices, they have some drawbacks. To achieve an interactive rate, these techniques stream a massive volume of images to the client, what can easily result in a congested network. Moreover, image quality usually suffers beacuse of the lossy compression algorithms used to reduce the trafic. In the other category rendering tasks are delegated to the client. This approach reduces the streaming load however the client must provide the computational power required to render good quality and to manage a depth enough viewing distance.

2.1 Server-based Techniques

Many server-based techniques have been reported in the literature. For example, Lamberti et al [LS05] and Jeong and Kaufman [JK07], offer solutions of image-based rendering techniques of generic scenes on mobile devices. Their approaches are based on the use of image and video compression algorithms, such as JPEG or MPEG. Jeong and Kaufman claim to achieve a speed of 5 fps (frames per second) using a 802.11b radio interface on a Personal Digital Assistant (PDA).

Some authors have presented alternative methods for image-based approaches by using scheduling mechanisms and partial streaming of images. However, these approaches severely limits how the viewer can move and do not perform well in dynamic scenes. Boukerche et al. [BJP08, BPF08] present the idea of *key partial panorama*. Instead of transmitting a new image from the server when the viewer moves, only the lost pixels are streamed to the client. Unfortunately, this technique only offers reliable results when the viewer rotates or performs strafe moves.

Pazzi et al. [PBH08] use a simple prediction technique to fill a buffer in the client side that stores all the possible viewpoints where the user can go in the next move. The movement of the viewer is limited to point on a plane, the camera cannot look up or down and can rotate only in angles of 30°.

2.2 Client-based Techniques

Most networked rendering techniques dealing with large terrains found in the literature fall in the client-based category. A great amount of work has been done on the subject of large terrain rendering, and a comprehensive overview of this subject is beyond the scope of this paper. For a detailed survey, we refer the reader to [PG07]. In what follows, we will briefly discuss the distributed, network-based approaches more closely related with our work.

The goal of a view-dependent level-of-detail algorithm is to extract at run time a consistent, minimum complexity mesh that for a given view is a good approximation of the original, full detailed mesh. That is, the mesh minimizes some view-dependent error measure. Whenever the viewpoint changes, the mesh is updated to reflect the change.

Most real-time terrain triangulation and visualization algorithms assume that the entire terrain data is stored in virtual memory, and therefore, they are not useable in a remote database context [PG07], e.g. ROAM [DWS+97]. Pajarola [Paj98] proposed a solution based on a restricted quadtree triangulation. He proposed a paging system that dynamically maintains a fraction of the entire data set in main memory. A dynamic scene manager updates the visible tiles by uploading them from disk or a remote server. The triangulation of a restricted quadtree is represented by a single triangle strip, generated by a recursive traversal of the quadtree hierarchy. This representation is not the most appropiate for a GPU because it requires re-meshing the quadtree and re-sending the geometry to the GPU for every frame.

Lindstrom and Pascucci [LP01, LP02] proposed an out-of-core terrain rendering technique that relies on the virtual memory management performed by the operating system. Since, so far, mobile devices do not feature secondary memory, this solution does not apply to mobile phones. Moreover, they do not provide tools to manage streaming through the network.

Current GPUs included in general purpose computers o embedded in mobile devices are optimized for rendering stripified, indexed and batched primitives, [Pow05]. This is the reason why recent techniques try to avoid re-meshing by composing at run-time pre-assembled surface patches that can be cached in GPU memory. This way, the CPU-GPU bandwidth and the number of draw calls can be significantly reduced. However, most of these new techniques rely on the programmable GPUs, see [LH04, SW06, LKES09, DSW09] just to cite a few. Since, in general, mobile devices do not feature programmable GPUs, these techniques do not apply.

Lossaso and Hoppe [LH04] proposed the geometric clipmap applying the idea of mipmapping to terrains. The terrain is cached into a set of nested regular grids of decreasing resolution centered around the viewer. This representation is noy suitable for streaming and requires a programmable GPU.

Cignoni et al. presented the BDAM [CGG^{+}03], which assumes that all data is stored on local secondary storage unit. Then Gobbetti et al. [GMC^{+}06] and Bettio et al. [BGMP07] adapted the BDAM to a networked environment. These approaches, rely on lossy wavelet data compression algorithms which increases the CPU requirements on the client side. Furthermore, both techniques require a programmable GPU.

Lerbour et al. [LMG09] described a networked architecture for streaming and rendering terrains. The terrain is subdivided into a tree of blocks, where each block contains a set of levels of detail with increasing resolution. The blocks are adaptively loaded from a server on demand. This structure has been specifically crafted for avoiding streaming redundant data. However, paging of terrain is not addressed and geometry gaps are not avoided. No solution is proposed for streaming textures.

Livny et al. [LKES09] use an implicit quadtree to generate a planar mesh in the CPU for a given a view dependent metric. Each patch is represented by a pre-computed set of triangle strips at different resolutions. The elevation is assigned by a shader, which uses the position of each vertex to fetch and assign the appropriate height value from a cached texture. Progressive terrain streaming is not possible, because it requires to upload in the GPU memory the heightmap of the rendered area at full resolution.

Solutions based on triangle fans, while are easier to implement, result in a large number of separate primitives. Since graphics libraries for mobile devices usually only support triangle strips as primitives, e.g. M3G, [Jav05], terrain rendering algorithms based on triangle fans, such as [SW06], should be avoided when targeting mobile devices.

Cai et al. [CLS08] report on a technique that uses strip masks, that is, regular triangulations of tiles at a given resolution. The right strip mask is selected for rendering each tile at the desired resolution. Gaps are avoided by adding a significant amount of zero-area triangles along the region boundaries of every tile. However, the approach is local and no solution is proposed to handle data streaming.

2.3 Reosurce-limited Environments

Even though some authors already noticed the need to develop terrain visualization methods allowing running at interactive rates in resource-limited environments [RG97], the tendency has been clearly the opposite: Terrain visualization algorithms have grown in both complexity and computing requirements. Therefore, adaptive and streaming rendering of large terrains on mobile devices is still a largely unexplored field.

Pouderoux et al. [PM05] proposes a very simple paging approach using a grid of tiles, specifically targeting mobile devices. Terrains are adaptively rendered using tiles. The right strip mask is selected for rendering each tile at the desired resolution. Authors claim that they have managed to render a scene of 3744 triangles at 7 fps on a PocketPC Toshiba e800, using a wired 480 Mbps USB 2.0 network. This approach, although fast on very light devices, is very crude and have some drawbacks. No multi-resolution data structure is used, precluding progressive transmission of tiles. Therefore, to render a tile, all its vertices must be fetched even at its lowest level of detail. Moreover, boundary gaps are not avoided, which makes this approach useless in most applications.

As far as we know, the solutions reported in the literature concerning mobile devices do not consider the two features that characterizes them: small bandwidth wireless networks and limited computing capabilities. Our approach allows a better use of the available computational resources and network bandwidth, which is crucial to wireless streaming applications on thin devices.

3 Mobile 3D Graphics Overview

In this section we give a brief overview of the Application Programming Interfaces (API) used in mobile 3D graphics, operating systems powering mobile devices as well as a preliminary assessment of performances.

3.1 APIs for Mobile Devices

The two most widely used APIs used in programming 3D graphics of mobile devices are OpenGL ES, [Gro04], and M3G for Java Mobile Edition,[PAM$^+$07].

OpenGL ES [Gro04] is a well-defined subset of the OpenGL 3D graphics API, designed for embedded devices including mobile phones, personal digital assistants (PDAs), and video game consoles. OpenGL ES offers a flexible and powerful low-level interface between software and graphics acceleration, is royalty-free and platform independent.

OpenGL ES is typically accessed through C or C++, however the OpenGL ES API library can also be accessed through the Java Binding for OpenGL ES API (JSR-239), [JCP06]. which acts as a bridge to the underlying platform's OpenGL technology.

M3G (JSR-184), [Jav05] is a high-level API for Java Mobile Edition (JME), [bib09e]. It has been designed to be implemented on top of OpenGL ES. OpenGL ES provides the low-level transformation, lighting, and rasterization functionality for M3G. On top of that, M3G adds the scene graph and animation features. M3G offers a level of abstraction higher than that of OpenGL ES, thus developers can concentrate more on the content and less on the minor code and platform details. The M3G specification also defines tools to manage a binary scene graph file which can be parsed and rendered with little programming effort. Unfortunately, Java Binding is not widely supported and few mobile devices with 3D graphics hardware support it.

3.2 Windows Mobile

Windows Mobile is a compact operating system for mobile devices based on the Microsoft Win32 API, [bib09c]. Devices that run Windows Mobile include Pocket PCs, Smartphones and Portable Media Centers. The lack of hardware accelerated graphics in Windows Mobile devices has motivated the development of open-source software libraries that implement the OpenGL ES interface. An example is Vincent3D 3D Rendering Library, [vin04], an open source graphics library for mobile and embedded devices.

Currently, Windows Mobile based devices which have 3D graphics hardware are very sparse. One of them is the Dell Axim X50v PDA, equipped with a 3D Intel 2700G graphics processor.

3.3 Google Android

Android SDK is an open source software platform and operating system for mobile devices, built on the open Linux Kernel, developed by Google. Furthermore, it utilizes a custom virtual machine that was designed to optimize memory and hardware resources in a mobile environment, [bib09a]. It only allows developers to write managed code in the Java language. However, it does not support the standard Java JME libraries, the specific Google-developed Java libraries must be used instead. JME was designed some years ago for small handsets, whereas the Google Java libraries have been developed with the aim of running more powerful modern smartphones.

In terms of graphics capabilities, Android currently supports only the old OpenGL ES 1.0 specification. The specific API provided by Android is similar to the JME Java Bindings for the OpenGL ES API (JSR-239). However, there are some differences. AS far as we know, no Android-based 3D hardware accelerated mobile has been released to the market yet.

3.4 Apple iPhone

The iPhone is a multimedia smartphone designed by Apple Inc. It is powered by the iPhone Operating System (iPhone OS) which is based on a variant of the same basic Mach kernel that is found in Mac OS X, [bib09d]. The iPhone features a PowerVR MBX 3D Graphics Processor.

Objective-C and Cocoa API, [Dav06], is the development framework for applications running on iPhone OS provided by Apple. This language is a superset of C, which includes both syntactic and semantic features to support object-oriented programming. iPhone OS also provides the OpenGL ES framework which conforms to the OpenGL ES v1.1 specification. Therefore straight ANSI C and C++ code can be freely intermixed and linked into the same executable, thus allowing to port existing C++/OpenGL ES code to the iPhone. The standard C libraries are also available.

Distributing iPhone applications outside of the Apple online store is very limited. Loading a third party application onto the iPhone is only possible after paying a membership fee. Once a developer has submitted an application to the Apple store, Apple holds control over its distribution.

3.5 Symbian OS S60

Symbian 60 Operating System (Symbian 60) has been designed primarily for mobile phones and other mobile devices with limited resources. Symbian 60 supports both Java JME with M3G, and native C++ with OpenGL ES 1.1, [Bab05].

Currently there are a number of mobile devices and phones that run Symbian OS and that include 3D graphics hardware. Examples are Nokia N93 and Nokia N95, featuring a PowerVR MBX 3D Graphics Processor. For more S60 devices visit, for example, [bib09f].

3.6 Preliminary Performance Assessment

In order to focus our research, we conducted some preliminary tests to assess devices and libraries performances. Specifically, we tested the Dell Axim X50v PDA, and the Nokia N95 cellular phone.

The benchmark included three Digital Terrain Models (DTM) with 65×65, 129×129 and 257×257 vertices including 8192, 32768, and 131072 triangles respectively. For each DTM we considered floating-point and integer-point heights. The heights in the 65×65 DTM terrain were fixed to a constant value. Figure 3 shows the 129×129 and 257×257 floatint point heights DTMs.

Concerning Nokia N95, two versions of applications were developed. One used Java with M3G, and the other was developed as a native Symbian C++ application with OpenGL ES.

According to [bib09b], the Dell Axim X50v PDA features an Intel 2700G multimedia accelerator with 16 MB of video memory, and supports dual display capability for professional presentations. We used the OpenGL ES 1.0 PowerVR SDK toolkit, see [VR09]. Unfortunately, this API does not provide tools to deal with floating point values.

For each test, an animation of the camera rotating around the model was generated and the number of frames per second displayed was measured. Each animation was run 100 times. Table 1 shows the averaged number of frames per second for vertices encoded as 16 bits integers. When using C++ and OpenGL ES, the Nokia N95 platform outperforms Dell Axim v50 in all the cases studied. However, as the number of vertices in the terrain increases, differences are smaller. Performance of Nokia N95 under Java and M3G is similar to that of Dell Axim v50.

Results yielded by Nokia N95 when the terrain is represented with floating vertices are given

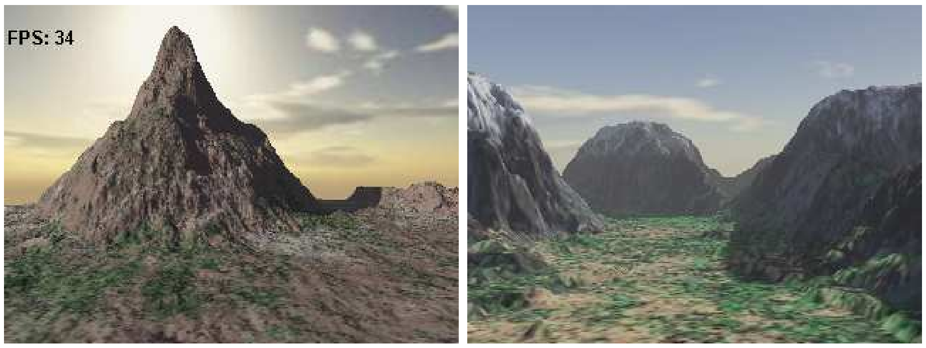


Figure 3: Screenshots of terrains used in the preliminary performance assessment. a) 129×129 vertices, b) 256×257 vertices.

in Table 2. Two things should be noticed. First terrains with size larger than 128×128 cannot be rendered at interactive speed. Second, rendering ratios for floating point terrains are very close to those yielded by integer vertices. Therefore, availability of the floating point processor must be taken into account.

With these values at hand, we selected the Nokia N95 as the hardware platform and C++ with OpenGL ES as the software tools to develop our system and run the experiments.

4 Decoupling Applications from Platforms

In a world of fragmented and rapidly changing mobile platforms with computing capabilities [PAM+07], developing applications easily portable to different platforms is a must. With this aim, we have developed a classical multilayer software architecture illustrated in Figure 4. The graphics hardware, operating system, native system API and graphics 3D API are those supplied by the platform at hand. We have developed an intermediate layer which includes the Common System API and the Common 3D API blocks in Figure 4 and that decouples our application from the specific platform. The 3D application shall be described in Section 6.

Both the Common System API and the Common 3D API blocks in the intermediate layer illustrated in Figure 4 have been implemented for Win32, POSIX-oriented operating systems (like GNU/Linux), Symbian OS UI and Windows Mobile. Both APIs are slim wrapper over the underlying native API and do not increase the overhead of the application. Our

Table 1: Rendering average performance when rendering the terrain represented as integer points.

		FPS		
Hardware	Software	65×65	129×129	257××257
Nokia N95	Java, M3G	24	22	11
Nokia N95	C++, OpenGLES	60	39	14
Dell Axim v50	C++, OpenGLES	40	18	9

Table 2: Average performance in frames per second when rendering the terrain as floating points.

		FPS		
Hardware	Software	65×65	129×129	257×257
Nokia N95	C++, OpenGLES	60	32	10

abstraction layer also allows for mobile device software to be developed and tested on a desktop workstation, providing a shorter development and debug time.

4.1 Common System API

The Common system API provides a common interface for platform-dependent tasks. Several libraries aiming at the same goal, see for example [nsp09, apr09], have been developed but, in general, they are too heavy and high resource-demanding to be useful in mobile phones. Moreover, porting them to a mobile device is expensive.

Our Common system API provides a common interface for platform-dependent tasks including window creation, user interface, control events, program main loop, image decoding, threads management, input/output and TCP/IP communication. It captures the platform-dependent events, for example, keyboard inputs, and sends them to the upper layer in a standard form shared by different platforms. It also handles several mobile device specific events, say GPS-positioning, incoming calls, low battery, touchscreen and accelerometer inputs.

4.2 Common 3D API

The Common 3D API provides a high level, object oriented common graphics library. Current multiplatform-libraries, for instance [GLF07, SFM09, GLT02], offer a limited service and do not fully abstract from the underlying platform-dependent graphics API.

Our Common 3D API is similar to other high level graphics APIs such as Java3D or M3G. It can be built upon OpenGL, OpenGL ES or Direct3D. Since vertex buffer objects are the

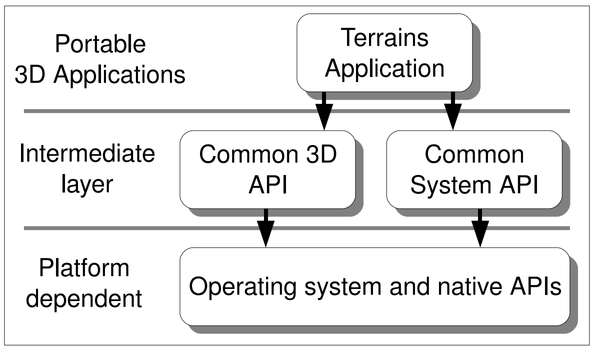


Figure 4: Software architecture.

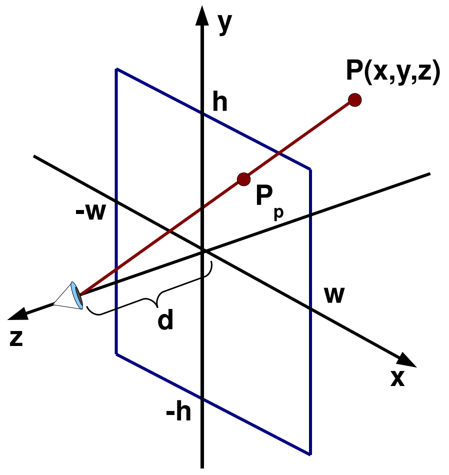


Figure 5: The projection plane and window.

most efficient way of sending geometry to the graphics processor of the mobile devices [Pow05], our API offers an exhaustive interface for handling vertex buffer objects and ordinary vertex arrays, It also manages meshes, vertex properties (normals, texture coordinates), textures, cameras, lights, fog, and the like.

5 Elements of Viewing in 3D

In Computer Graphics, 3D objects are displayed in 2D available devices by introducing projections which transform objects in real world onto projections on the plane. In this section we recall basic concepts concerning projections that will be used later on. An indepth study of 3D viewing can be found in [FvFH90].

When displaying 3D objects, three concepts must be defined: a view volume in the 3D world, a projection onto the projection plane and, a viewport on the projection plane. Geometry of objects in the 3D world are clipped against the 3D view volume and are then projected. The projected geometry are then transformed into the 2D viewport for display.

Since we are interested in mimeting the way human eyes perceive the 3D world, we will use a perspective projection where the center of projection is coincident with the viewer position o camera. The projection plane is such that its normal is given by the direction vector of the line through the viewing point, the view direction, and that goes through the reference point. The projection plane is also called the view plane. The view plane may be anywhere with respect to the 3D world objects to be projected. Then, a window on the view plane is defined such that its contents will be mapped into the viewport. Figure 5 illsutrates these concepts.

In this conditions, the view volume is a rectangular pyramid, whose apex lies at the camera position and the edges passing through the corners of the window. The four lateral faces of the pyramid define the lateral clipping planes. The view volume is limited along the view direction by means of the front culling plane and the back culling plane, which are parallel to the view plane and that are located at the *front culling* distance and *back culling* distance

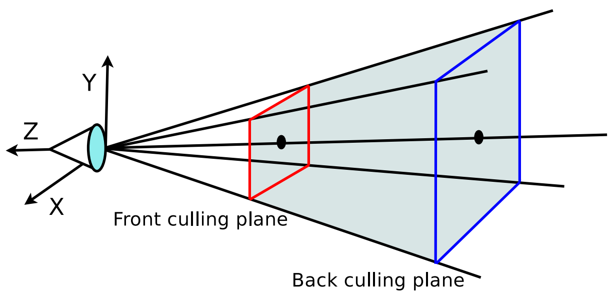


Figure 6: The view volume shown in gray.

relative to the camera measured along the view direction, [FvFH90]. Figure 6 shows an example of view volume.

Among the mathematical elements of perspective projections we are interested in recalling how a 3D point is actually projected. In this work, we assume that the projection plane is at a distance d from the camera position. This distance is also known as the focal length of the camera. Then, it is easy to see that if $P(x, y, z)$ is a 3D point, its perspective projection onto the projection plane P_p (see Figure 5) is given by equations, [FvFH90],

$$P_{px} = P_x \frac{d}{P_z}; \qquad P_{py} = P_y \frac{d}{P_z} \tag{1}$$

Sometimes these equations are written as

$$P_{px} = \frac{P_x}{P_z/d}; \qquad P_{py} = \frac{P_y}{P_z/d} \tag{2}$$

Notice that the distance d is just a scale factor for coordinates P_x and P_y. Furthermore, division by coordinate P_z causes the perspective projection of more distant objects to be smaller that those which are closer to the camera.

6 The Hybrid Approach

In this section we describe our approach to develop wireless remote streaming and rendering of terrains on mobile phones making use of low bandwidth networks available in the market, such as UMTS and GPRS. Preliminary versions of this work ca be found in [SRNF08] and [NSOJA09].

Our approach is basically a hybrid rendering technique where tasks are split between a remote server, generally featuring high-end hardware and software resources, and a mobile client, usually with very limited resources. Figure 7 shows the general architecture.

The main tasks in charge of the server are: 1) to store the terrain, 2) to supply the server with small chunks of terrain close to the user's position to be displayed and 3) rendering and sending to the client impostors for the terrain that is far from the users current position.

The main client tasks are 1) Rendering the terrain close to the user's position according to the required level of detail, 2) Displaying impostors that replaces the actual terrain in the

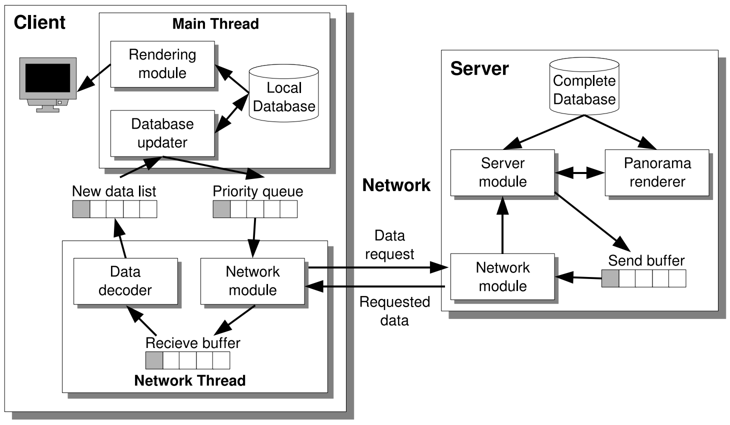


Figure 7: General hybrid architecture.

background and 3) requesting from the server updates for both the terrain and the panorama to be displayed as the users changes its position. The client dynamically adjusts its requests for updating terrain and panoramas according to its own capacity and the network congestion.

Rendering large-size terrain in real time on available mobile devices is still an huge complex task. However, our approach applies to a wide variety of server and client devices including from powerful desktop computers with high-end GPUs to low-end mobile devices laking of graphics hardware at all. To summarize, our approach offers the following advantages:

- The terrain area to be rendered by the client can be small without reducing the depth of view.
- Any terrain rendering technique can be used by the server to generate the impostors, including those based on GPU.
- Since screen resolution of mobile devices is small, usually in ranges like 320×240 and 640×480, impostors do not need to be generated at high resolution, thus saving bandwidth.

6.1 Terrain Representation

Adaptively streaming and viewing large-scale terrains on a mobile device require using specifically adapted algorithms and data structures. Recent trends show increasingly elaborated algorithms for large-scale terrain visualization and streaming. Since available CPU and memory resources in mobile devices are limited, we have designed our algorithms and data structures aiming at simplicity, efficiency and scalability.

Terrain data is obtained from a *height map* also know as *height field* or *digital elevation map* (DEM). A height map is an array of values which give the terrain elevations that are distributed for ground positions sampled at regularly spaced horizontal intervals. See Figure 8.

We organize the terrain representation according to two different levels. The first level subdivides the terrain in a regular grid of equal size tiles, each tile covering a squared heigthmap

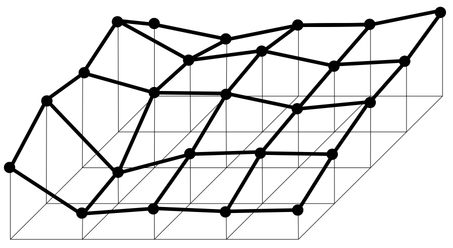


Figure 8: Height map.

including $(2^n + 1) \times (2^n + 1)$ height values where n is an integer greater that zero. Neighbor tiles share points in the common borders.

The second level consists of a set of quadtrees, [Sam89], each quadtree associated with one terrain tile. The quadtree root stores two sets of information. The first set stores the index that define how triangle strips will be built. The second set contains the terrain representation with the coarsest level of detail, $l = 0$, including $(2^{n_0} + 1) \times (2^{n_0} + 1)$ height values whith $n_0 \leq n$. The terrain in the root node is recursively subdivided into four equal squared size nodes defined by the two perpendicular node bisectors. Each node resulting from the subdivision adds a set of terrain points that were missing in previous levels. The specific new points are those placed on the perpendicular bisectors of the grid cells in the split node, their number is constant and depend on the inicial value of n_0. Figure 9 illustrates the idea for $n_0 = 2$. Notice that, after the subdivision, the layout of terrain points in the grid is preserved and the splitting law can be applied in subsequent subdivisions. The recursive subdivision stops when the level of detail reaches the highest precision when the quadtree nodes include the full set of points in the given terrain.

For each level of detail $l > 0$, the terrain is rendered as the set of points stored in the subtree whose root is the quadtree root and whose leaves are the set of nodes at tree depth l. Figure 10 shows the quadtree nodes for a terrain example for $n = 3$, $n_0 = 1$ and $l \in [0..2]$.

Textures associated with the terrain are also structured according to a quadtree defined as before.

Our data organization offers the following advantatges.

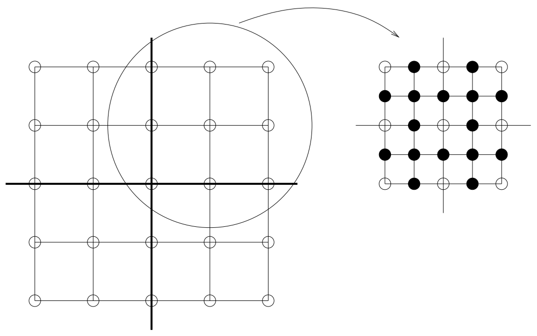


Figure 9: Terrain subdivision. Terrain points in a quadtree node at level of detail l and terrain points included in nodes at level of detail $l + 1$.

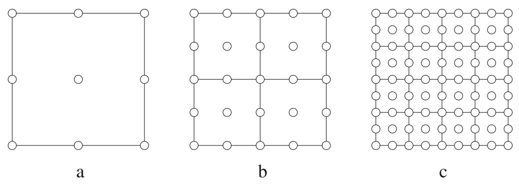


Figure 10: Sets of terrain points rendered. a) At level of detail $l = 0$. b) At level of detail $l = 1$. c) At level of detail $l = 2$.

- Works in [CGG$^+$03, LMG09] and [DSW09] advocate of using a single quadtree. However, our approach splits the terrain into tiles and associates a different quadtree with each tile yielding trees with a smaller number depth.
- The structure is suitable for both rendering and transmission.
- Quadtrees naturally are multi-resolution structures such that allow to dynamically choose the level of detail that best fits to render the terrain according to the needs of the application.
- The CPU resources need to manage the data structure are low. Moreover, no programmable GPU is requiered.
- Each node can be rendered on its own. Quadtrees can be streamed progressively, starting from the root nodes. Once the root node has been received, the full area covered by the quadtree can be rendered at it lowest level of detail. The quality is then increased as successively levels of detail are received.
- Height maps are typically provided at different resolutions. Since our structure holds a grid of independent quadtrees, the depth of each quadtree can be adjusted according to the resolution available at the area covered by the quadtree.
- The terrain can be rendered using only batched triangle strips, allowing for faster rendering.

6.1.1 Server Database

The server database consists of two components. One is the terrain height map and the associated texture images which are stored in the server's hard disk using the data structure described in the previous section. Besades we built a look-up table indexed by the quadtree identifier to easily determine the name of the file which stores a given quadtree.

With each quadtree node we associate a key that uniquely identifies it in the data structure. The key has two parts. The first part is an integer that enumerates the quadtrees in the data structure. The second part identifies a node in a given quadtree. Each node is identified by a string of characters defined following the Morton coding, [Mor66]. Basically, it consists of a sequence of digits each in {0, 1, 2, 3}), that respectively denote the parent node quadrant where the child node is located. The code length represents the node depth in the quadtree. Figures 11a and 11b illustrate the codes in a three-level quadtree.

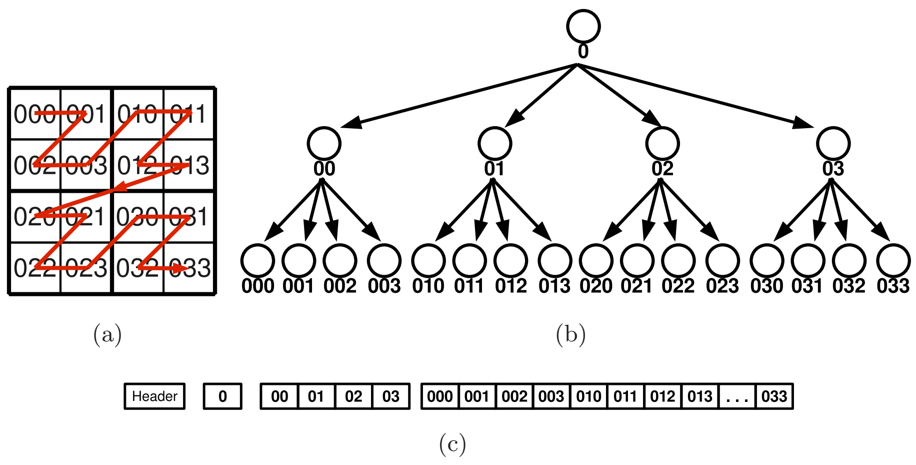


Figure 11: a) Morton codes for the third level of a quadtree. b) Morton codes for each quadtree level. c) File organization to store the quadtree.

From the server's point of view, the dataset is just a database with an unique key pair for indexing a block of bytes which contain the height values and the texture image of a node. This means that it is possible to use any standard database manager. Nevertheless, we have implemented our own storage manager, which is optimized for our data structure.

In a preprocessing step the grid of quadtrees to store terrain and textures is built. For each terrain tile a complete and balanced quadtree is built. Then, each quadtree is serialized into a different file. The files begin with a header which contains the quadtree identifier, the size of the height map covered by the quadtree, the number of height values stored in each node, the UTM coordinates of the bottom-left corner terrain tile and the distance (in meters) between adjacent height values.

Our approach optimizes the geometry data layout to improve memory coherency and input/output performance, [LP02, CGG$^+$03, LMG09], To minimize the number of disk accesses, data in files are stored according to the tree traversal order. Records in the file are sorted according to quadtree levels. The first record corresponding to the quadtree root. For each level, nodes in the quadtree are sorted according to their associated Morton code. See Figure 11b and Figure 11c. Notice that this organization allows to figure out the position of a node in the file in constant time. Height values are discretized and stored as 16-bit integers.

6.1.2 Client Database

The client database, like its server's counterpart, stores a set of quadtrees each corresponding to a terrain tile. The grid in the client stores a small subset of the tiles available in the server's database such that define a squared terrain area centered at the viewer position. This way we guarantee that views along any line of sight can be rendered. Depending on the data requested by the client, quadtrees here can be unbalanced and incomplete. Since access to the terrain data must be fast, the data is loaded into the main memory of the client. Due to the small amount of memory available in mobile devices, client's database is adaptively updated as explained in Section 7.1.

Each node in the client's database quadtrees stores two sets of information. One set connects the terrain in the node to the terrain model. The other is the terrain geometry. Specifically, the quadtree root node stores the quadtree identifier, the absolute world coordinates of the bottom-left corner of the terrain tile covered by the quadtree, the length of the squared terrain tile associated with the quadtree and a boolean which flags whether the node is being rendered or not. To reduce loss of precision, tiles sides and offsets are stored as floating point values. The terrain geometry is stored as a colection of coordinates of three dimensional points which are well suited for fast rendering. Coordinates are referred to the bottom-left node corner and coded as 16 bits short integers to allow integer arithmetic that reduces the amount of resources required in mobile platforms. Besades, the root node stores the terrain texture that will be shared by the quadtree nodes.

Quadtree nodes other than the root store the node identifier, the length of the squared tile side as well as the offset of its bottom-left corner with respect to the reference in the quadtree root and, the boolean flag. Points coordinates are stored as local coordinates referred to its associated tile bottom-left corner. The number of points in each quadtree node is kept constant for a given quadtree.

This intuitive and easy to implement structure allows us to render large-size terrains in an easy, fast and seamless way. This method is also fully compatible with the streaming nature of our application.

6.1.3 Integer Arithmetic and Precision

Most terrain rendering methods use floating points vertices. However, current commodity mobile devices do not feature floating point processing unit. Therefore, in our approach, points coordinates are represented as 16 bits short integers and floating point operations are performed as multiple integer functions. Besades improving integer arithmetic performance, vertex buffers size are clearly reduced when compared to using float values with 4 bytes o 8 bytes. Since memory is a valuable and scarce resource on mobile devices, and wireless links are usually slow, a reduction of the data structures size is an interesting improvement.

The main drawback of this approach is that points coded in absolute world coordinates are just too big to be represented by short integers. As explained in Section 6.1.2, we avoid this problem by segmenting the terrain into a grid of smaller chunks of terrain and then storing them as offsets from their respective origins.

Another precision problem concerns the terrain height values. As short integer values range from 0 to 65536, the actual height values given in meters should be scaled to cover the whole range. Otherwise, visible stair-stepping artifacts might appear.

6.2 Panoramas

In general, an impostor is a two-dimensional image that is used instead of a true three dimensional model to improve the rendering performance. Next we discuss how panoramas are defined and computed in our approach.

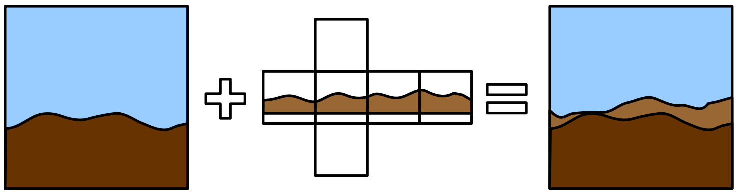


Figure 12: Synthesis of the nearby terrain rendered by the client and the panorama rendered by the server.

6.2.1 General Concepts

In our approach, impostors consist of two-dimensional synthetic images that simulate a wide view of a the physical terrain placed in the background far from the viewer. These impostors are called *panoramas*, [BJP08]. A panorama captures a visible view from a point in space in all directions. To visualize a panorama, it is first projected on a three dimensional shape to give the user the illusion of a 360 degrees view. The most usual three dimensional shapes used are spheres, cylinders and cubes. In the case of cubic shapes, panoramas cover viewing directions in 360 degrees by projecting the image on the inner six faces of a cube. Our method makes use of these kind of panoramas.

Panorams projected on a cube are usually referred to as *skybox* [Sha01]. Skyboxes are widely used to increase visual beauty of three-dimensional scenes because they provide an economical and effective way to render distant scenery. A viewer placed at the exact middle of the cube will perceive a three dimensional world around him. As the viewer moves across the scene, the cube remains stationary with respect to him. Since objects in the scene move while the skybox does not this technique gives the illusion that objects in the skybox are infinitely distant from the viewer, Notice that this behavior simulates real life, where distant objects such as clouds or mountains appear to be stationary when the viewer moves within relatively small areas.

The construction of a skybox panorama is straightforward [BJP08]. Each face of the cube covers 90 degrees of view both horizontally and vertically. The panorama is built by the server by placing its camera in the viewer coordinates in the client and making use of the terrain nearby. Then 6 orthogonal images are rendered, compressed using any standard image compression algorithm, such as JPEG [Wal91], and they are sent to the client, which mappes them onto each of the six faces of the cube.

The resulting image is composed by the client by merging the terrain and the skybox panorama as illustrated in Figure 12. Figure 13 shows two examples of terrains rendered with and without panorama.

6.2.2 Defining the Panorama

In our approach, the server renders the terrain in the background as a panorama and the client renders the terrain close to the viewer at a given level of detail. However, terrain shown to the user in the screen must be seamless rendered. Therefore, we split the terrain into nearby terrain and panorama as follows. Let the view volume in the client be limited by the

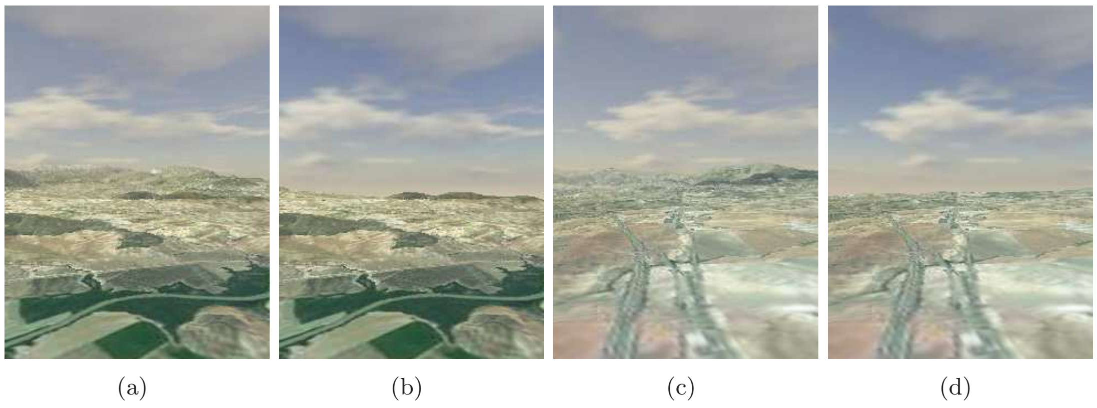


Figure 13: Scenes rendered in a Nokia N95. Left) Pictures with panorama. Right) Pictures without panorama.

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 Figure 14. The focal length of the camera d is the same for both server and client.

Clearly, the client renders terrain close to the viewer whereas the distant terrain is culled. On the contrary, the server culls the nearby part of the scene, and only the distant part is taken into account to render the panorama.

7 Updating the Scene

In an interactive navigation system, updating and displaying terrain data at the proper ratio is paramount. Our goal is to stream data over limited bandwidth communication networks where one of the ends is a mobile device with limited resources. Thus, specific techniques to minimize data traffic through the network must be devised. In what follows we describe the techniques we have developed to update the terrain and the panorama.

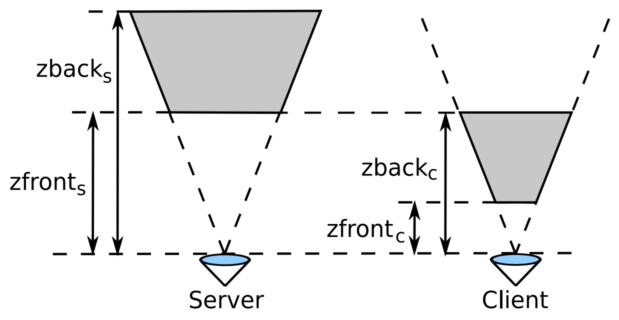


Figure 14: Splitting the view volume as terrain to be rendered and panorama.

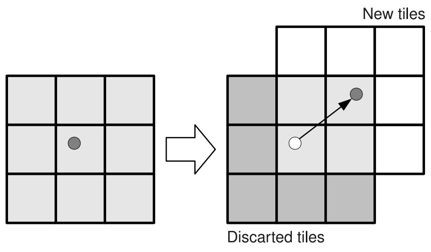


Figure 15: A square area of active tiles is maintained when the viewer moves.

7.1 Updating the Terrain

The server sends data to the client on demand. If a client needs a finest level of detail for a particular node which is not stored in its local database, the client streams it from the server. The existing data is used for rendering until the new level of detail is actually received.

When the users starts the walkthrough, the client initializes the local grid by downloading the root node of the quadtree corresponding to the tile where the viewer is positioned. This data allows the client to render the scene at a low resolution. As the user moves across the virtual environment, the local database is updated by maintaining the squared terrain area centered at the current viewer position which guarantees that views along any line of sight can be rendered. Newer data is requested to the server and nodes labeled as useless are removed.

Updating the local data base consists of two steps. First the set of tiles in the grid is updated. Then the set of quadtrees covering the current tiles are downloaded from the server. Since the grid for the whole terrain does not fit in the client's main memory, to select the set of tiles stored in the client we apply a windowing scheme, [Paj98]. The scheme works like a classical paging system for terrains and aims at maintaining an active square area centered at the viewpoint. If the viewpoint moves to a new tile (not necessarily adjacent), new tiles are fetch from the server and tiles outside the new squared area are removed. See Figure 15.

In the second step of the update operation, each quadtree covering the local grid is top-down traversed and a set of active nodes are selected. Listing 1 illustrates the process.

To select active nodes, many criteria can be used. Nevertheless, since our goal is to minimize the local CPU load, we use a simple measure based on the observation that, in general, the resolution needed to render terrain points drecreases as the distance from terrain to the viewer increases. Let e be the edge length of the tile covered by the quadtree node, let d be the distance from the current viewpoint position to the tile center and let C be a configurable terrain quality parameter. We define the importance for a node, f, by Equation 3, [SRH$^+$98]

$$f = \frac{d}{e \cdot C} \tag{3}$$

This criterion guarantees that the difference between the level of detail of two quadtree nodes whose tiles are adjacent is less than or equal to one [SRH$^+$98, LKES09]. If $f < 1$ and a fixed maximum quadtree depth, say *max_depth*, is not reached, the quadtree traverse continues. If children nodes are not available, they are streamed from the server. The streaming is performed in parallel to rendering, as explained in Section 8.1. If children are available, the current quadtree node is labeled as active and will be rendered. Their children are removed

```
procedure update( node $n$ )
  $\varepsilon \leftarrow$ compute_visual_importance ( $n$ )
  if $\varepsilon < 1$ or depth($n$) = $max\_depth$ then
    set $n$ as active
    if $\varepsilon < \sigma$ and not isLeaf( $n$ ) then
      delete children of $n$
    end if
  else
    if isLeaf( $n$ ) then
      set $n$ as active
      request children of $n$
    else
      for each $child_i$ of $n$
        update( $child_i$ )
      end for
    end if
  end if
end procedure
```

Listing 1: Progressive loading algorithm for a quadtree with caching

because they are no longer used for rendering. If $f > 1$ or the depth tree *max_depth* is reached, the terain in the current quadtree node is rendered and the recursivity halts.

To minimize reloading quadtree nodes that have been just removed, we apply a caching technique to keep in memory some of the local quadtree nodes that fall outside the squared area of current active nodes. Our caching techique is based on defining a threshold value, σ, such that nodes outside the current active area for which $f < \sigma < 1$ are actually cached for a possibly future use.

Un criterio de actualizacin ”perezoso” que s creo justificable es el siguiente: Si el observador no vara ni su posicin ni su lnea de visin, y la ltima operacin de actualizacin determin que la estructura de datos estaba actualizada, entonces no es necesario volver a actualizarla hasta que: a) el observador vare; o b) la estructura se modifique (porque se aada nueva informacin proveniente del servidor).

The scene is updated anly whenever the position or the line of sight of the viewer changes or when the contents of the scene undergoes any change. This approach reduces the data management costs significantly without loss of display quality.

7.2 Updating the Panorama

According to projection equations given in Section 5 as distance from the camera to terrain points increases, projected points became less significant. Moreover, changes in projections of distant points for small moves of the viewer are almost unnoticiable. Therefore, sending a new panorama to the client for every user move would result in a useless network traffic. Besades, updating the panorama just when changes in the background are noticiable, reduces the computing and memory requirements in the client side. Our updating strategy is based on assessing the error incurred in the rendered scene when the viewer moves and the panorama is not updated.

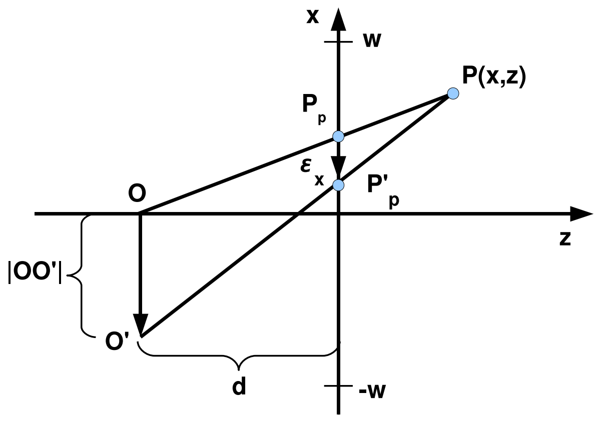


Figure 16: Translation along the X-axis.

Any arbitrary movement can be expressed as combination of translations and rotations. Since panoramas cover 360 degrees around the viewer, changing the view direction when the viewer rotates will not affect the scene displayed. However, when the viewer changes its position, the scene also changes. A general translation can be defined as the linear combination of three orthogonal translations along the framework axis. Thus, we consider two different cases. In one case the viewer moves along the X and Y axis. In the other case the viewer moves along the view direction. Let us detail each case. In any case, the distance from the viewer position to the projection plane will be kept constant.

7.2.1 Translation Along the X and Y Axis

Let V_s denote the set of points within the view volume in the server. Let V_c denote the set of points within the view volume in the client. See Figure 14. Let the viewer be placed at point O and let $P \in V_s$ be a point of the terrain in the view volume. When the panorama is rendered, the server projects point P onto point P_p on the projection plane. See Figure 16. Assume now that the viewer moves from O to O' along either the X-axis or Y-axis. Point P should now be projected onto P'_p. Therefore, if the panorama is not updated, the error in the projected point measured in pixels is

$$\varepsilon_x = P'_{px} - P_{px}; \qquad \varepsilon_y = P'_{py} - P_{py} \tag{4}$$

As expected, the larger the viewer translation, the greater the error. Consequently, as the viewer moves farther form the initial position, discontinuities between the terrain rendered in real time and panorama become aparent.

Having into account the perspective projection defined in Equation 1, we have:

$$P_{px} = P_x \frac{d}{P_z}; \qquad P'_{px} = (P_x + |OO'|)\frac{d}{P_z} \tag{5}$$

where $|OO'|$ is the distance in the X-axis from O to O' and d is the focal length of the camera. Therefore, the error in the projected points can be expressed as

$$\varepsilon_x \ = \ P_x \frac{d}{P_z} - (P_x + |OO'|)\frac{d}{P_z}$$

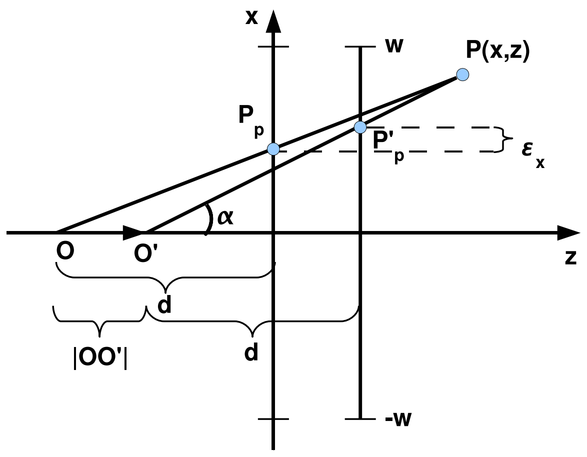


Figure 17: Translation along the Z-axis.

$$= \frac{d}{P_z}(P_x - P_x - |OO'|)$$
$$= -\frac{d}{P_z}|OO'|$$

Since coordinate P_z divides, the error decreases as distance from terrain points to the projection plane increases. Ignoring the sing, the viewer translation along the X-axis corresponding to a given error threshold, ε_x, is

$$|OO'| = \varepsilon_x \frac{P_z}{d}$$

The maximum error allowed for points $P(x, y, z)$ in V_c is achieved for those points on the back culling plane, $zback_c$, that is

$$|OO'| =\leq \varepsilon_x \frac{zback_c}{d}$$

If $P(x, y, z)$ is a terrain point in V_s we have that $P_z \geq zfront_s$. But, according to our definition of panorama given in Section 6.2.2, $zback_c = zfront_s$. Hence, to preserve continuity in the terrain-panorama transition, we will update the panorama whenever the viewer translation fulfils

$$|OO'| > \varepsilon_x \frac{zfront_s}{d} \tag{6}$$

Similarly, if the window in the projection plane is not squared, we will update the panorama whenever

$$|OO'| > \varepsilon_y \frac{zfront_s}{d} \tag{7}$$

7.2.2 Translation Along the View Direction

Consider now translating the viewer along the view direction, that is, along the Z-axis, as illustrated in Figure 17. Recall that, in our approach, the focal distance d must be kept constant.

Let the viewer be placed at point O. Let $P(x, y, z)$ be a point in V_s whose projection onto the projecting plane is P_p, If the viewer moves to point O', point $P(x, y, z)$ is projected onto P'_p. Proceeding in the same way as in the translation along the X and Y-axis, the components of projected points are now

$$P_{px} = P_x \frac{d}{P_z}; \qquad P'_{px} = P_x \frac{d}{P_z - |OO'|} \tag{8}$$

and

$$P_{py} = P_y \frac{d}{P_z}; \qquad P'_{py} = P_y \frac{d}{P_z - |OO'|} \tag{9}$$

To keep distance d constant, the projection plane must move with the viewer. Notice that when the viewer moves along the X-axis or the Y-axis, projected points only move on the projection plane along the corresponding axis. Now, when the viewer moves along the Z-axis, projected points in general move on the projection plane along both X- and Y-axis. In what follows, we only figure out the error in the projected points for the component on the X-axis. The same rational applies to the error on the Y-axis.

The error defined by Equation 4 is now is

$$\varepsilon_x = P_{px} - P'_{px} \tag{10}$$

Combining Equations 8 and 10, we have

$$\varepsilon_x = P_x \frac{d}{P_z - |OO'|} - P_x \frac{d}{P_z}$$

Dividing by $P_x d$, results

$$\begin{aligned}
\frac{\varepsilon_x}{P_x d} &= \frac{1}{P_z - |OO'|} - \frac{1}{P_z} \\
\frac{1}{P_z - |OO'|} &= \frac{\varepsilon_x}{P_x d} + \frac{1}{P_z} \\
\frac{1}{P_z - |OO'|} &= \frac{P_z \varepsilon_x + P_x d}{P_x P_z d} \\
P_z - |OO'| &= \frac{P_x P_z d}{P_z \varepsilon_x + P_x d} \\
|OO'| &= P_z - \frac{P_x P_z d}{P_z \varepsilon_x + P_x d}
\end{aligned}$$

Finally, given an error threshold, ε_x, the allowed translation along the Z-axis is

$$|OO'| = P_z \left(1 - \frac{d}{\frac{P_z}{P_x} \varepsilon_x + d} \right) \tag{11}$$

The allowed translation given in Equation 11 can be expressed as a function of the parameters that define the viewing volume. Let $P(x, y, z)$ be an arbitrary point in V_s. If w denotes the viewing window half-width, see Figure 18, we have

$$\tan(\alpha) = \frac{P_x}{P_z} < \frac{w}{d}$$

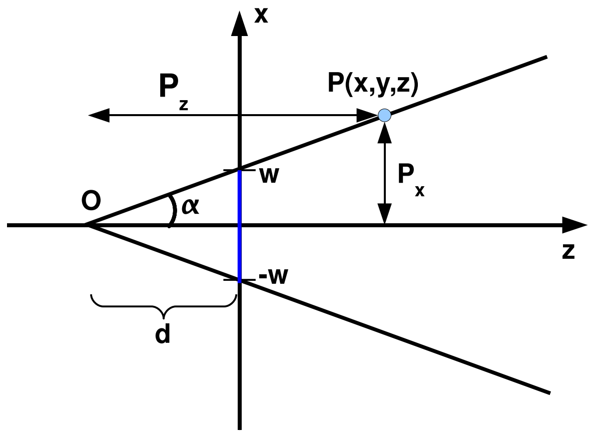


Figure 18: Points P located on the lateral culling plane define the angle spanned by the viewer placed at point O.

Therefore

$$\frac{P_z}{P_x} > \frac{d}{w} \tag{12}$$

Replacing Equation 12 in Equation 11, results

$$|OO'| \geq P_z \left(1 - \frac{d}{\frac{d}{w}\varepsilon_x + d}\right) = P_z \left(1 - \frac{w}{\varepsilon_x + w}\right)$$

According to the definition of panorma given in Section 6.2.2, terrain points in the panorama fulfil $P_z \geq zfar_c$, (see Figure 14). Consequently, the client requests to update the panorama whenever the following condition holds

$$|OO'| \geq zfar_c \left(1 - \frac{w}{\varepsilon_x + w}\right) \tag{13}$$

7.2.3 Algorithm to Update the Panorama

To summarize, our method to update the panorama has the following steps:

1. The server renders the distant terrain into a panorama and sends it to the client. The client renders the nearby terrain and uses the panorama as an impostor of the distant terrain. Parameters that both client and server use to render the terrain are the viewers position, O, the focal distance of the camera, d and the half-width of the viewport in the viewing plane, w.
2. The viewer moves from O to O'.
3. The client renders again the nearby terrain according to the new parameters O', d and w.
4. The client decides whether to request a new panorama to the server according to whether any of the criteron given by Equations 6, 7, or 13 is met.

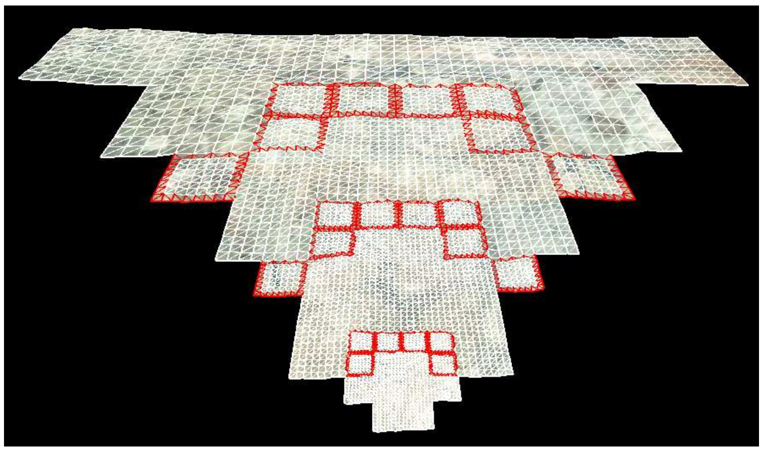


Figure 19: View volume culling. Stitching strips are in red.

7.3 Visualization

After updating the terrain, the quadtree nodes that must be rendered are those labeled as active. They store the current terrain approximation in a format suitable for graphics rendering. Each node stores the 3D coordinates of the sampled terrain points.

The rendering is performed as follows. For each frame, quadtrees in the client grid are top-down traversed and nodes are culled according to the view volume as depicted in Figure 19. Points in a quadtree nodes are linked in triangle strips according to standard strip masks, [PM05]. Since the number of terrain points in nodes of a given quadtree is constant, strip masks can be computed in a preprocessing step. As stated in Section 7.1, level of detail of adjacent nodes can differ at most in one unit. Therefore, the number of different stitching strips required to avoid cracks is small. The stitching strips used in our approach are shown in Figure 20b and Figure 20c.

To determine the strips required to render a node, we first identify the Morton codes of the four neighbour nodes. They are computed from the current node Morton code. See Section 6.1.1. This is done through simple shift operations.If the level of detail of the four adjacent nodes is equal to or smaller than that of the node at hand, it is rendered as a tessellated quadrangular

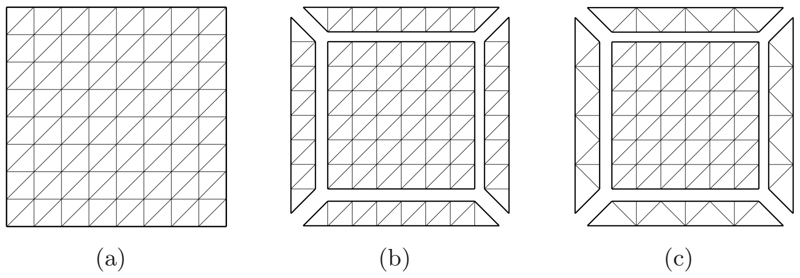


Figure 20: a) Triangle strips in a 9×9 node. b) Stitching strips for neighbors with the same level of detail. c) Stitching strips for neighbors with different level of detail.

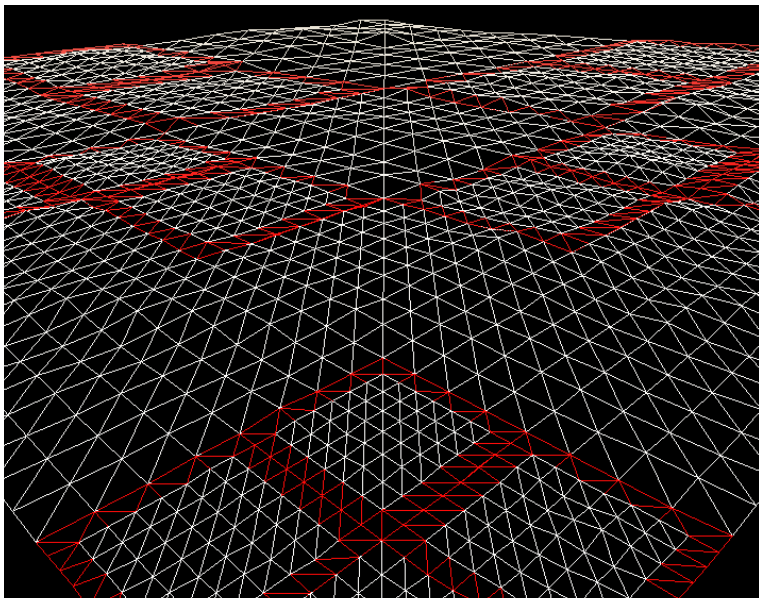


Figure 21: Surface triangulation. Quadrangular meshes are in white, stitching strips are in red.

region using a unique triangle strip. See Figure 20a.

When there is a neighbor whose level of detail is larger than the level of detail of the node at hand, stitching strips must be used to matching different levels of detail and avoid cracks. Then terrain points inside the node are rendered as a tessellated quadrangular region using a unique triangle strip. Figure 20b and Figure 20c illustrate these ideas. Examples of scenes rendered applying this technique are shown in Figures 21 and 22.

Viewer motion is usually smooth, so we can exploit spatial and temporal coherence by reusing the same data in several frames. If terrain points in quadtree nodes are stores as vertex buffer objects, they can be sent to the client GPU, cached in local GPU memory and reused without moving them again from main memory to the GPU. Whenever vertex buffer objects are unavailable, standard vertex arrays can be used to transfer the data from the CPU to the GPU.

Since terrain points are stored in local coordinates, relative to the quadtree node they belong to, the triangulated mesh of the node is translated and scaled to properly place it with respect to the grid covered by the quadtree. The same transformation is applied to the triangulation of the terrain covered by the quadtree to place it with respect to the global coordinates.

If a given node has to be rendered with texture, it is propagated from the texture in the tree root by applying the translation and scaling need to fit the texture in the current node.

8 Other Issues

In this section we consider three aspects that play an important role in our approach. First we describe the multi-threading architecture to manage client-server data traffic. Then our spe-

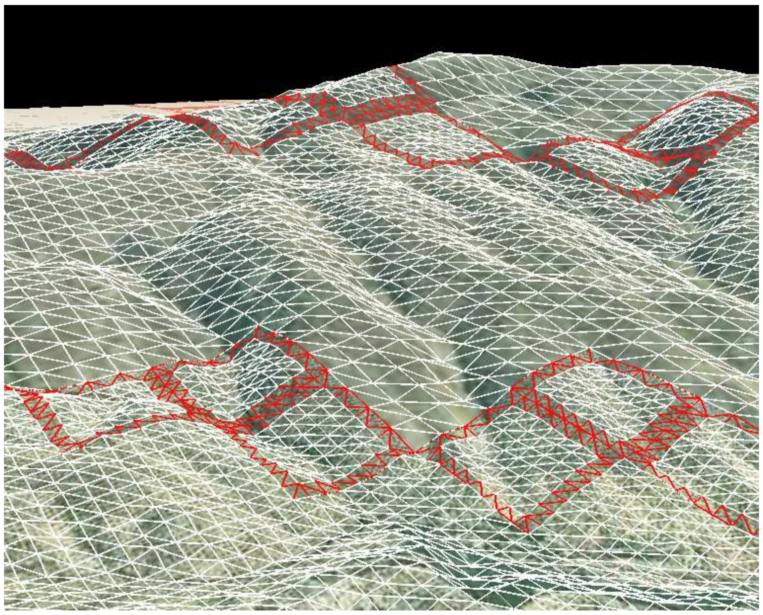


Figure 22: A terrain view with a wire-frame on top of it. Quadrangular meshes are in white, stitching strips are in red.

cific end-to-end network protocol is presented. Finally we offer some consideration concerning the scalability of our approach.

8.1 Multi-threading Architecture

To manage the data traffic between the client and the server we have developed thea multi-threading architecture shown in Figure 23.

8.1.1 The Server

The aim is to define an architecture that allow an variable number of clients to be connected simultaneously to the server. The higher the number of clients that can be served, the better

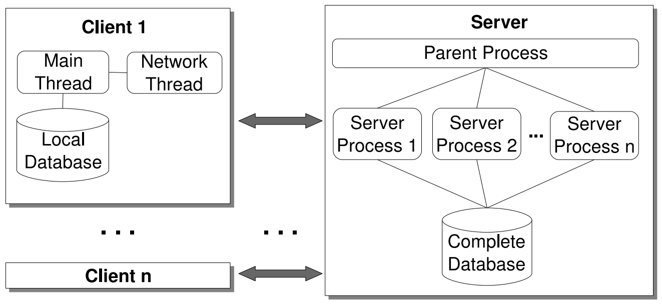


Figure 23: Multi-threading client-server architecture

the system scalability.

For the server, we propose a multi-process architecture as shown in Figure 23. The data flow is managed by a master process that to a network socket, waiting for incoming clients. When a client connects to the server, a new process is created with an associated socket and an established connection with the client. The child process lives until the connection is closed or the parent process dies.

Processes in the server are independent from each other and do not share any kind of information apart from the read-only terrain database. It might be argued that the multi-process architecture does not allow two server processes to share in-memory structures, such as the quadtrees used for rendering a panorama. However, this is not a problem in our case. First notice that different clients might be navigating over different terrain zones far from each other. Moreover, different clients may move at different speeds and in arbitrary divergent directions. Therefore, sharing terrain data is of little interest.

Many 3D graphics libraries, such as OpenGL/OpenGL ES and Direct3D 10 [MB05], are not thread-safe, That is, in a multi-threading environment the graphics context is bound to a specific thread, making it impossible for multiple threads to access to the graphics API. In spite of that, this problem does not exist in a multi-process environment, so server processes can render panoramas safely and independently from each other. If more than a GPU are available in the system, they can be used concurrently by different processes.

8.1.2 The Client

The client also follows a multi-threaded paradigm, as depicted in Figures 7 and 23. The main thread manages the user interface, renders the scene, updates the local database and, process the new quadtree nodes and panoramas provided by the network thread. To reduce CPU work in the main thread, networking tasks are moved to a second thread which manages the communication with the server and decodes quadtree nodes and textures.

These two threads communicate through a shared priority queue (labeled as *Priority queue* in Figures 7. When the main thread updates the local database and determines that a locally unavailable level of detail or a new panorama is needed, it pushes the request in the queue. High priority requests are sent first to the server through the network thread.

Sine panoramas and quadtree nodes with the lowest depth carry the most relevant data, they are issued first. To prevent network congestion, when the number of sent requests awaiting for their response to arrive recahes a given threshold, the client stores new requests to the server in a waiting queue. The queue also allows to apply speculative prefetching to download new level of details before they are needed. This is performed by pushing into the queue new levels of detail which are likely to be required soon. These requests are set with the lowest priority, so they are only issued if no request with higher priority is present and no network congestion has ocurred.

Similarly, the network thread supplies the received data to the main thread. Since most graphics libraries are not thread-safe, the network thread is unable to update the local database. Consequently, this thread cannot send textures nor terrain points to the graphics context. To solve this issue, we have provided a second shared cache where the network thread stores all the received and decoded data, labeled as *New data list* in Figures 7. The main thread reads this cache and updates the local database accordingly.

8.2 Network protocol

Several middleware solutions which provide communication transparency and makes it easier to develop a network protocol have beeen developed. Examples are CORBA, SOAP and RMI. These solutions, however, are high resource-consuming and add a significant network overhead. Therefore not well suited for mobile devices.

Most terrain streaming protocols found in the literature either use a simple file transfer protocol, such as HTTP [PM05, BGMP07], or do not provide details at all. Since we need our server to be smart enough to be able to render on-demand panoramas, we have developed our own end-to-end protocol, able to work through heterogeneous networks and to transmit large amounts of binary data. It is a request-response protocol. When the client needs to download new data, it sends a request message to the server. Then, it waits for the response to arrive from the server.

Since the Transmission Control Protocol (TCP) is a reliable connection oriented protocol which provides control over flow and congestion, we have built our protocol on top of TCP.

As [FYPA05] points out, wireless links usually exhibit slow bandwidth and high latencies what affect TCP performance. TCP assumes that packet losses occur due to network congestion and, consequently, decreases its congestion window. However, packet losses may occur due to the high bit-error rate of the transmission medium or due to signal fading. Therefore, reducing the congestion window results in an unnecessarily reduction of network utilization on wireless networks.

Our network protocol addresses these drawbacks as follows. TCP uses the *slow start* congestion avoidance algorithm, [Pos81], which penalizes the speed of the first transmissions. To avoid this drawback, once a connection is established, our protocol keeps it as long as the application is running.

The communication between server and client cannot be synchronous (blocking) because the slow bandwidth and high latencies of the wireless links might cause our application to stall during long transmissions. Therefore, our protocol uses asynchronous (non blocking) networking. This, however, increases the risk of an overflow of the TCP socket buffers, specially on slow wireless links, [FYPA05].

In order to fully exploit the network bandwidth and to avoid buffer overflows, we provide and manage our own communication buffers, stored in user memory. Server responses are serialized into a *send buffer* (see Figure 7) then they are trasnmitted via TCP through an asynchronous process. When either several messages are available in the send buffer or a time-based trigger delay is over, they are sent together through the TCP socket.

This approach can result in a dramatic improvement in response times, because it allows to pack several messages into one TCP/IP packet, reducing the network overhead. Most messages in our network protocol are small. For example, if a quadtree node stores 9×9 16-bit integer height values, a message that sends a new node to the client takes 182 bytes (header included). The typical TCP Maximum Segment Size, [Pos83], for Ethernet is around 1500 bytes, so a single packet can pack multiple messages. The TCP header is 20 bytes long,

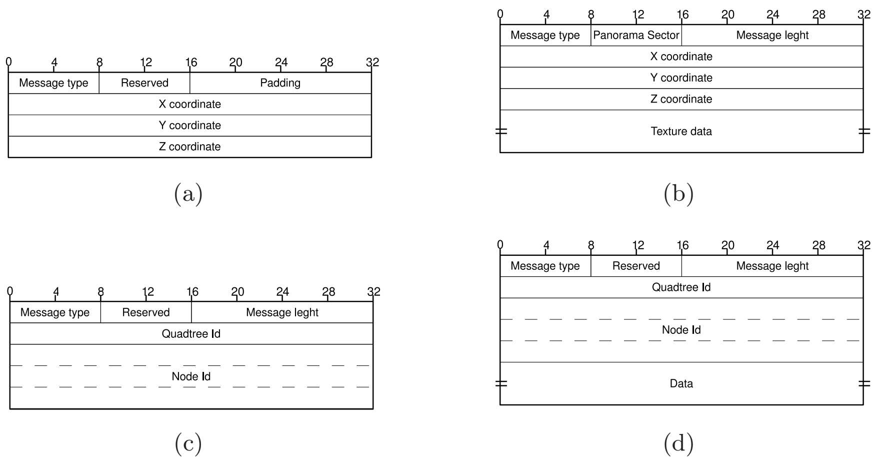


Figure 24: Messages used by our protocol. (a) Request for a new panorama. (b) The response (c) Request for a quadtree node (d) The response.

so the network overhead can be minimized by sending larger packets. Moreover, the burden on IP routers and networks is also reduced.

Incoming responses are also asynchronously received by the client and stored in a *recieve buffer*. See See Figure 7. The responses then await there to be processed. This buffer avoids the risk of a buffer overflow in the TCP socket.

Our protocol consists of two kind of messages: requests and responses. A request can query a new grid tile (actually, a root node of a quadtree), a non-root quadtree node or a panorama. The server then issues a response message, which provides the requested data to the client. When the client needs to split a quadtree node, it issues a single node request to the server. The server answers with another single message which packs the four children nodes together. These nodes are read together from secondary memory and issued to the client in a single message. This scheme minimizes the number of disk access as well as the number of network messages. See Section 6.1. However, since node textures size are large, they are sent independently.

Each message consists of a header and a posible payload. The header provides data such as the length of the message in bytes, quadtree and node identifiers, camera coordinates from where a panorama needs to be rendered, etc. The most important messages in our protocol are summarized in Figure 24.

8.3 System Scalability

Many streaming and rendering techniques have been proposed in the literature, [BGMP07, LMG09, Paj98]. However, they have been designed to run on homogeneous desktop computers with similar performance and the metrics they propose to choose the visualization quality or level of detail are usually prefixed.

Mobile devices are heterogeneous. On the lowest end we can find mobile devices which are equipped with reduced memory and limited low speed CPUs. Floating processing unit and any kind of specific graphics hardware are also missing. On the highest end, we find powerful smartphones featuring multi-core CPU, floating point processor and even programmable GPUs. Therefore one cannot expect that reported general techniques properly perform in this environment.

Our server-client approach develops a highly scalable architecture, capable of dynamically balancing the workload according to the specific capabilities of the server and client at hand. The main idea is that the client should receive terrain models that can be handled at interactive rates. Thus users with a low-end mobile can perform a fluid navigation across a coarse terrain, while users with a high-end mobile can enjoy a richer and more detailed environment. Notice that this approach naturally applies to desktop-based platforms.

Our hybrid model is highly parameterized. Terrain quality is controlled through parameters C and *max_depth*, defined in Section 7.1. The role played by the panorama and how often it is updated are controled respectively by volume view planes $zback_c$ and $zfront_s$, defined in Section 6.2.2, and by the maximum error allowed on the viewing projection, ε_x, ε_y defined in Section 7.2. By properly controling values of these parameters, the workload can be dynamically balanced. For example, according to Equations 7 and 13, server-based and client-based approaches are particular instances of our approach. If the planes $zfront_c$ and $zback_c$ are coincident, see Figure 14, the server is in charge of the whole rendering tasks, the panorama requires to be updated after any viewer move and the approach falls into the server-based category. If in the client plane $zback_c$ is moved away from the viewer with $zback_c = zback_s$, the client is in charge of the visualization and there is no need for updating the panorama. That is, the system behavies like a client-based system.

9 Results

To evaluate and validate our approach, we carried out an extensive set of experiments on a mobile device and a desktop PC using different wireless networks.

9.1 The Terrain

The terrain used was the Puget Sound dataset, a standard benchmark for terrain rendering applications. It is made of 16385×16385 samples at 10m spacing, 2 bytes height values with a vertical resolution of 0.1m. The texture map has a size of 16385×16385 pixels, with a resolution of 10m per pixel. The terrain data was first partitioned into a grid of 16×16 tiles each with 1025×1025 heights. Every tile was then organized as a eight-level quadtree. Each node of the quadtree stored an array of 9×9 height values. Nodes in the first five quadtree levels also stored a texture with 64×64 pixels.

The terrain was represented following two different levels of geometric complexity. The low quality level was characterized by a quality parameter $C = 3$ in Formula 3, and a quadtree depth of 4. The high quality level had a quality parameter $C = 9$ and a maximum quadtree depth of 8. Given the small size of the screen, low quality level offers good visual quality on mobile devices. See Figure 25. The image in Figure 25a was rendered using 5883 triangles while image in Figure 25b included 53039 triangles. Differences in the visual image quality

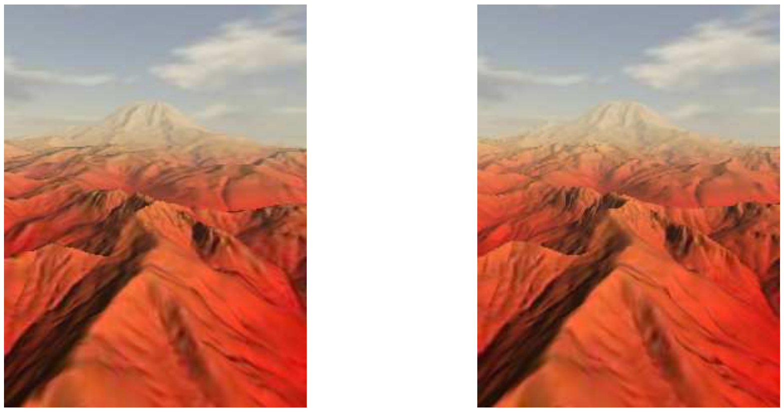

(a) (b)

Figure 25: Puget Sound rendered by a Nokia N95. a) 5883 triangles. b) 53039 triangles.

are hardly noticeable.

High quality level is more suitable for desktop PCs tahn for commodity mobiole devices. However, to experiment how mobile devices perform under high stress conditions, we have also run the high quality level tests on the mobile device.

9.2 The Platforms

According to the preliminary study detailed in Section 3, we selected the Nokia N95 mobile phone, powered by a 332 MHz OMAP 2420 CPU (ARM11-based), with 128 MB system memory and a PowerVR MBX GPU which supports OpenGL ES 1.1.

In addition, our implementation was also tested on a system including two desktop PC with a 2.40GHz Intel Core-2 Duo CPU, 4 GB system memory and a nVidia GeForce 8800 GTS GPU. The window size in the mobile phone was 240×320 pixels and on the desktop PC was 1024×768 pixels.

9.3 The Flyover

For each test, we performed a flyover following the diagonal of the terrain. To avoid fake results, the terrain boundaries were never reached.

The viewer always moves forward at constant speed and at a constant height of 100m over the terrain. To allow the data to be partially fetched, we introduced a 15 seconds delay before the viewer started the flyover.

To study the influence of the viewer speed on the system performance we run three sets of flyovers with navigation speeds of bicycle (25 km/h), car (150 km/h) and jet (750 km/h).

Notice that following a rectilinear trajectory at high speed is the worst possible scenario for our hybrid-rendering technique. The rational is as follows.

- Panoramas cover 360 degrees around the viewer and, in general turns do not change the distance to the terrain close to the panorama. Therefore in trajectories with turns a given panorama is valid for a longer period of time than in rectilinear trajectories.
- On a system where all the geometry-rendering task is performed by the client, a change in the viewer line of sight forces the client to download new terrain. In our hybrid-rendering technique, this does not always happen, as the panorama covers 360 degrees around the viewer. However, in our test the line sight of the viewer never changes, thus nulling this advantage of our technique.
- To avoid downloading and rendering terrain far from the viewer our approach uses panoramas. Nonetheless, in a rectliniear trajectory the viewer is always moving forward and the distant terrain is soon reached. This will require downloading the actual geometry of the terrain that is now closer to the viewer and to update the panorama.

9.4 The Network

Most streaming techniques found in literature use fast and wired network connections, such as USB 2.0 [PM05], ADSL [LMG09, BGMP07] or even local access to the dataset [PM05]. However, since we want to study how our technique performs in real-world wireless connections, we carried out our tests using two popular mobile telecommunications technologies: UMTS (3G) and GPRS (2G). The bandwidth of UMTS is wider that that of GPRS. However, UMTS coverage is usually limited to densely populated areas while GPRS is the only available network in sparsely populated areas. We did not account for the effects rsulting from the viewer roaming back and forth in neighboring cells.

9.5 The Tests

All tests have been run with and without panoramas. In both cases, the minimum viewing distance was 30Km. When using panoramas, they were placed 7.5 Km away from the viewer. the maximum allowed percentage of error was 5% and the resolution was 256×256 pixels per skybox face.

For each test we recorded a set of measures taken along the trajectory. Results are collected in the Appendix. Table 3 summarizes the averaged results for the Nokia N95 platform. The measures include the number of frames per second rendered, the average number of rendered triangles in each frame and, the total downloaded data measured in Kb in 300 seconds flyovers. Notice that values for 1000km/h and high quality geometry in the GPRS network are missing. The reason is that in these conditions, the mobile device collapsed. Table 4 summarizes the averaged results for a Desktop PC with nVidia GeForce 8800GTS.

The rendering frame rate depends mainly on the amount of rendered triangles. The low network bandwidth requirements of our approach makes that, in general, it manages to achieve an almost constant frame rate and number of triangles on most of the tests carried out. For instance, in a five minutes flyover at 150 km/h and using a UMTS network, only 2.69 Mb of data have been transferred in the high quality test. In the low quality test the data transferred amounts to 0.41 MegaBytes. In both cases the data included geometry, textures and panoramas.

			High Quality			Low Quality		
Network	Speed	Panor.	FPS	Trg.	Download	FPS	Trg.	Download
UMTS	Bicycle	Yes	11,53	42015	447	53,08	4993	158
		No	7,47	68999	707	38,43	8242	187
	Car	Yes	11,20	41659	2757	51,90	5139	429
		No	7,30	66507	3216	37,25	8587	393
	Jet 750	Yes	12,13	30827	6160	49,05	5041	1629
		No	8,83	49549	6489	36,36	8310	1251
	Jet 1000	Yes	13,09	27085	7845	49,16	4945	2298
		No	8,32	46111	8286	36,19	8285	1737
GPRS	Bicycle	Yes	18,79	35042	420	52,86	4771	149
		No	17,91	44047	629	38,83	7705	179
	Car	Yes	20,15	28563	1284	53,28	4579	373
		No	20,13	32607	1156	38,83	7867	359
	Jet 750	Yes	58,60	2500	1435	58,58	2826	1401
		No	40,39	8567	1393	45,62	5570	1096
	Jet 1000	Yes				59,47	1359	1193
		No				50,46	4636	1363

Table 3: Results yielded by the Nokia N95 device.

			High Quality			Low Quality		
Network	Speed	Panor.	FPS	Trg.	Download	FPS	Trg.	Download
UMTS	Bicycle	Yes	506,70	73230	609	2174,26	7425	154
		No	345,03	113253	925	1961,28	12081	211
	Car	Yes	496,85	73216	1985	2148,82	7548	489
		No	337,94	115924	2353	2083,25	12493	481
	Jet 750	Yes	501,54	70027	8492	1792,73	7420	1909
		No	351,34	110267	9117	1695,95	12289	1608
	Jet 1000	Yes	547,24	65869	11408	2245,69	7228	2715
		No	357,45	108496	12065	2052,61	12065	2254
GPRS	Bicycle	Yes	791,42	59318	585	2135,50	7019	154
		No	757,06	78168	854	2005,72	11086	211
	Car	Yes	1239,47	24569	1363	2194,90	7024	466
		No	939,03	47215	1459	2149,85	11083	455
	Jet 750	Yes	1918,49	2638	1428	1924,77	2200	1325
		No	1735,56	7312	1389	1699,74	5483	1339
	Jet 1000	Yes	2411,90	1754	1522	2542,48	1695	1550
		No	2241,06	6151	1516	2195,34	4601	1523

Table 4: Results yielded by a Desktop PC with nVidia GeForce 8800GTS.

The system performance when using an UMTS connection stays stable through all the tests. Plots in the Appendix show that, starting from scratch, the amount of triangles transferred quickly converges to a certain number which depends on the desired quality. Then, this number remains stable.

The initial loading time until achieving a stable number of triangles takes around 10-15 seconds. During this initial loading time, the downloading plots is very steep, which suggests that the available bandwidth is fully used. Once the initial loading is over, the downloading curve flattens on all tests that use the UMTS network. Since rendering is progressive, only a few nodes per frame need to be refined, and only those nodes not cached yet are requested to the server. The plots indicate that only a small fraction of the bandwidth provided by UMTS are required for a fluid and stable navigation, even when the user is moving at high speeds.

Our tests show that, as used in our approach, the GPRS connection provides a TCP maximum download rate of only 5.5 Kb/s. Since our tests last at most 5 minutes, no more than 1.5 Mb can be downloaded during the whole flyover. Plots in the Appendix prove that our approach performs quite well even when using such a limited network.

When moving at bicycle and car speeds, the number of triangles downloaded also converges to a certain value which depends on the desired quality. About 60-120 seconds are needed by the high quality tests, and 30-50 seconds by the low quality tests. The time needed by the GPRS networking is lobger that that needed by the UMTS.

In standard conditions, once the initial loading time has ended, the amount of triangles remains stable during the rest of the test when moving at bicycle or car speed and using a GPRS connection. Specifically, when moving at car speed (150 Km/h), the amount of triangles stays around 30000 (when panoramas are being used) and around 50000 (when panoramas are not present) on both the mobile device and the PC platform. Notice that these numbers widely exceeds the needs required by the small screen of a mobile phone.

Tests at jet speed are worse affected by lower avaible bandwith, as a larger amount of data needs to be transmitted to the client. The combination of a high latency and low bandwith network with a high speed motion of the viewer pushes our streaming architecture to the limit. When moving at high speed, the GPRS network is unable to cope with the requested data, causing the application to reduce the number of rendered triangles. Furthermore, most of the downloaded data are no longer needed because they get behind the viewer. The GPRS high delay and latency makes unvfeasible downloading higher quality nodes for visible areas. Nevertheless, our architecture still manages to render a scene of around 2000-5000 triangles which gives a good enough approximation of the terrain on a small screen and is an good result given the low performance of the GPRS network. Interestingly, the low quality test downloads a number of triangles larger that those downloaded in the high quality test.

Results in Table 3 and Table 4 show that using panoramas and high quality terrains might produce network congestion on GPRS when the viewer moves at high speed, which causes a loss of performance. This is due to the fact that, given the high latencies and the small network bandwidth of the GRSM connection, the time required to update a panorama is longer that the time the panorama remains valid. Therefore, subsequent requested panoramas can stall into the network buffers, and the system spends network bandwidth on downloading out-of-date panoramas that never will be used. This situation is easily avoided by adding a flag which allows to remove panorama requests when network congestion is detected. This way, the client does not issue new request of panoramas until the last request has been served. Figure 88 in Appendix shows a significant performance improvement when using this method. The error in the panorama is compensated by an increment in the number of triangles of the nearby terrain.

In the experiments with GPRS we have noticed that the network speed does occasionally slow down that do not completely shutdown the client-server connection but do severely degrade the transfer rate through the network. During time intervals that might last for up to 30 seconds, the transfer rate decreases to values of a few bytes per second. This phenomenon is clearly noticeable in Figure 43, corresponding to the tests on Nokia N95, bicycle speed, high quality and without panoramas and, in Figure48 for Nokia N95, bicycle speed, low quality, with panoramas. Both figures are included in The Annex. As the viewer moves along the flyover path, these network slowdowns cause a dcrease in the number of triangles transferred and thus, an increment of the frame rate. In this situation, the downloading plots become almost horizontal. Once the quality of the link is restored, data waiting in the cues are transferred and the downloading curve shows a sudden steep rise and, after a few seconds, an state similar to that shown before the slowdown is reached.

When using the UMTS connection we did not observe these problems. Therefore we conjecture that they could be attributed to factors beyond our systems' control, like signal fade outs due to bad meteorological conditions. Since these phenomena are to be expected, a streaming system which aims at working in real-life networks, beyond optimal laboratory conditions, should be able to handle unexpected problems such as disconnections and occasional degradation of the network service quality. When the networking suffers a cut, the client in our system keeps rendering the data cached in the local database. The results of our experiments, see the Appendix, show that our approach is able to deal with interruptions in the network service of up to 30s when moving at 150 km/h without noticing a severe decrease in the number of rendered triangles and, therefore without a severe decrease of quality in the rendered scene. The results plotted in the Appendix empirically prove the effectiveness of our hybrid-rendering technique. As can be seen in Table 3 and Table 4 using UMTS, our technique reduces the avergae number of triangles by a factor of 0.6 - 0.7 (30-40%). It consequently increases the avergae frame rate by a 1.5 factor (50%) on the high quality tests, and by a 1.4 (40%) on the low quality tests.

When using GPRS, the improvements of our hybrid-rendering technique shown in Table 3 and Table 4 seem to be not as good. However, it should be kept in mind that the averaged values include in the flyovers the initial downloading time. Since the number of triangles initially downloaded is small, the averaged values do not show the actual benefit of using panoramas. For instance, Figure 46a and Figure 47a depict the results when moving at car speed and using GPRS, with panoramas and without panoramas, respectively. The first one needs 80 seconds to converge, whereas the second one needs 155 seconds.

The use of panoramas also nas an effect on the bandwidth requirements. Table 3 and Table 4 show the total downloaded data during each test. For our discussion, we will use the UMTS results, as on GPRS the total downloaded data is bounded by the limited bandwidth.

Our hybrid-rendering technique causes a reduction of the downloaded data on high quality tests because sending a panorama takes less network resources that sending the actual geometry. On the other hand, the terrain far from the viewer is roughly approximated by a small number of triangles on low quality tests, so our technique could improve the download ratio when the viewer moves at high speed.

10 Summary and Future Work

Due to the limited computing resources and restricted bandwidth available in current mobile devices 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 proposed a hybrid rendering technique that combines a fast and lightweight multi-hierarchical terrain representation with a 2D panoramic impostor generated on demand by a remote server. This approach allows progressively transferring complex scenes while keeping high visual quality. The workload distribution over the server and the client can be balanced at run-time, according to the computing power of the client, the server and the network congestion.

We have implemented a prototype and carried out a comprehensive set of tests that proved its effectiveness and portability over a wide range of devices and mobile operating systems. The set of experiments were conducted on GPRS (2G) and UMTS (3G) wireless connections. The devices ranged from low-end thin handhelds to powerful smartphones equipped with GPU and to desktop computers.

Currently we are investigating how to enhance the system performance by making a better use of the programmable GPU on the server side when it is available in devices such as the iPhone 3GS.

We plan to develop future work following several lines. To take advantatge of the built-in GPS that most new smartphones include today, we plan to integrate a GIS server into our architecture. The GIS will deliver geospatial data and extra services according to the location of the user. Such services may include roads, cities, name an altitude of the mountains in the line of sight, points of touristic interest, weather services and the like. The addition of a GIS server will also give the opportunity of using our application as a tour guide system.

In addition, we also want to study the use of a prediction mechanism based on the path followed by the viewer that will enable loading tiles in advance. This would result in a smoother streaming of geometry from server to client. Adding vector data like roads from a remote GIS server will also help to develop more advanced prediction mechanism.

Finally, we plan to improve our network protocol by adding support for data compression, to study the roaming effects when the user moves back and forth along neighboring cells and the connection problems that might appear.

Acknowledgements

This work has been partially granted by the Junta de Andalucía, the Spanish Ministry of Education and Science, and the European Union ERDF funds under research projects P06-TIC-01403, P07-TIC-02773, TIN2007-67474-C03-01 and TIN2007-67474-C03-02.

References

[apr09] Apache Portable Runtime (APR). `http://apr.apache.org/`, 2009.

[Bab05] Steve Babbin. *Developing Software for Symbian OS: An Introduction to Creating Smartphone Applications in C++ (Symbian Press)*. John Wiley & Sons, 2005.

[BGMP07] Fabio Bettio, Enrico Gobbetti, Fabio Marton, and Giovanni Pintore. High-quality networked terrain rendering from compressed bitstreams. In *Web3D '07: Proceedings of the twelfth international conference on 3D web technology*, pages 37–44, New York, NY, USA, 2007. ACM.

[bib09a] Android SDK. `http:/code.google.com/int/ca/android`, 2009.

[bib09b] Dell Axim 50v. `http://www.dell.com/content/topics/segtopic.aspx/brand/axim_x50`, 2009.

[bib09c] For Windows Mobile developers. `http://developer.windowsmobile.com`, 2009.

[bib09d] iPhone OS 3.0. `http://www.apple.com/iphone/preview-iphone-os`, 2009.

[bib09e] OpenGL ES - The standard for embedded accelerated 3D graphics. `http://java.sun.com/javame/index.jsp`, 2009.

[bib09f] S60 Devices. `http:/www.es.s60.com/life/s60phones`, 2009.

[BJP08] Azzedine Boukerche, Raed Jarrar, and Richard W.N. Pazzi. 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*, pages 45–52, New York, NY, USA, 2008. ACM.

[BPF08] Azzedine Boukerche, Richard W.N. Pazzi, and Jing Feng. An end-to-end virtual environment streaming technique for thin mobile devices over heterogeneous networks. *Comput. Commun.*, 31(11):2716–2725, 2008.

[CGG$^+$03] Paolo Cignoni, Fabio Ganovelli, Enrico Gobbetti, Fabio Marton, Federico Ponchio, and Roberto Scopigno. BDAM – batched dynamic adaptive meshes for high performance terrain visualization. *Computer Graphics Forum*, 22(3):505–514, September 2003. Proc. Eurographics 2003.

[CLS08] Xingquan Cai, Jinhong Li, and Zhitong Su. Large-scale terrain rendering using strip masks of terrain blocks. In *Intelligent Control and Automation, 2008. WCICA 2008. 7th World Congress on*, pages 1768–1772, June 2008.

[Dav06] A.J. Davidson. *Learning Cocoa with Objective-C*. O'Reilly, 2006.

[DSW09] Christian Dick, Jens Schneider, and Rüdiger Westermann. Efficient geometry compression for GPU-based decoding in realtime terrain rendering. *Computer Graphics Forum*, 28(1):67–83, 2009.

[DWS$^+$97] M. Duchaineau, M. Wolinsky, D.E. Sigeti, M.C. Miller, C. Aldrich, and M.B. Mineev-Weinstein. Roaming terrain: Real-time optimally adapting meshes. *Visualization Conference, IEEE*, 0:81, 1997.

[FvFH90] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer graphics: principles and practice (2nd ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

[FYPA05] Sonia Fahmy, Ossama Younis, Venkatesh Prabhakar, and Srinivas R. Avasarala. *Handbook of Algorithms for Wireless Networking and Mobile Computing*, chapter TCP over Wireless Networkss. Chapman & Hall/CRC, 2005.

[GLF07] GLFW. `http://glfw.sourceforge.net/`, 2007.

[GLT02] GLT OpenGL C++ Toolkit. `http://www.nigels.com/glt/`, 2002.

[GMC$^+$06] Enrico Gobbetti, Fabio Marton, Paolo Cignoni, Marco Di Benedetto, and Fabio Ganovelli. C-BDAM – compressed batched dynamic adaptive meshes for terrain rendering. *Computer Graphics Forum*, 25(3):333–342, September 2006. Proc. Eurographics 2006.

[Gro04] Khronos Group. OpenGL ES - The standard for embedded accelerated 3D graphics. `http://www.khronos.org/`, 2004.

[Jav05] Java Community Process. JSR 184: Mobile 3D Graphics API for J2ME. `http://www.jcp.org/en/jsr/detail?id=184`, 2005.

[JCP06] JCP. Jsr 239: Java binding for the opengles api. `http://www.jcp.org/en/jsr/detail?id=239`, 2006.

[JK07] Seok-Jae Jeong and Arie E. Kaufman. Interactive wireless virtual colonoscopy. *Vis. Comput.*, 23(8):545–557, 2007.

[LH04] Frank Losasso and Hugues Hoppe. Geometry clipmaps: terrain rendering using nested regular grids. *ACM Trans. Graph.*, 23(3):769–776, August 2004.

[LKES09] Yotam Livny, Zvi Kogan, and Jihad El-Sana. Seamless patches for gpu-based terrain rendering. *Visual Computer*, 25(3):197–208, 2009.

[LMG09] R. Lerbour, J.-E. Marvie, and P. Gautron. Adaptive streaming and rendering of large terrains: A generic solution. In *WSCG 2009. Proceedings of the 17-th International Conference on Computer Graphics, Visualization and Computer Vision*, feb 2009.

[LP01] Peter Lindstrom and Valerio Pascucci. Visualization of large terrains made easy. *Visualization Conference, IEEE*, 0:null, 2001.

[LP02] Peter Lindstrom and Valerio Pascucci. Terrain simplification simplified: A general framework for view-dependent out-of-core visualization. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):239–254, 2002.

[LS05] Fabrizio Lamberti and Andrea Sanna. A solution for displaying medical data models on mobile devices. In *SEPADS'05: Proceedings of the 4th WSEAS International Conference on Software Engineering, Parallel & Distributed Systems*, pages 1–7, Stevens Point, Wisconsin, USA, 2005. World Scientific and Engineering Academy and Society (WSEAS).

[MB05] Tom McReynolds and David Blythe. *Advanced Graphics Programming Using OpenGL (The Morgan Kaufmann Series in Computer Graphics)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[Mor66] G.M. Morton. A computer oriented geodetic data base and a new technique in file sequencing. Technical report, 1966.

[NSOJA09] José M. Noguera, Rafael J. Segura, Carlos Ogáyar, and Robert Joan-Arinyo. Arquitectura híbrida para la visualización de gráficos 3D en dispositivos móviles. In *Workshop Aplicaciones de Nuevas Arquitecturas de Consumo y Altas Prestaciones (ANACAP)*, Nov 2009.

[nsp09] Netscape Portable Runtime (NSPR). `http://www.mozilla.org/projects/nspr/`, 2009.

[Paj98] Renato Pajarola. Large scale terrain visualization using the restricted quadtree triangulation. In *VIS '98: Proceedings of the conference on Visualization '98*, pages 19–26, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.

[PAM$^+$07] Kari Pulli, Tomi Aarnio, Ville Miettinen, Kimmo Roimela, and Jani Vaarala. *Mobile 3D graphics with OpenGL ES and M3G*. Morgan Kaufmann, 2007.

[PBH08] R.W.N. Pazzi, A. Boukerche, and Tingxue Huang. Implementation, measurement, and analysis of an image-based virtual environment streaming protocol for wireless mobile devices. *Instrumentation and Measurement, IEEE Transactions on*, 57(9):1894–1907, Sept. 2008.

[PG07] Renato Pajarola and Enrico Gobbetti. Survey of semi-regular multiresolution models for interactive terrain rendering. *Vis. Comput.*, 23(8):583–605, 2007.

[PM05] Joachim Pouderoux and Jean-Eudes Marvie. 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*, pages 299–306, New York, NY, USA, 2005. ACM.

[Pos81] J. Postel. RFC 793: Transmission control protocol, September 1981.

[Pos83] J. Postel. RFC 879: TCP maximum segment size and related topics, November 1983.

[Pow05] PowerVR. *PowerVR MBX. 3D Application Development Recommendations.* Imagination Technologies Ltd., oct 2005.

[RG97] Boris Rabinovich and Craig Gotsman. Visualization of large terrains in resource-limited computing environments. In *VIS '97: Proceedings of the 8th conference on Visualization '97*, pages 95–102, Los Alamitos, CA, USA, 1997. IEEE Computer Society Press.

[Sam89] H. J. Samet. *Design and analysis of Spatial Data Structures: Quadtrees, Octrees, and other Hierarchical Methods*. Addison–Wesley, Redding, MA, 1989.

[SFM09] Simple and Fast Multimedia Library (SFML). `http://www.sfml-dev.org/`, 2009.

[Sha01] Jason Shankel. *Game Programming Gems 2*, chapter Rendering Distant Scenery with Skyboxes. Charles River Media, Inc., Rockland, MA, USA, 2001.

[SRH$^+$98] Stefan Ottger Snroettg, Stefan Röttger, Wolfgang Heidrich, Hans peter Seidel, Graphische Datenverarbeitung (immd, and Universität Erlangen-nürnberg. Real-time generation of continuous levels of detail for height fields. pages 315–322, 1998.

[SRNF08] Rafael J. Segura, Antonio J. Rueda, José M. Noguera, and Francisco Feito. Adaptación de algoritmos geométricos al uso de hardware gráfico programable. In *Workshop Aplicaciones de Nuevas Arquitecturas de Consumo y Altas Prestaciones (ANACAP)*, November 2008.

[SW06] Jens Schneider and Rüdiger Westermann. Gpu-friendly high-quality terrain rendering. *Journal of WSCG*, 14(1-3):49–56, 2006.

[vin04] Vincent 3D. rendering library open source graphics libraries for mobile and embedded devices. `http://www.vincent3d.com/`, 2004.

[VR09] Powe VR. Khronos opengl es 1.x sdk for powervr mbx. `http://http://www.imgtec.com/PowerVR/insider/sdk/KhronosOpenGLES1xMBX.asp`, 2009.

[Wal91] Gregory K. Wallace. The JPEG still picture compression standard. *Communications of ACM*, 34(4):30–44, 1991.

Appendices

A Index of Figures

Table 5 lists the index for results yielded by the Nokia N95 device. Table 6 lists the index for results for the Desktop PC, GeForce 8800GTS GPU device.

		Puget Sound Dataset	
Network	Test	High Quality	Low Quality
UMTS (3G)	Bicycle with Panoramas	Figure 26	Figure 28
	Bicycle without Panoramas	Figure 27	Figure 29
	Car with Panoramas	Figure 30	Figure 32
	Car without Panoramas	Figure 31	Figure 33
	Jet (750 km/h) with Panoramas	Figure 34	Figure 36
	Jet (750 km/h) without Panoramas	Figure 35	Figure 37
	Jet (1000 km/h) with Panoramas	Figure 38	Figure 40
	Jet (1000 km/h) without Panoramas	Figure 39	Figure 41
GPRS (2G)	Bicycle with Panoramas	Figure 42	Figure 44
	Bicycle without Panoramas	Figure 43	Figure 45
	Car with Panoramas	Figure 46	Figure 48
	Car without Panoramas	Figure 47	Figure 49
	Jet (750 km/h) with Panoramas	Figure 50	Figure 52
	Jet (750 km/h) without Panoramas	Figure 51	Figure 53
	Jet (1000 km/h) with Panoramas	-	Figure 54
	Jet (1000 km/h) without Panoramas	-	Figure 55

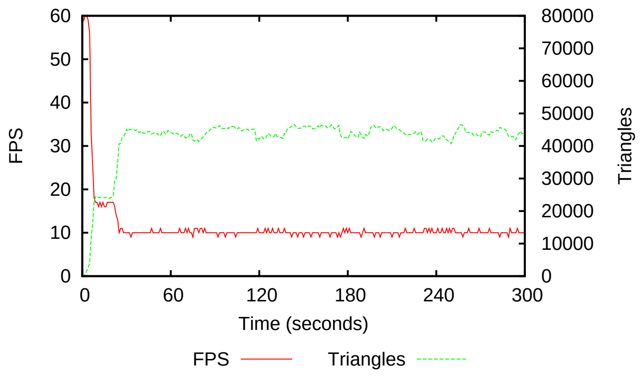
Table 5: Index of Figures for the Nokia N95 device.

		Puget Sound Dataset	
Network	Test	High Quality	Low Quality
UMTS (3G)	Bicycle with Panoramas	Figure 56	Figure 58
	Bicycle without Panoramas	Figure 57	Figure 59
	Car with Panoramas	Figure 60	Figure 62
	Car without Panoramas	Figure 61	Figure 63
	Jet (750 km/h) with Panoramas	Figure 64	Figure 66
	Jet (750 km/h) without Panoramas	Figure 65	Figure 67
	Jet (1000 km/h) with Panoramas	Figure 68	Figure 70
	Jet (1000 km/h) without Panoramas	Figure 69	Figure 71
GPRS (2G)	Bicycle with Panoramas	Figure 72	Figure 74
	Bicycle without Panoramas	Figure 73	Figure 75
	Car with Panoramas	Figure 76	Figure 78
	Car without Panoramas	Figure 77	Figure 79
	Jet (750 km/h) with Panoramas	Figure 80	Figure 82
	Jet (750 km/h) without Panoramas	Figure 81	Figure 83
	Jet (1000 km/h) with Panoramas	Figure 84	Figure 86
	Jet (1000 km/h) without Panoramas	Figure 85	Figure 87

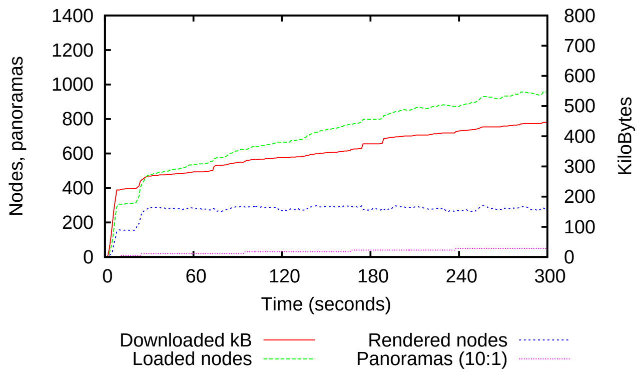
Table 6: Index of Figures. Desktop PC, GeForce 8800GTS GPU.

B Nokia N95 device

B.1 UMTS (3G) wireless network

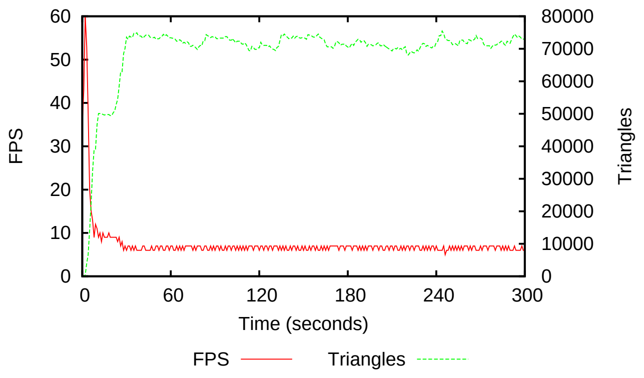


(a)

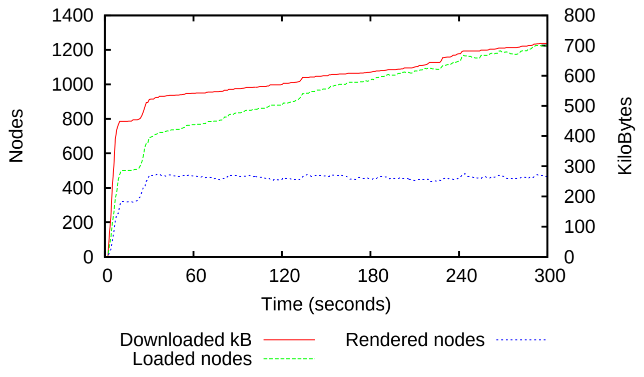


(b)

Figure 26: Perfomance measures for the walkthrough. Top: evolution over time of the archieved frame rate and the amount of triangles rendered. Down: evolution over time of data transferred (kiloBytes), the number of loaded nodes, rendered nodes and transmited panoramas. Nokia N95 device. UMTS (3G) wireless connection. Puget Sound Dataset. High quality. With panoramas. User motion at bicycle speed (25 km/h).

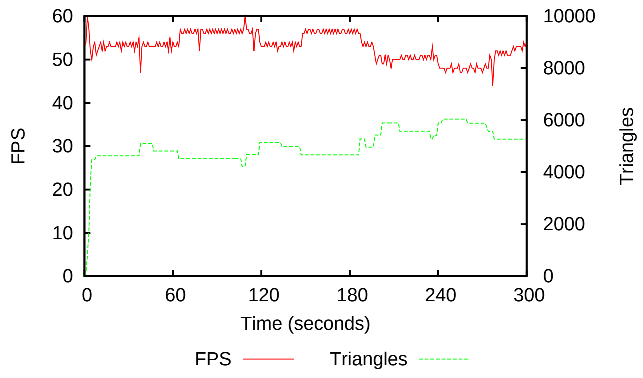


(a)

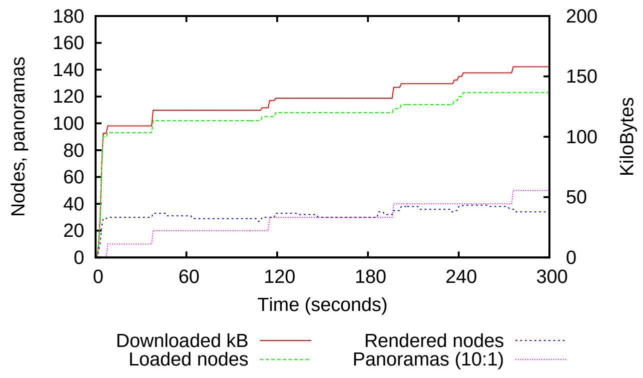


(b)

Figure 27: Perfomance measures for the walkthrough. Top: evolution over time of the archieved frame rate and the amount of triangles rendered. Down: evolution over time of data transferred (kiloBytes), the number of loaded nodes and the number of rendered nodes Nokia N95 device. UMTS (3G) wireless connection. Puget Sound Dataset. High quality. Without panoramas. User motion at bicycle speed (25 km/h).

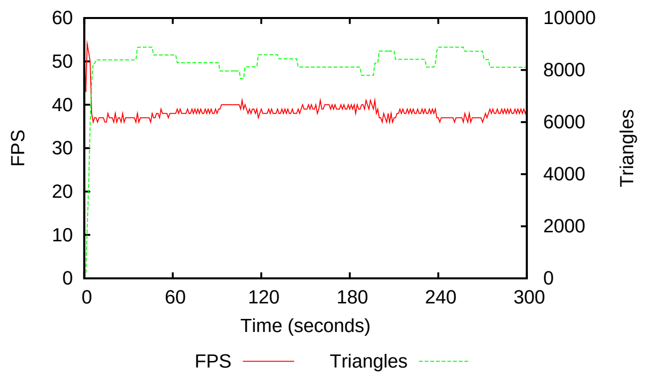


(a)

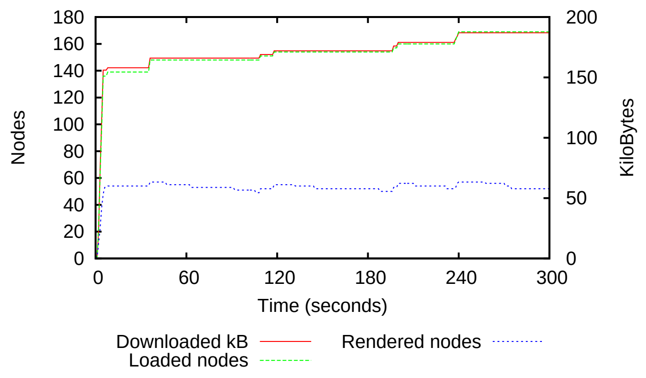


(b)

Figure 28: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 26. Nokia N95 device. UMTS (3G) wireless connection. Puget Sound Dataset. Low quality. With panoramas. User motion at bicycle speed (25 km/h).

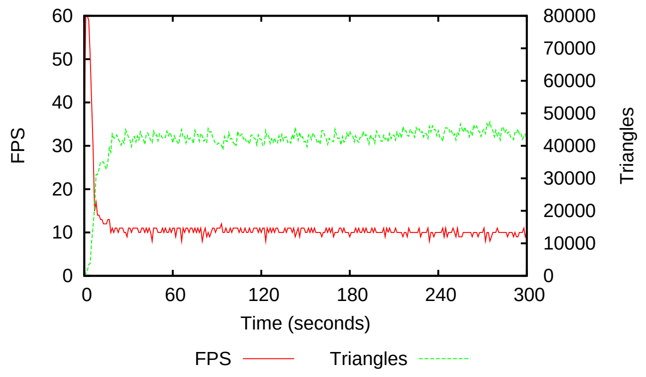


(a)

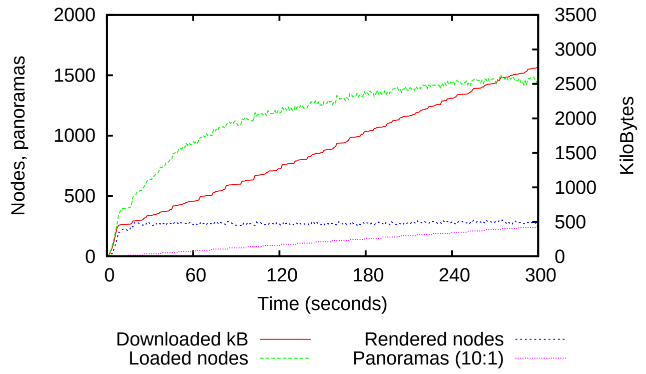


(b)

Figure 29: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 27. Nokia N95 device. UMTS (3G) wireless connection. Puget Sound Dataset. Low quality. Without panoramas. User motion at bicycle speed (25 km/h).

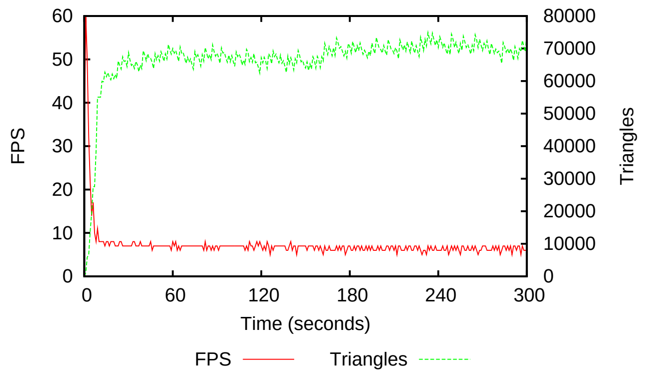


(a)

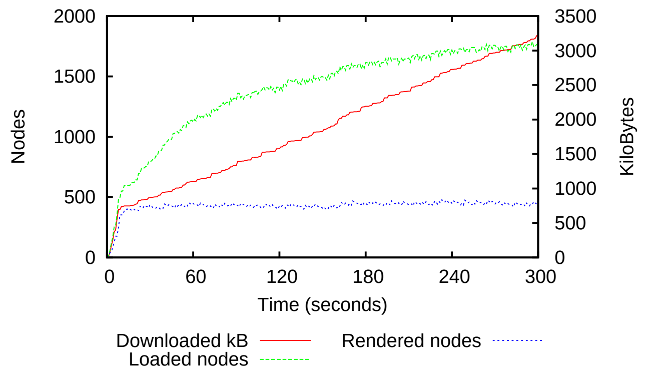


(b)

Figure 30: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 26. Nokia N95 device. UMTS (3G) wireless connection. Puget Sound Dataset. High quality. With panoramas. User motion at car speed (150 km/h).

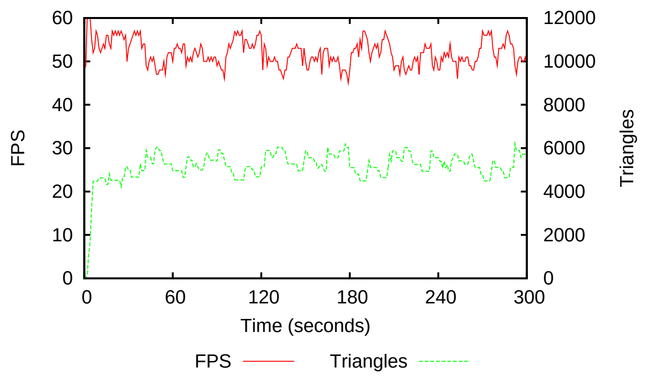


(a)

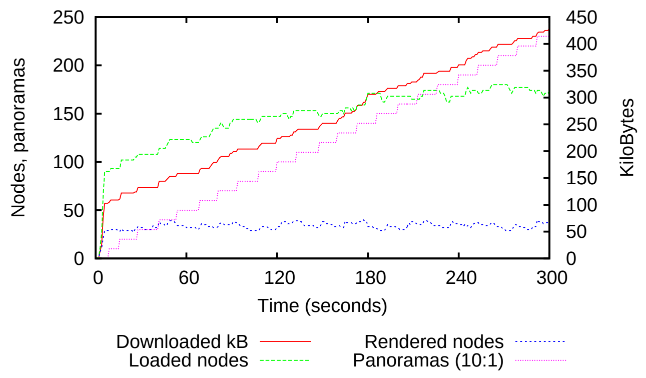


(b)

Figure 31: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 27. Nokia N95 device. UMTS (3G) wireless connection. Puget Sound Dataset. High quality. Without panoramas. User motion at car speed (150 km/h).

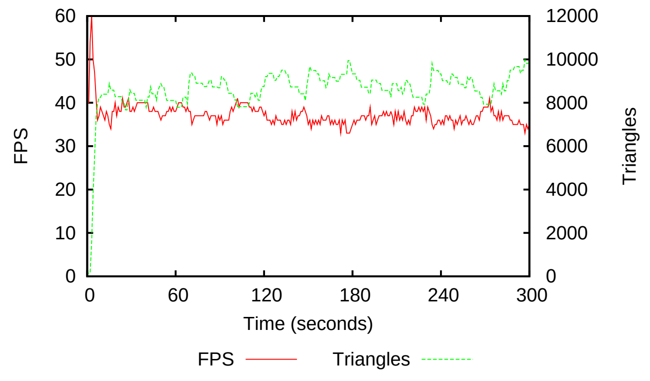


(a)

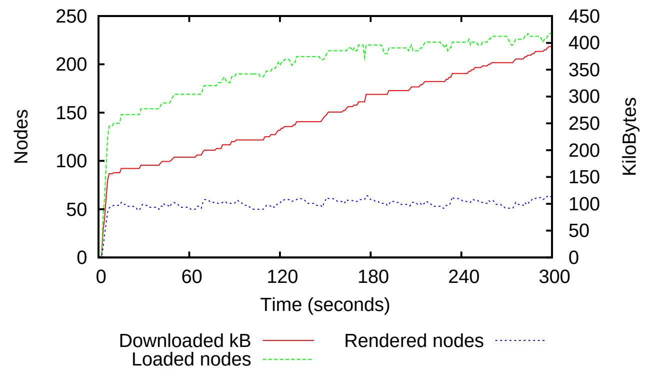


(b)

Figure 32: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 26. Nokia N95 device. UMTS (3G) wireless connection. Puget Sound Dataset. Low quality. With panoramas. User motion at car speed (150 km/h).

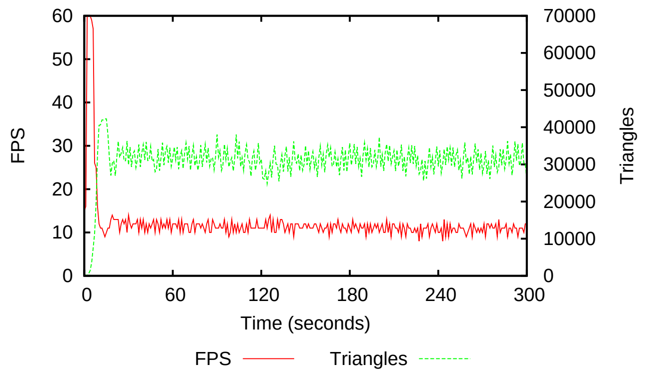


(a)

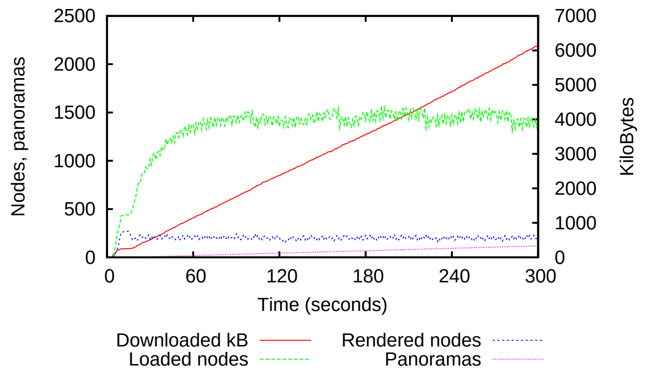


(b)

Figure 33: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 27. Nokia N95 device. UMTS (3G) wireless connection. Puget Sound Dataset. Low quality. Without panoramas. User motion at car speed (150 km/h).

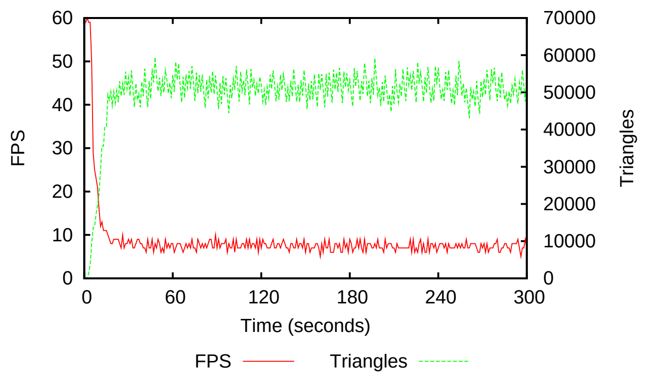


(a)

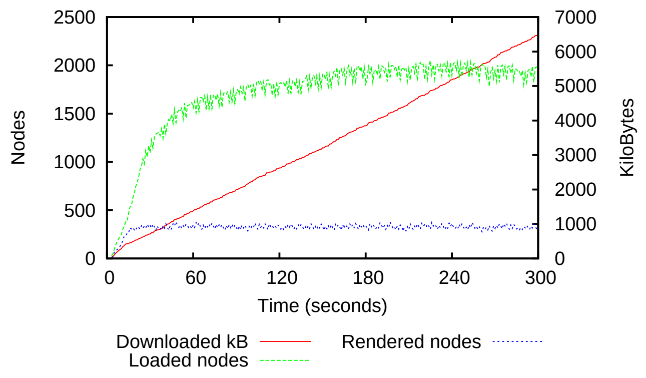


(b)

Figure 34: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 26. Nokia N95 device. UMTS (3G) wireless connection. Puget Sound Dataset. High quality. With panoramas. User motion at jet speed (750 km/h).

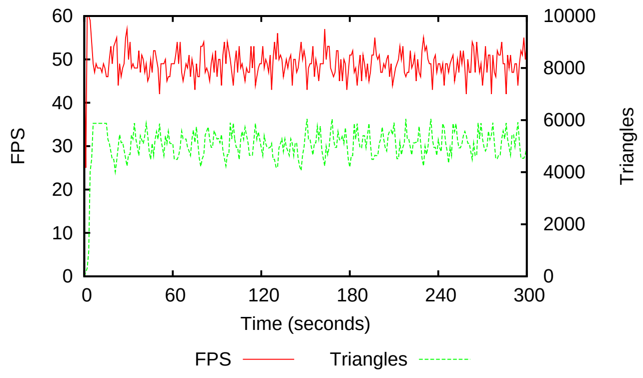


(a)

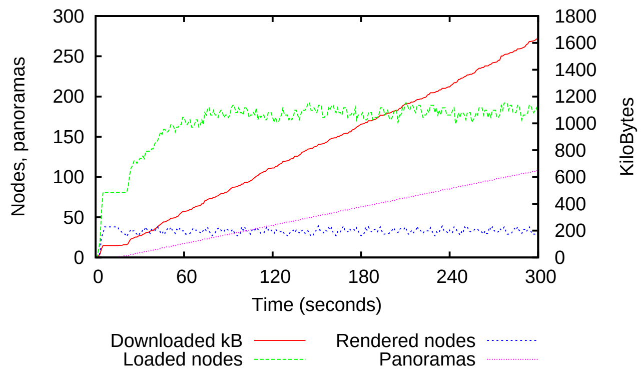


(b)

Figure 35: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 27. Nokia N95 device. UMTS (3G) wireless connection. Puget Sound Dataset. High quality. Without panoramas. User motion at jet speed (750 km/h).

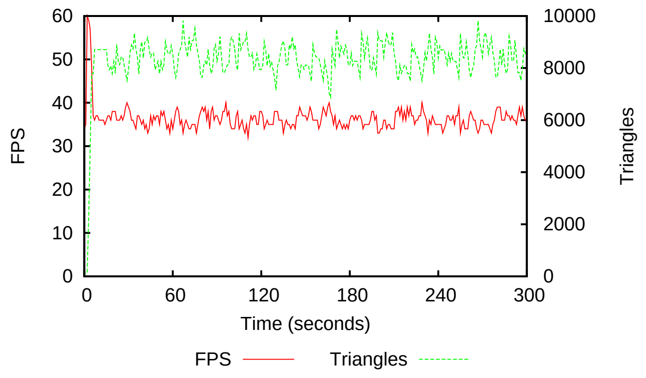


(a)

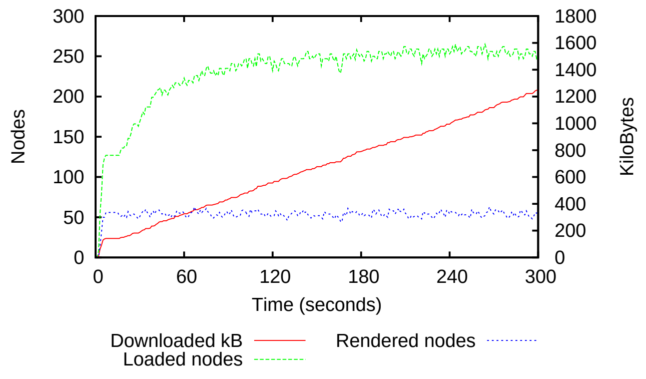


(b)

Figure 36: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 26. Nokia N95 device. UMTS (3G) wireless connection. Puget Sound Dataset. Low quality. With panoramas. User motion at jet speed (750 km/h).

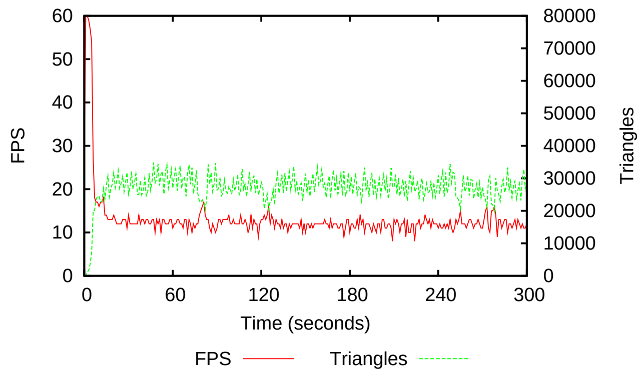


(a)

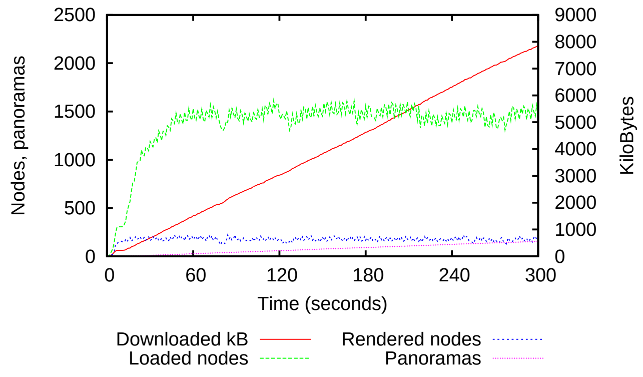


(b)

Figure 37: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 27. Nokia N95 device. UMTS (3G) wireless connection. Puget Sound Dataset. Low quality. Without panoramas. User motion at jet speed (750 km/h).

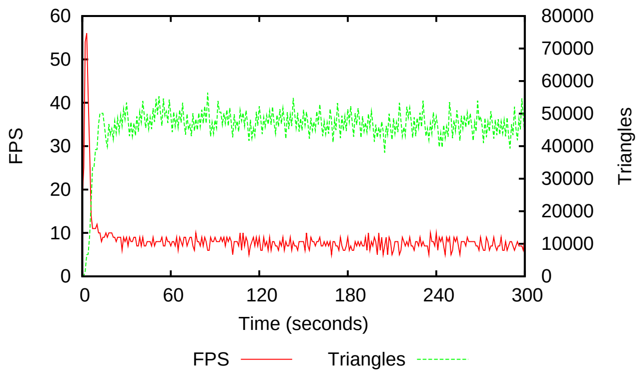


(a)

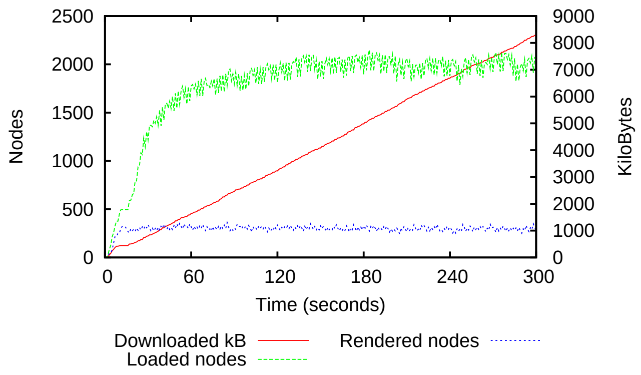


(b)

Figure 38: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 26. Nokia N95 device. UMTS (3G) wireless connection. Puget Sound Dataset. High quality. With panoramas. User motion at jet speed (1000 km/h).

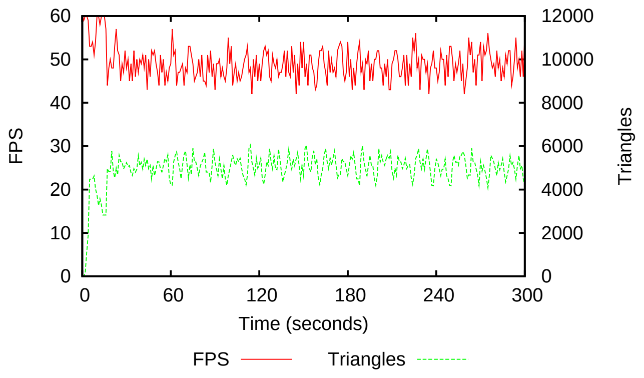


(a)

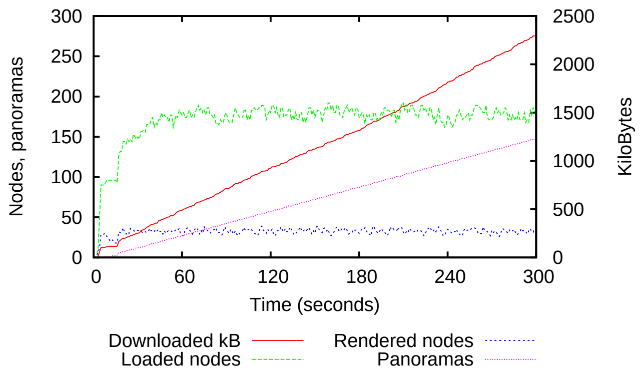


(b)

Figure 39: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 27. Nokia N95 device. UMTS (3G) wireless connection. Puget Sound Dataset. High quality. Without panoramas. User motion at jet speed (1000 km/h).

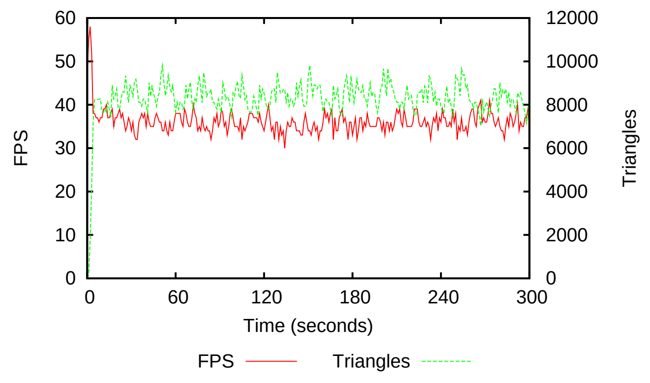


(a)

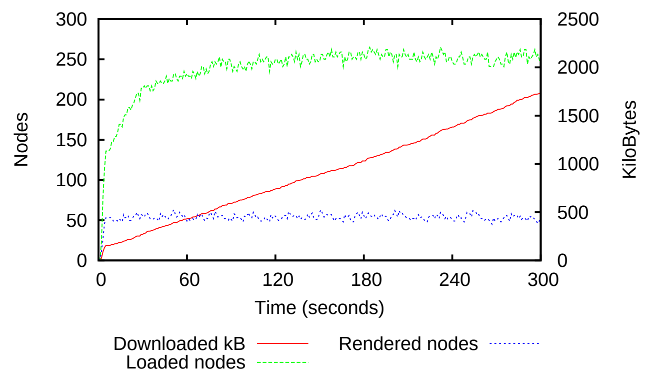


(b)

Figure 40: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 26. Nokia N95 device. UMTS (3G) wireless connection. Puget Sound Dataset. Low quality. With panoramas. User motion at jet speed (1000 km/h).



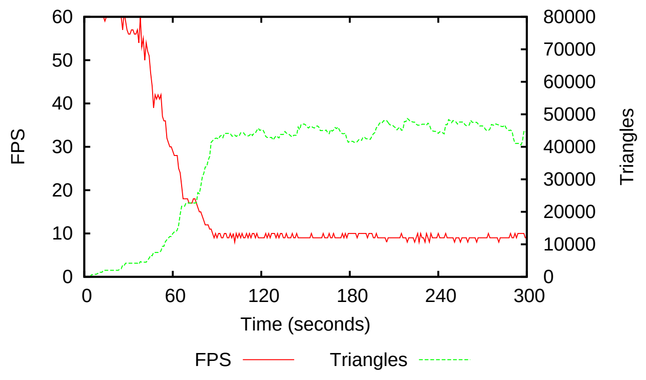
(a)



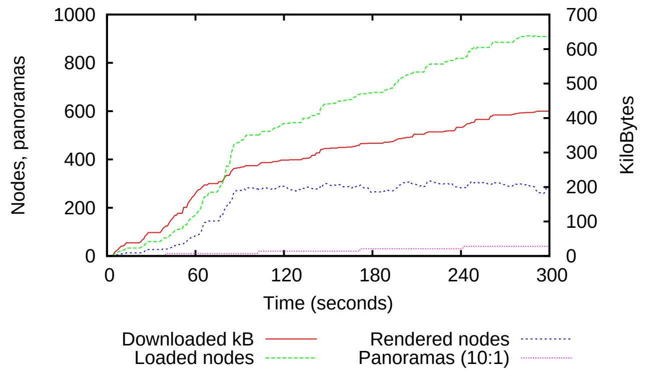
(b)

Figure 41: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 27. Nokia N95 device. UMTS (3G) wireless connection. Puget Sound Dataset. Low quality. Without panoramas. User motion at jet speed (1000 km/h).

B.2 GPRS (2G) wireless network

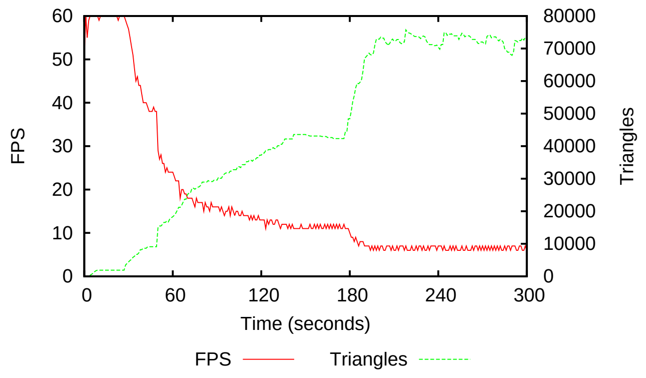


(a)

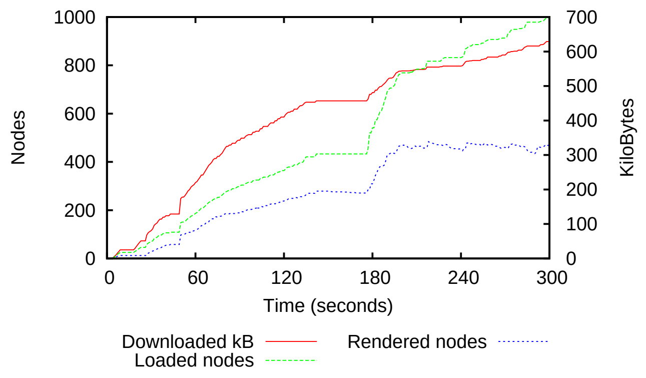


(b)

Figure 42: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 26. Nokia N95 device. GPRS (2G) wireless connection. Puget Sound Dataset. High quality. With panoramas. User motion at bicycle speed (25 km/h).

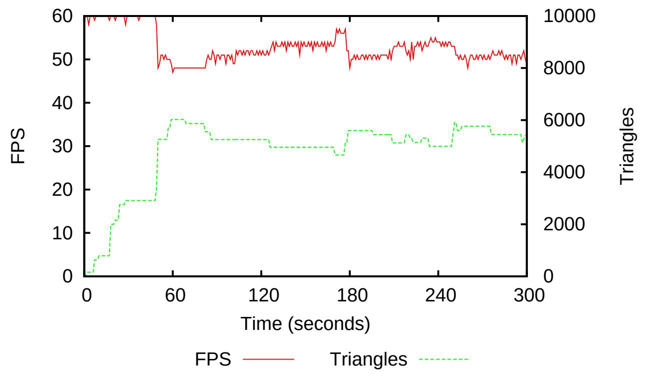


(a)

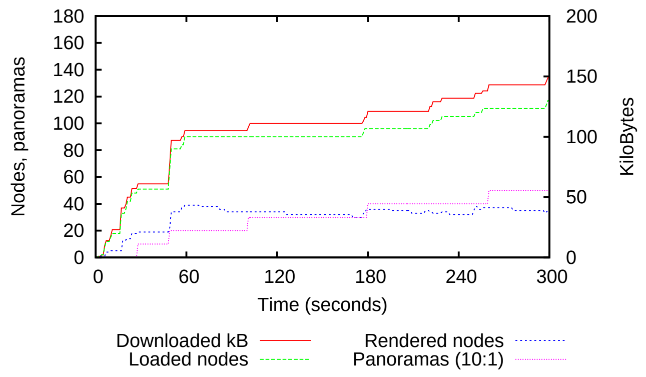


(b)

Figure 43: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 27. Nokia N95 device. GPRS (2G) wireless connection. Puget Sound Dataset. High quality. Without panoramas. User motion at bicycle speed (25 km/h).

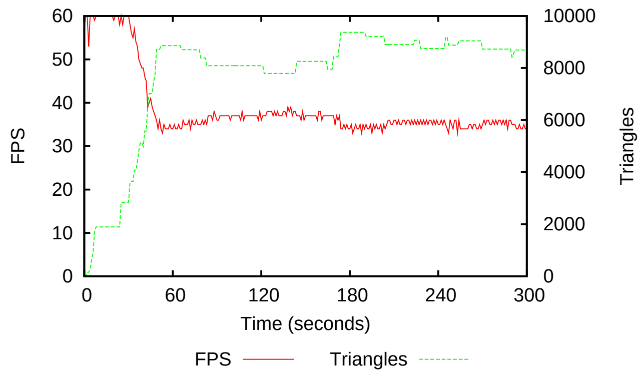


(a)

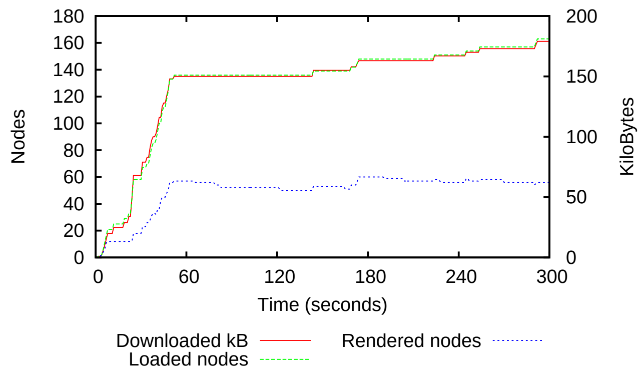


(b)

Figure 44: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 26. Nokia N95 device. GPRS (2G) wireless connection. Puget Sound Dataset. Low quality. With panoramas. User motion at bicycle speed (25 km/h).

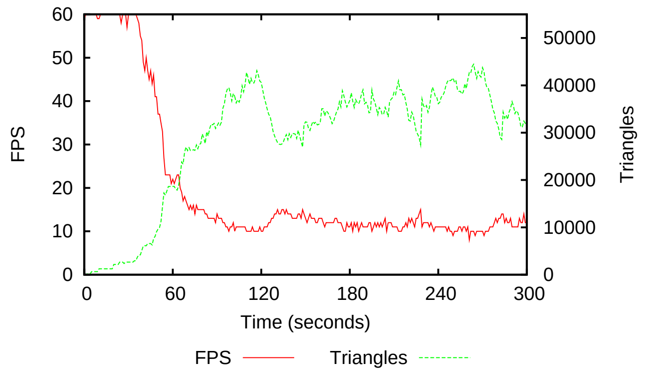


(a)

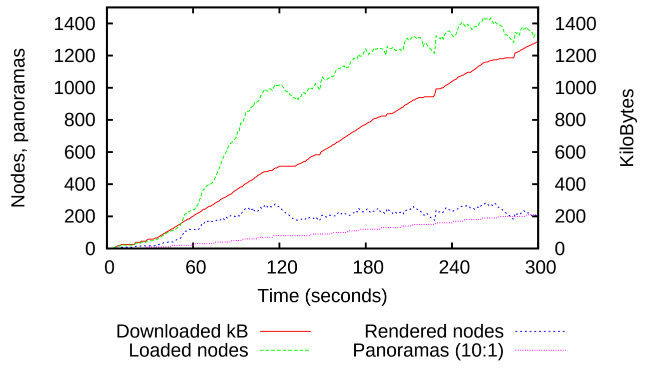


(b)

Figure 45: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 27. Nokia N95 device. GPRS (2G) wireless connection. Puget Sound Dataset. Low quality. Without panoramas. User motion at bicycle speed (25 km/h).

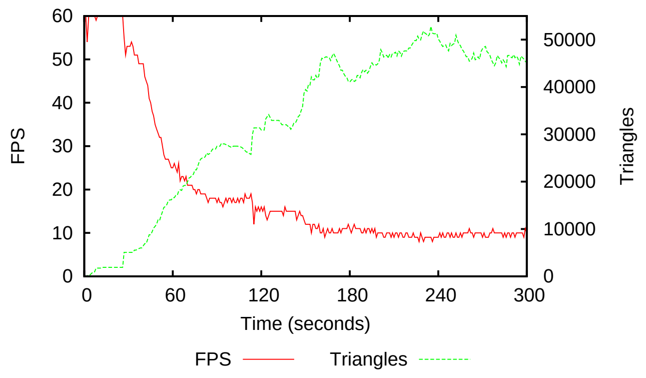


(a)

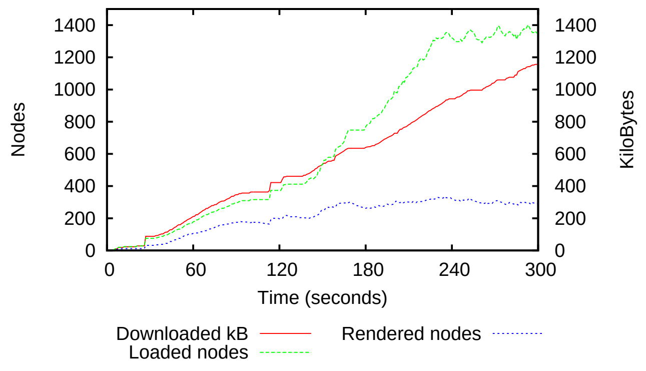


(b)

Figure 46: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 26. Nokia N95 device. GPRS (2G) wireless connection. Puget Sound Dataset. High quality. With panoramas. User motion at car speed (150 km/h).

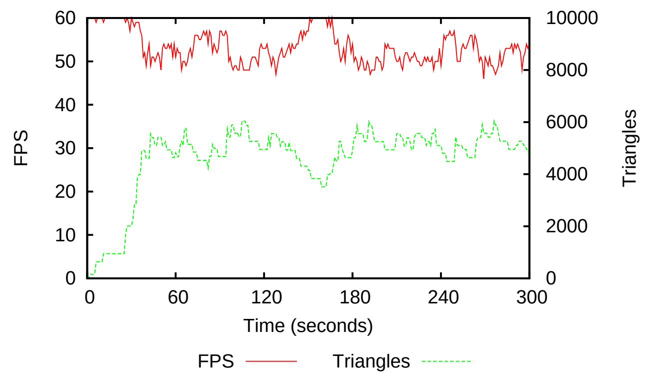


(a)

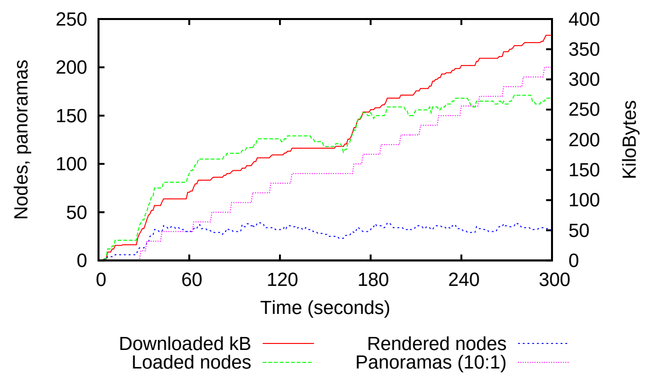


(b)

Figure 47: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 27. Nokia N95 device. GPRS (2G) wireless connection. Puget Sound Dataset. High quality. Without panoramas. User motion at car speed (150 km/h).

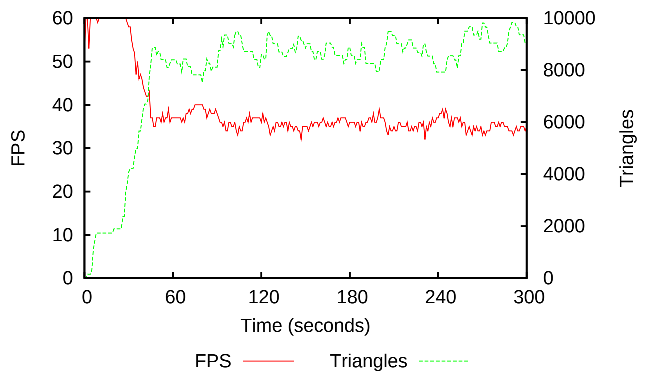


(a)

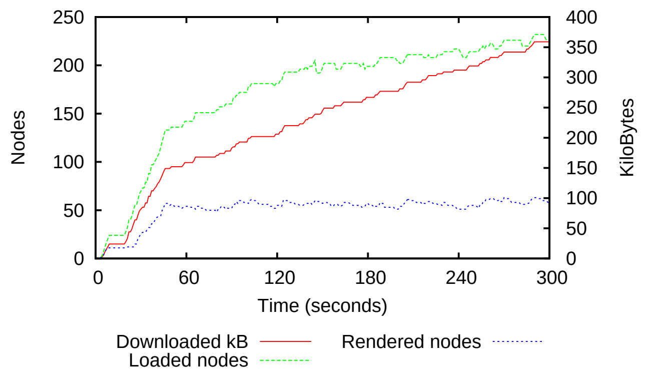


(b)

Figure 48: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 26. Nokia N95 device. GPRS (2G) wireless connection. Puget Sound Dataset. Low quality. With panoramas. User motion at car speed (150 km/h).

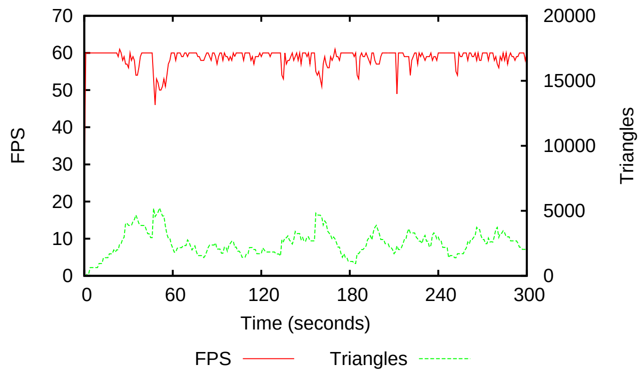


(a)

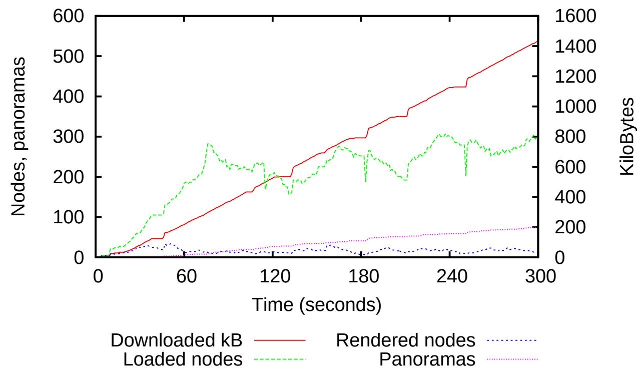


(b)

Figure 49: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 27. Nokia N95 device. GPRS (2G) wireless connection. Puget Sound Dataset. Low quality. Without panoramas. User motion at car speed (150 km/h).

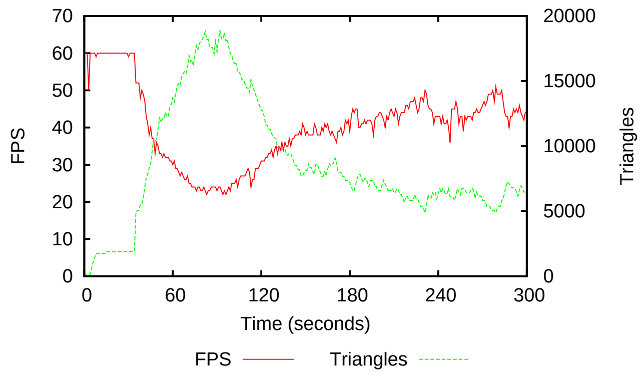


(a)

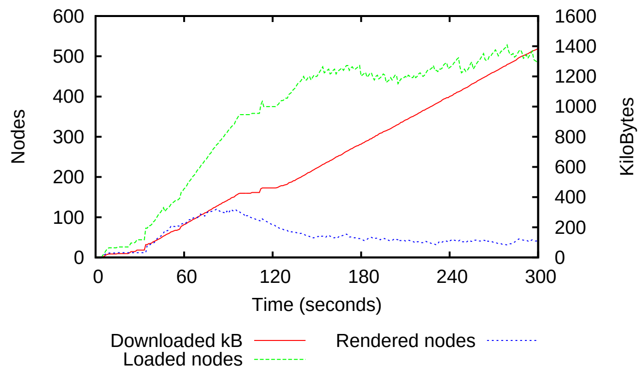


(b)

Figure 50: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 26. Nokia N95 device. GPRS (2G) wireless connection. Puget Sound Dataset. High quality. With panoramas. User motion at jet speed (750 km/h).

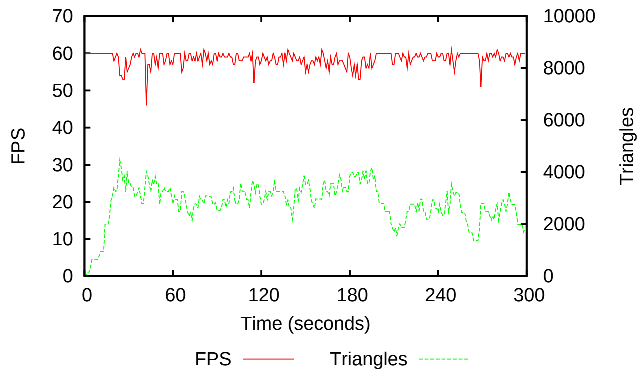


(a)

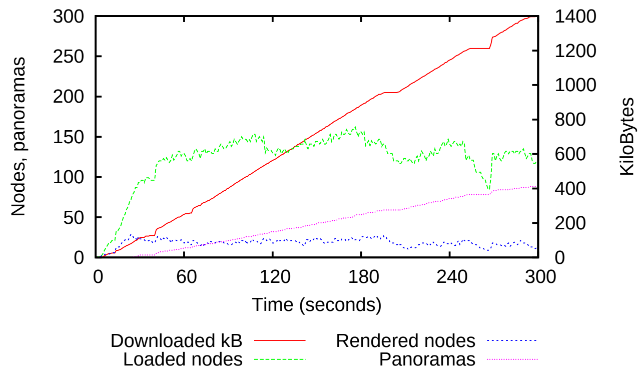


(b)

Figure 51: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 27. Nokia N95 device. GPRS (2G) wireless connection. Puget Sound Dataset. High quality. Without panoramas. User motion at jet speed (750 km/h).

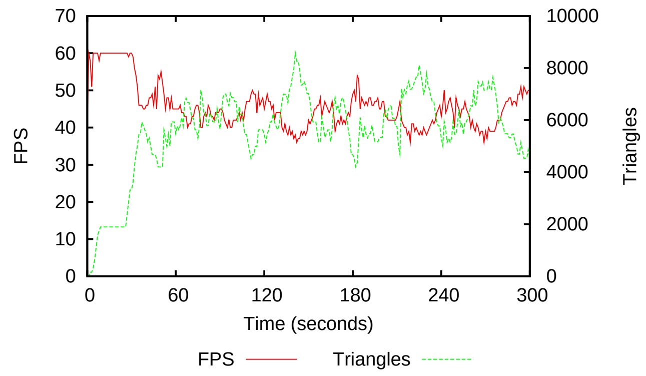


(a)

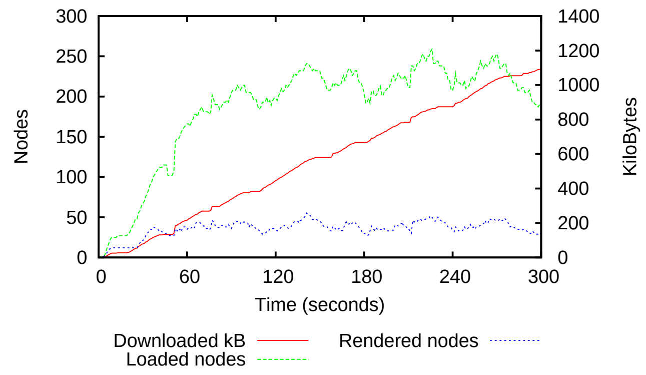


(b)

Figure 52: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 26. Nokia N95 device. GPRS (2G) wireless connection. Puget Sound Dataset. Low quality. With panoramas. User motion at jet speed (750 km/h).

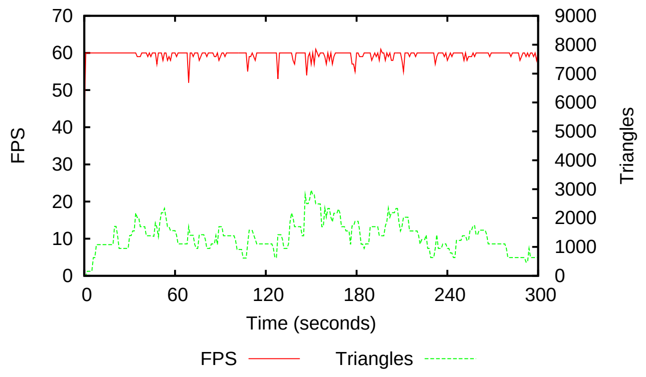


(a)

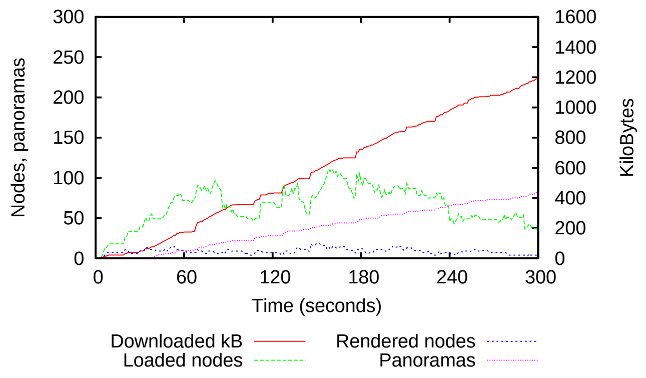


(b)

Figure 53: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 27. Nokia N95 device. GPRS (2G) wireless connection. Puget Sound Dataset. Low quality. Without panoramas. User motion at car speed (750 km/h).

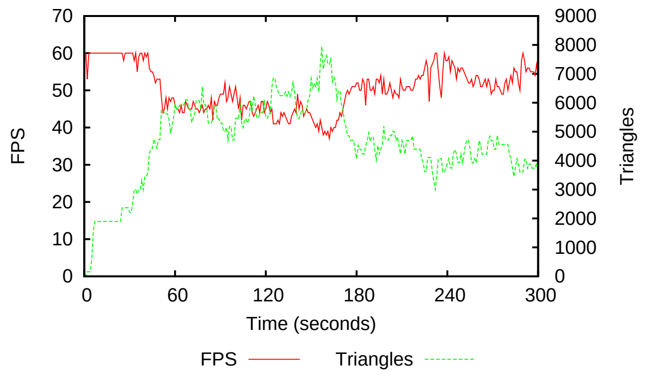


(a)

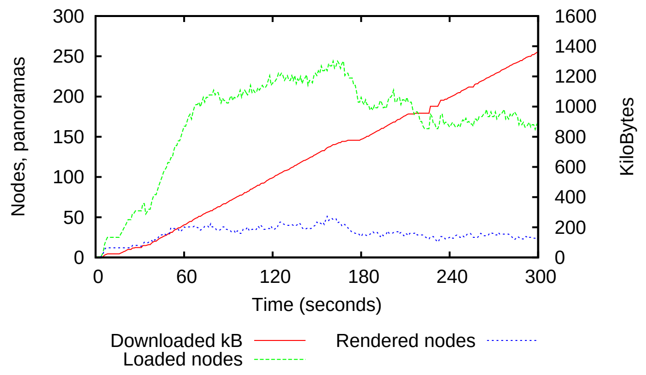


(b)

Figure 54: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 26. Nokia N95 device. GPRS (2G) wireless connection. Puget Sound Dataset. Low quality. With panoramas. User motion at jet speed (1000 km/h).



(a)

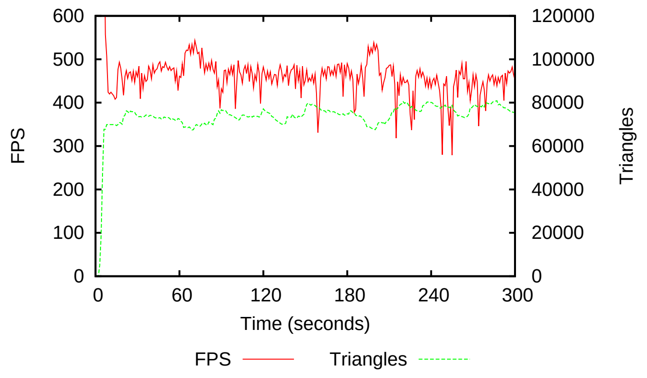


(b)

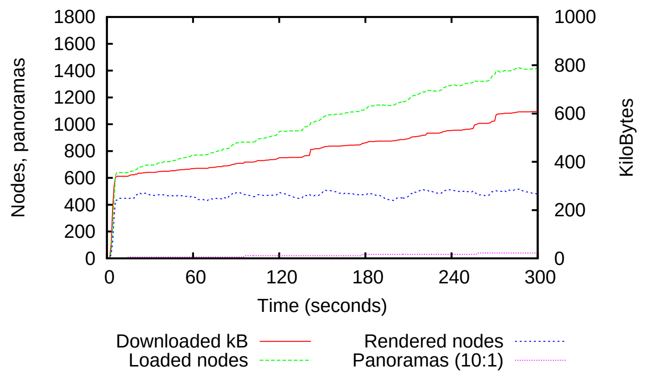
Figure 55: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 27. Nokia N95 device. GPRS (2G) wireless connection. Puget Sound Dataset. Low quality. Without panoramas. User motion at jet speed (1000 km/h).

C Desktop PC, GeForce 8800GTS GPU

C.0.1 UMTS (3G) wireless network

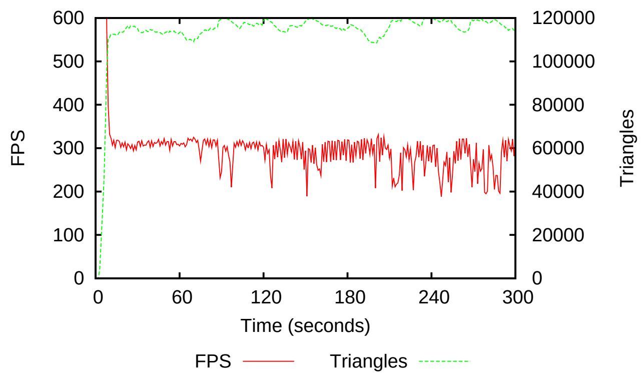


(a)

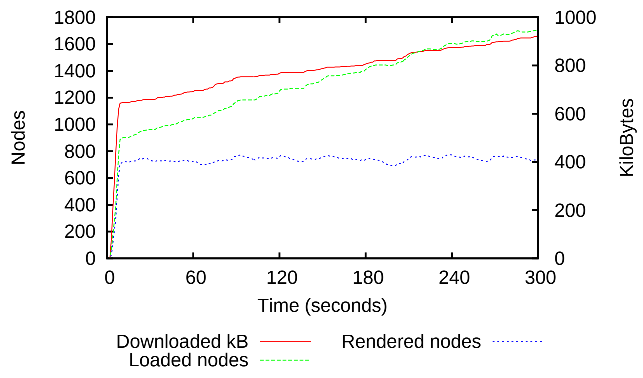


(b)

Figure 56: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 26. Desktop PC, GeForce 8800GTS GPU. UMTS (3G) wireless connection. Puget Sound Dataset. High quality. With panoramas. User motion at bicycle speed (25 km/h).

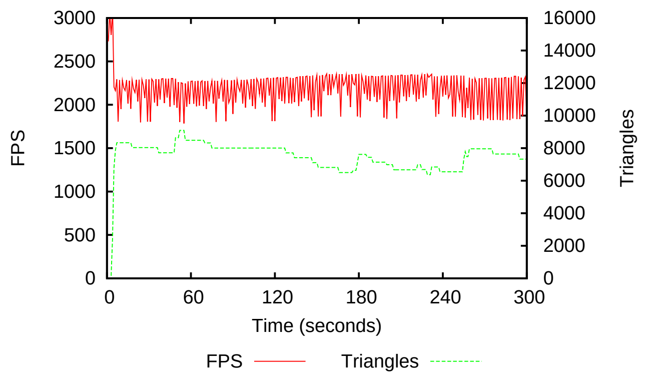


(a)

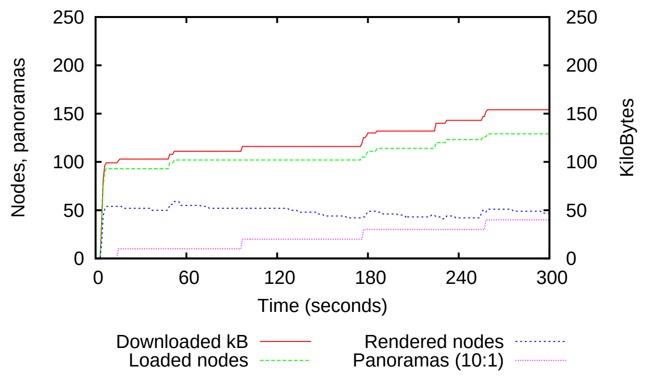


(b)

Figure 57: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 27. Desktop PC, GeForce 8800GTS GPU. UMTS (3G) wireless connection. Puget Sound Dataset. High quality. Without panoramas. User motion at bicycle speed (25 km/h).

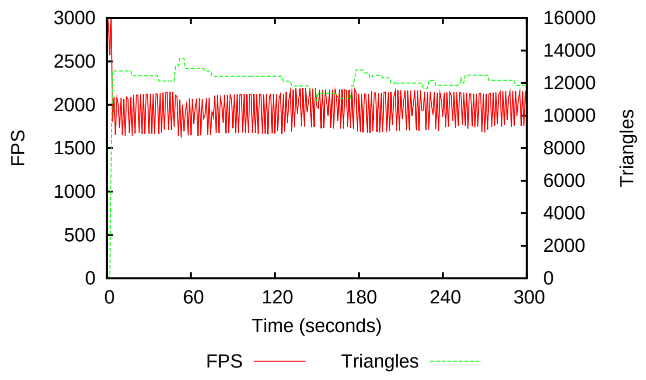


(a)

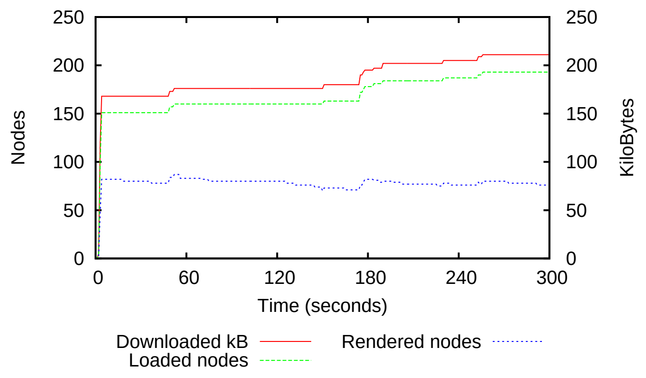


(b)

Figure 58: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 26. Desktop PC, GeForce 8800GTS GPU. UMTS (3G) wireless connection. Puget Sound Dataset. Low quality. With panoramas. User motion at bicycle speed (25 km/h).

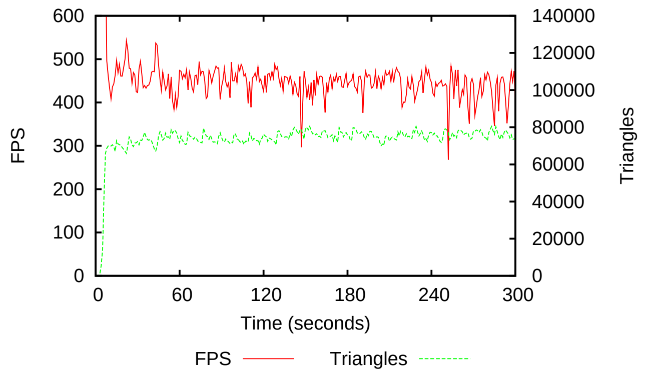


(a)

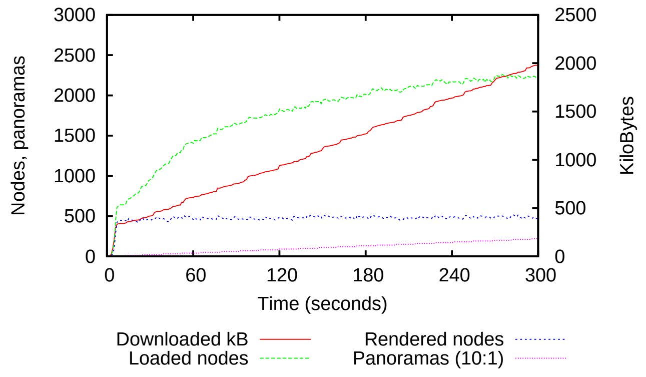


(b)

Figure 59: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 27. Desktop PC, GeForce 8800GTS GPU. UMTS (3G) wireless connection. Puget Sound Dataset. Low quality. Without panoramas. User motion at bicycle speed (25 km/h).

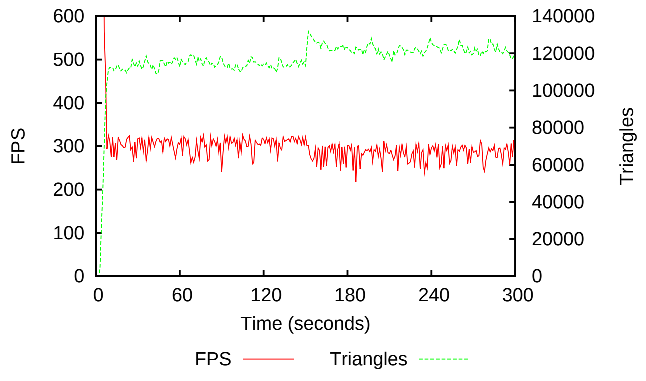


(a)

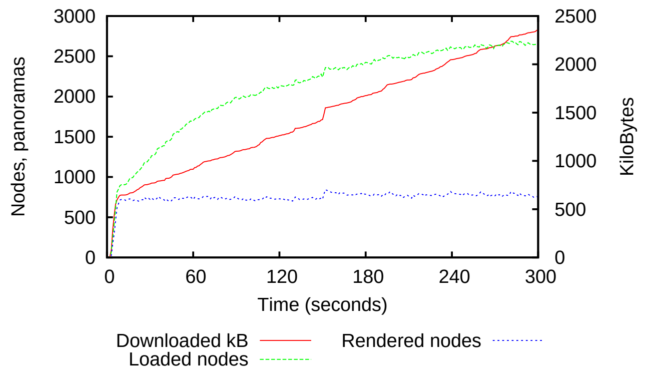


(b)

Figure 60: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 26. Desktop PC, GeForce 8800GTS GPU. UMTS (3G) wireless connection. Puget Sound Dataset. High quality. With panoramas. User motion at car speed (150 km/h).

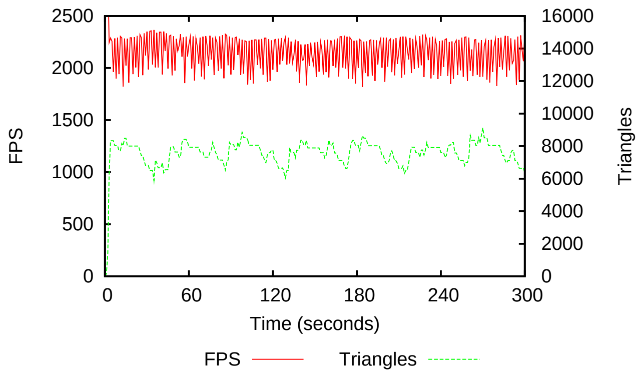


(a)

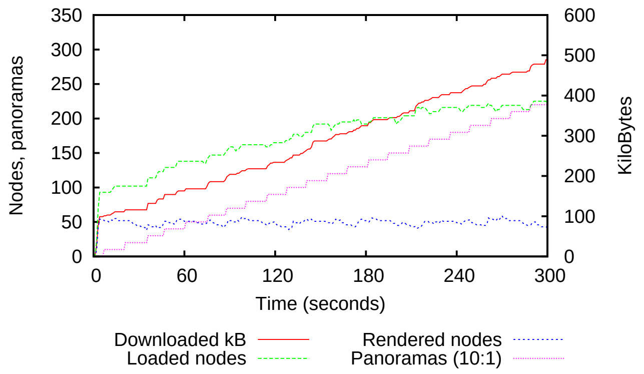


(b)

Figure 61: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 27. Desktop PC, GeForce 8800GTS GPU. UMTS (3G) wireless connection. Puget Sound Dataset. High quality. Without panoramas. User motion at car speed (150 km/h).

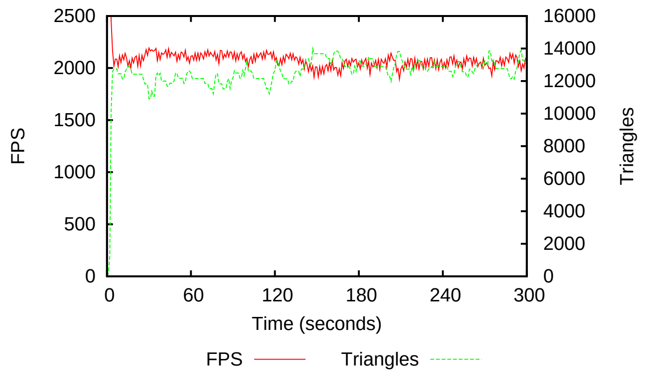


(a)

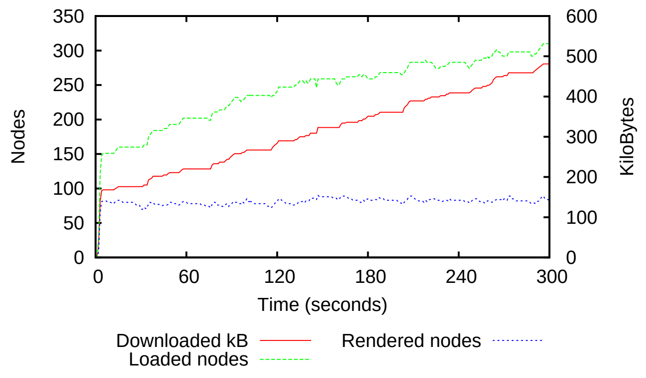


(b)

Figure 62: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 26. Desktop PC, GeForce 8800GTS GPU. UMTS (3G) wireless connection. Puget Sound Dataset. Low quality. With panoramas. User motion at car speed (150 km/h).

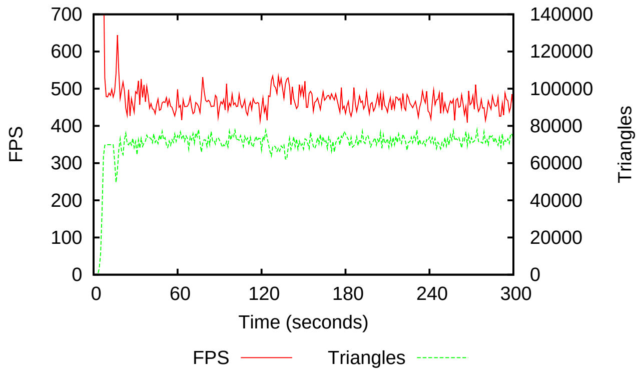


(a)

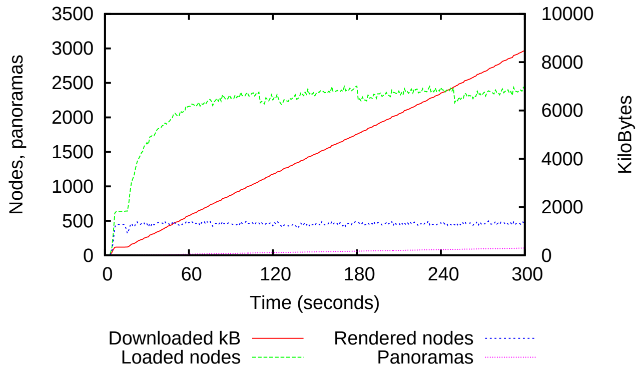


(b)

Figure 63: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 27. Desktop PC, GeForce 8800GTS GPU. UMTS (3G) wireless connection. Puget Sound Dataset. Low quality. Without panoramas. User motion at car speed (150 km/h).

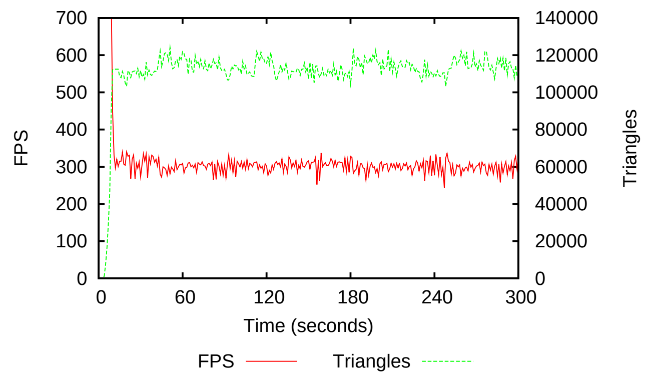


(a)

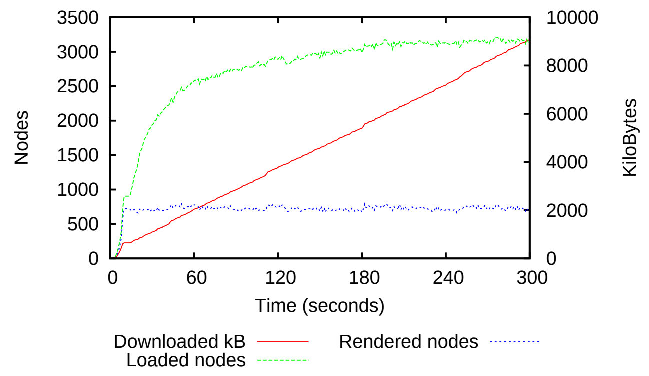


(b)

Figure 64: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 26. Desktop PC, GeForce 8800GTS GPU. UMTS (3G) wireless connection. Puget Sound Dataset. High quality. With panoramas. User motion at jet speed (750 km/h).

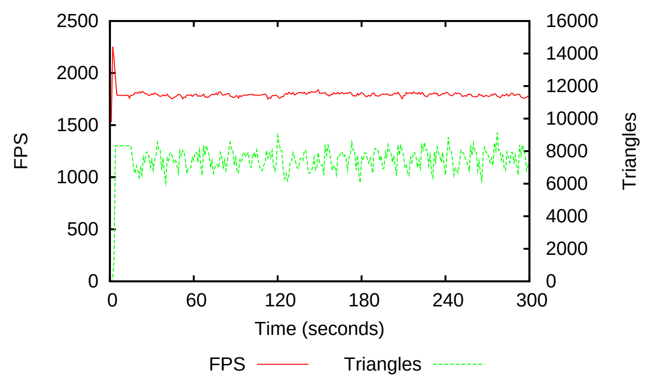


(a)

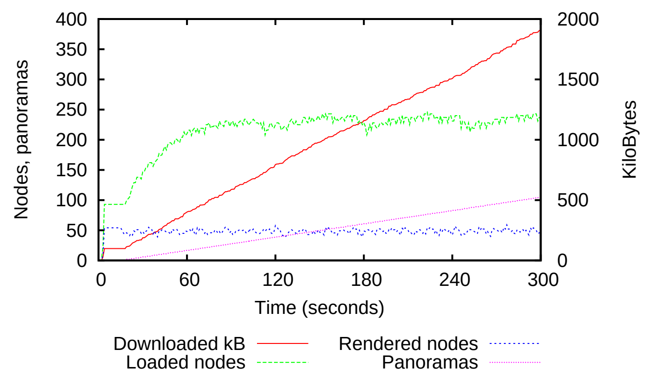


(b)

Figure 65: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 27. Desktop PC, GeForce 8800GTS GPU. UMTS (3G) wireless connection. Puget Sound Dataset. High quality. Without panoramas. User motion at jet speed (750 km/h).

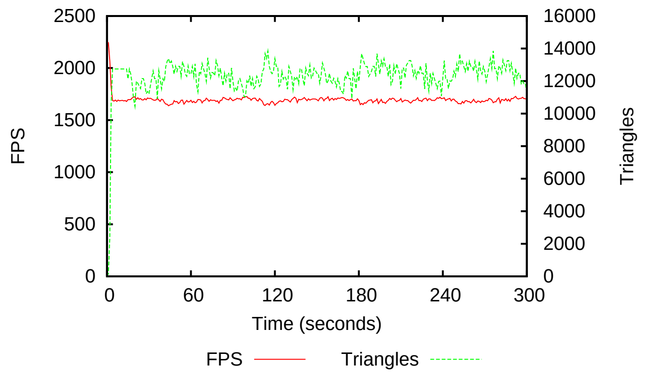


(a)

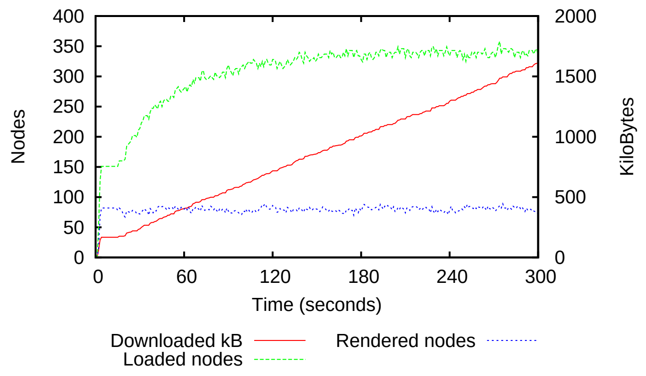


(b)

Figure 66: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 26. Desktop PC, GeForce 8800GTS GPU. UMTS (3G) wireless connection. Puget Sound Dataset. Low quality. With panoramas. User motion at jet speed (750 km/h).

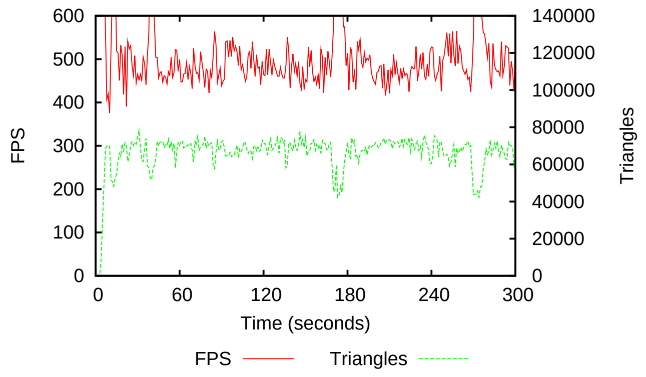


(a)

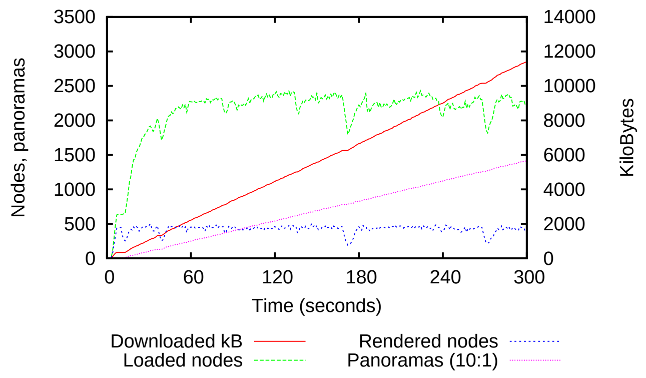


(b)

Figure 67: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 27. Desktop PC, GeForce 8800GTS GPU. UMTS (3G) wireless connection. Puget Sound Dataset. Low quality. Without panoramas. User motion at jet speed (750 km/h).

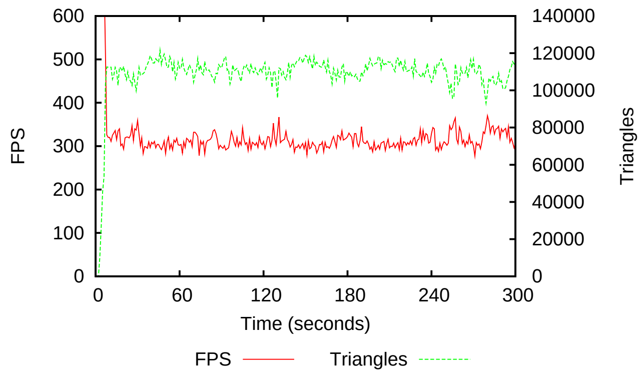


(a)

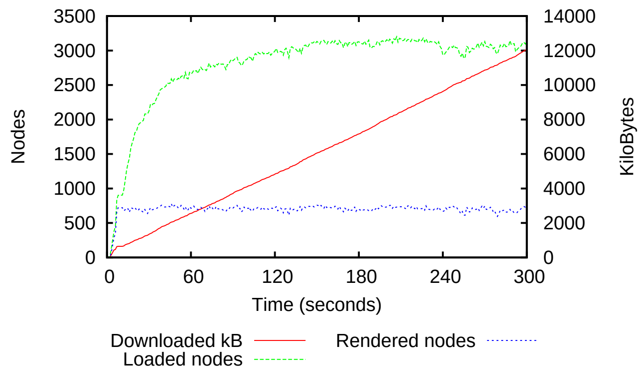


(b)

Figure 68: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 26. Desktop PC, GeForce 8800GTS GPU. UMTS (3G) wireless connection. Puget Sound Dataset. High quality. With panoramas. User motion at jet speed (1000 km/h).

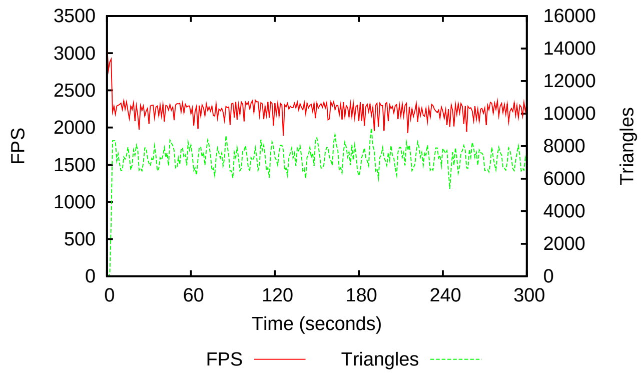


(a)

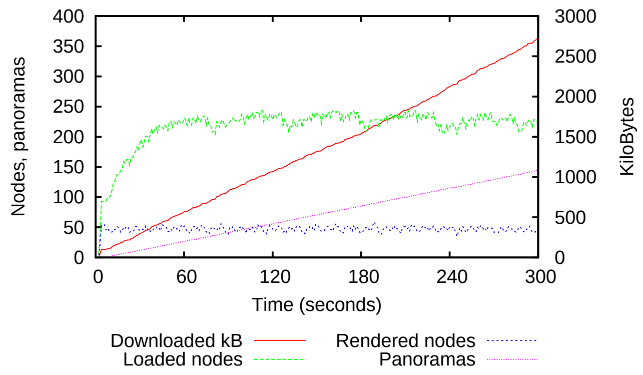


(b)

Figure 69: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 27. Desktop PC, GeForce 8800GTS GPU. UMTS (3G) wireless connection. Puget Sound Dataset. High quality. Without panoramas. User motion at jet speed (1000 km/h).

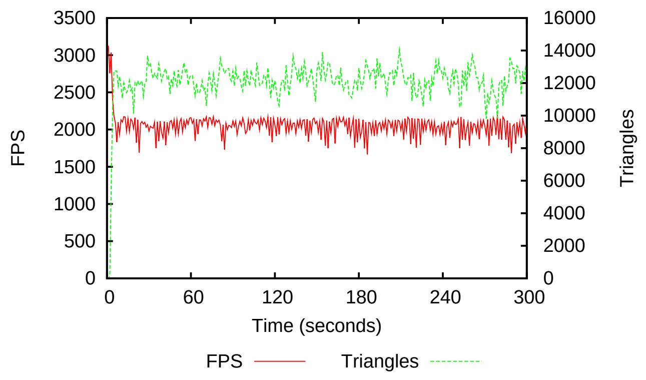


(a)

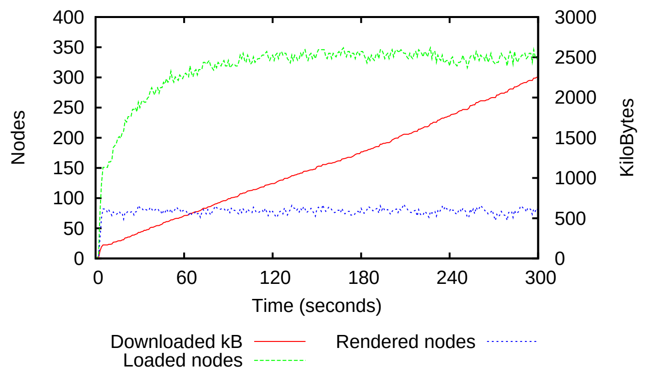


(b)

Figure 70: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 26. Desktop PC, GeForce 8800GTS GPU. UMTS (3G) wireless connection. Puget Sound Dataset. Low quality. With panoramas. User motion at jet speed (1000 km/h).



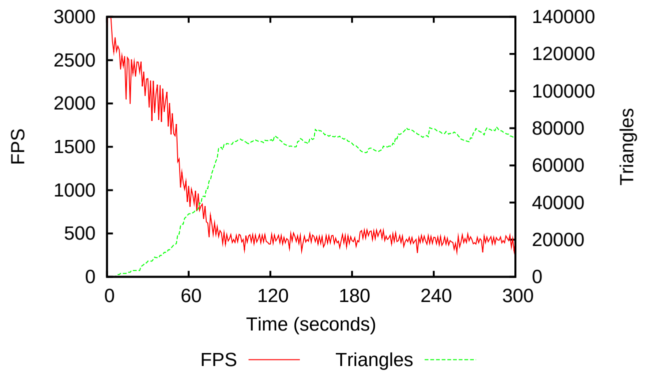
(a)



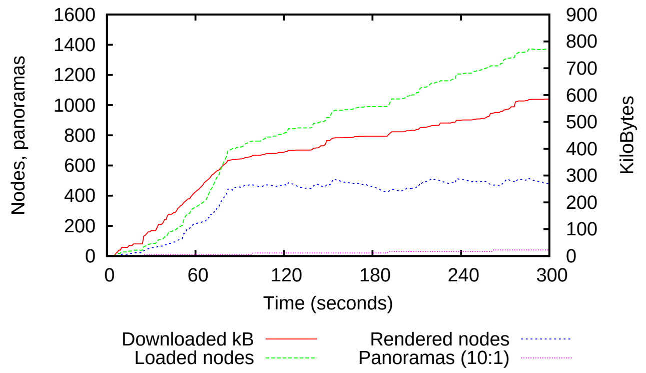
(b)

Figure 71: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 27. Desktop PC, GeForce 8800GTS GPU. UMTS (3G) wireless connection. Puget Sound Dataset. Low quality. Without panoramas. User motion at jet speed (1000 km/h).

C.1 GPRS (2G) wireless network

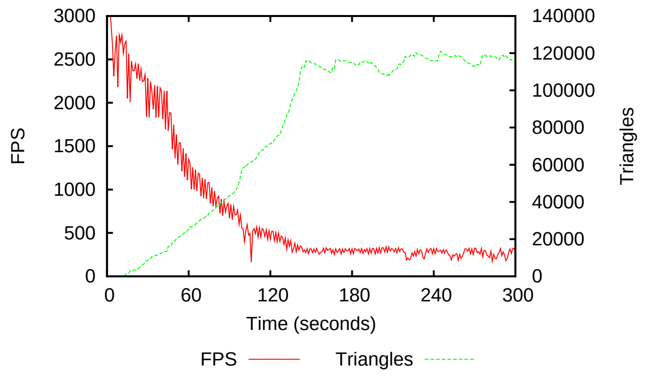


(a)

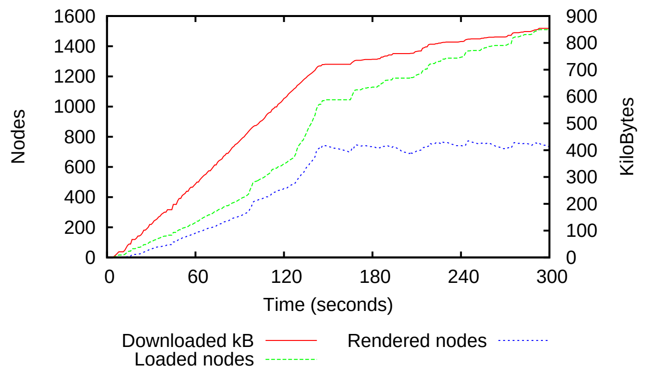


(b)

Figure 72: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 26. Desktop PC, GeForce 8800GTS GPU. GPRS (2G) wireless connection. Puget Sound Dataset. High quality. With panoramas. User motion at bicycle speed (25 km/h).

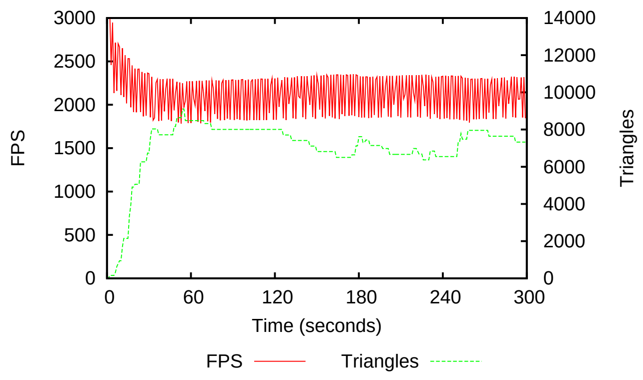


(a)

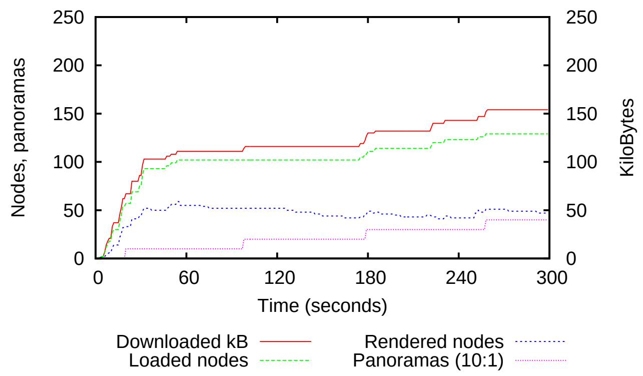


(b)

Figure 73: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 27. Desktop PC, GeForce 8800GTS GPU. GPRS (2G) wireless connection. Puget Sound Dataset. High quality. Without panoramas. User motion at bicycle speed (25 km/h).

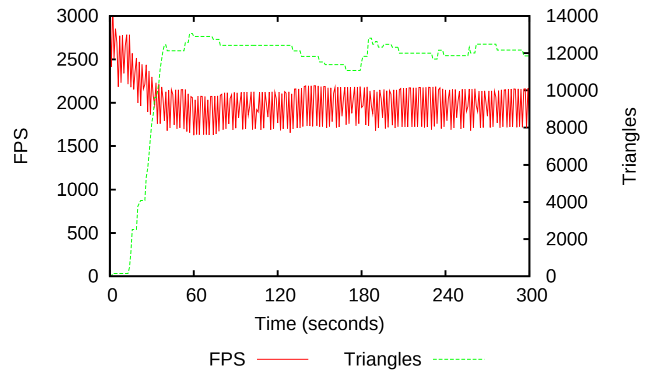


(a)

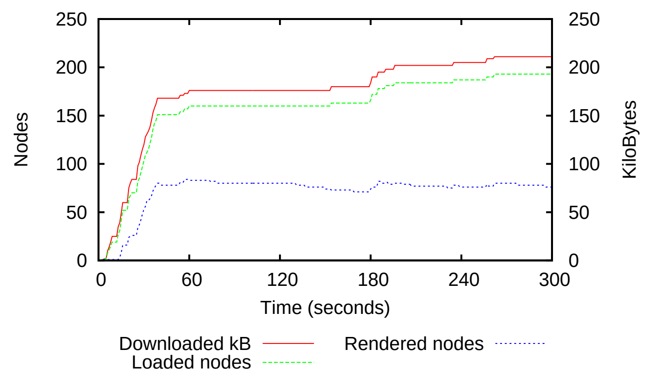


(b)

Figure 74: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 26. Desktop PC, GeForce 8800GTS GPU. GPRS (2G) wireless connection. Puget Sound Dataset. Low quality. With panoramas. User motion at bicycle speed (25 km/h).

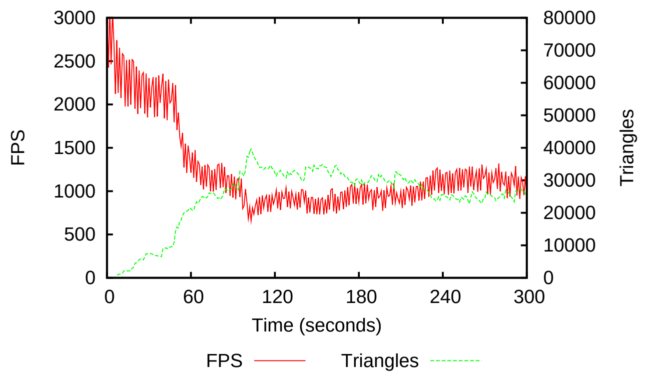


(a)

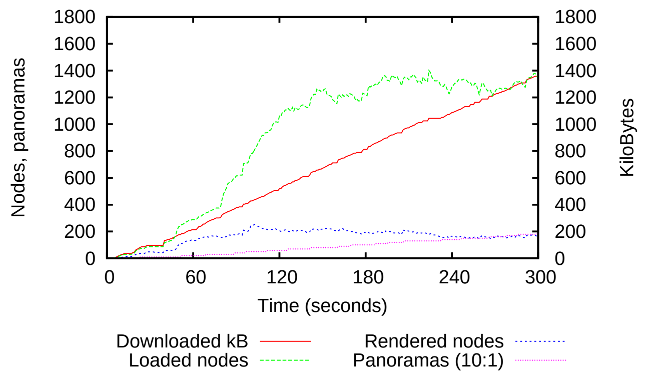


(b)

Figure 75: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 27. Desktop PC, GeForce 8800GTS GPU. GPRS (2G) wireless connection. Puget Sound Dataset. Low quality. Without panoramas. User motion at bicycle speed (25 km/h).

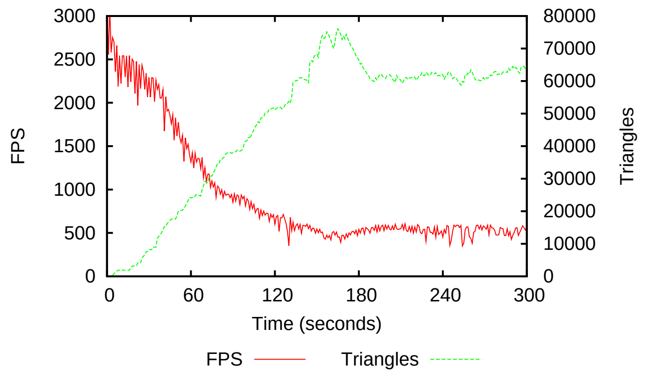


(a)

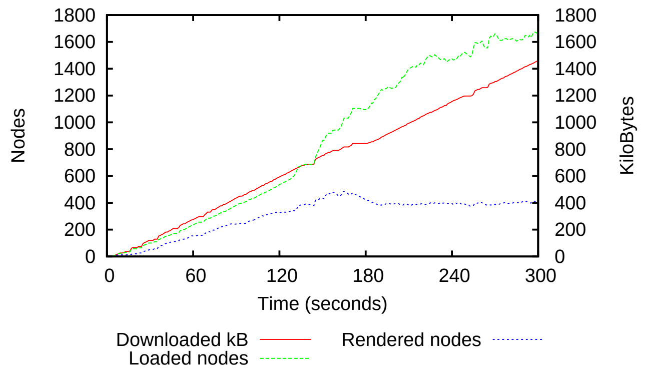


(b)

Figure 76: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 26. Desktop PC, GeForce 8800GTS GPU. GPRS (2G) wireless connection. Puget Sound Dataset. High quality. With panoramas. User motion at car speed (150 km/h).

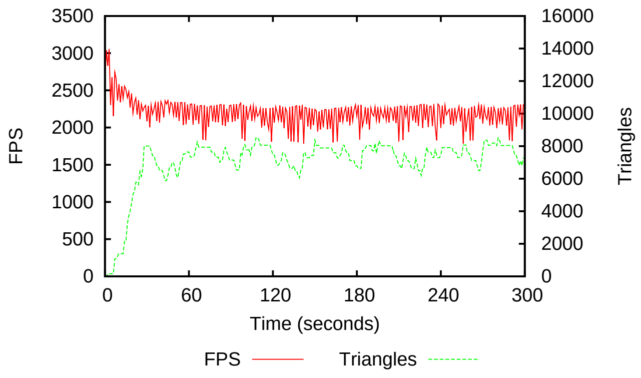


(a)

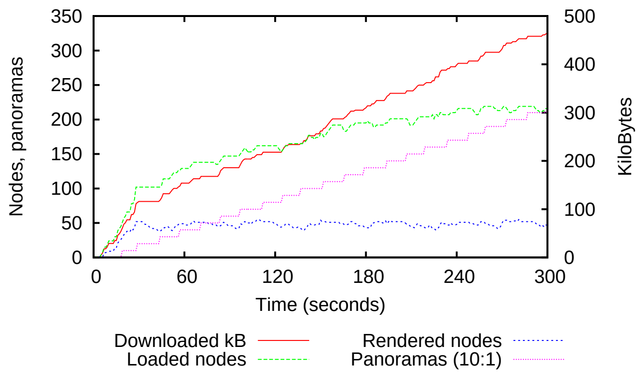


(b)

Figure 77: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 27. Desktop PC, GeForce 8800GTS GPU. GPRS (2G) wireless connection. Puget Sound Dataset. High quality. Without panoramas. User motion at car speed (150 km/h).

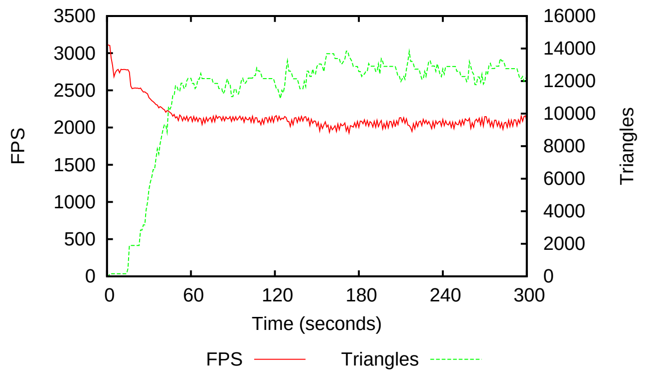


(a)

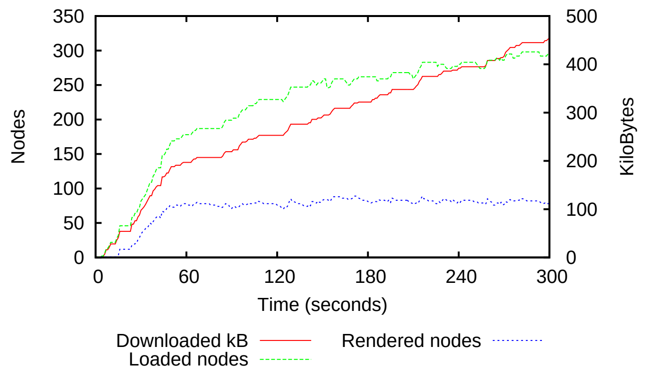


(b)

Figure 78: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 26. Desktop PC, GeForce 8800GTS GPU. GPRS (2G) wireless connection. Puget Sound Dataset. Low quality. With panoramas. User motion at car speed (150 km/h).

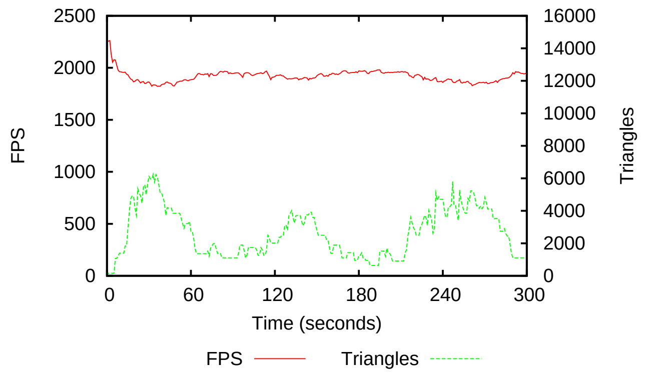


(a)

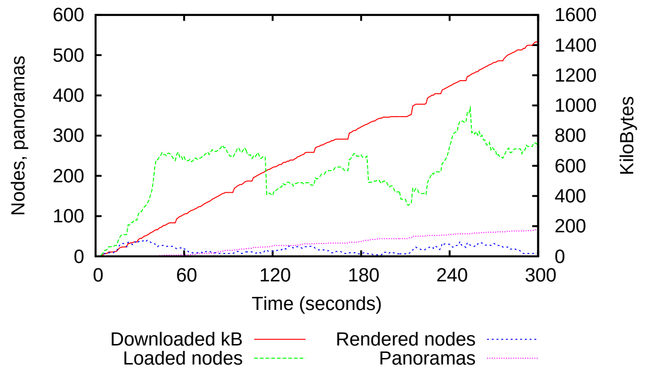


(b)

Figure 79: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 27. Desktop PC, GeForce 8800GTS GPU. GPRS (2G) wireless connection. Puget Sound Dataset. Low quality. Without panoramas. User motion at car speed (150 km/h).

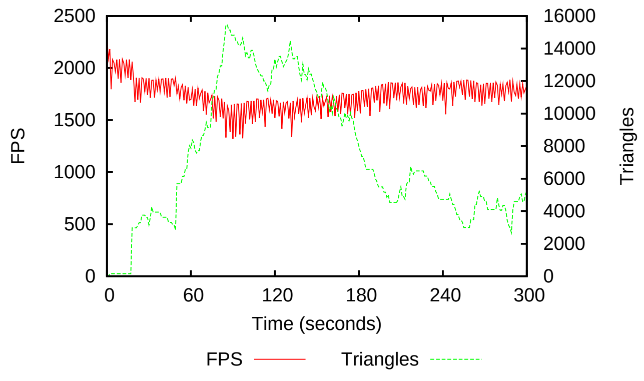


(a)

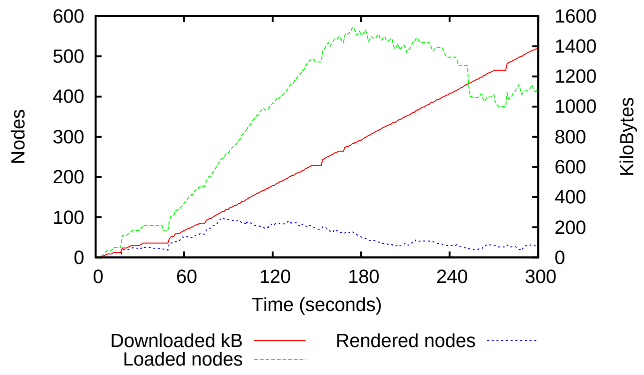


(b)

Figure 80: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 26. Desktop PC, GeForce 8800GTS GPU. GPRS (2G) wireless connection. Puget Sound Dataset. High quality. With panoramas. User motion at jet speed (750 km/h).

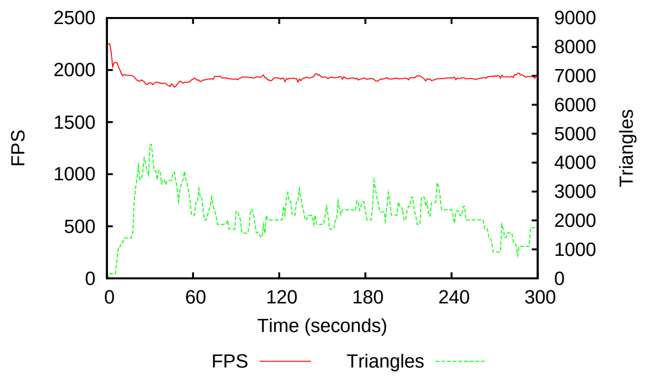


(a)

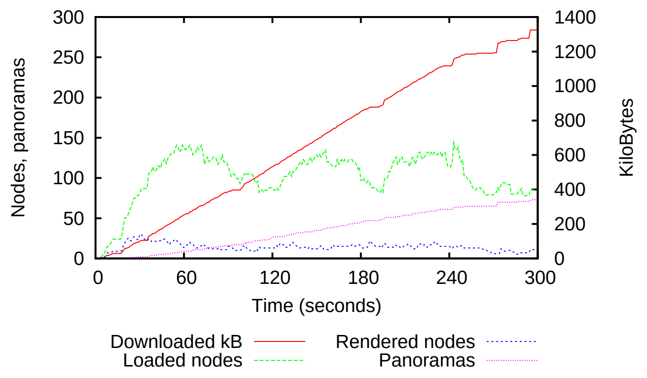


(b)

Figure 81: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 27. Desktop PC, GeForce 8800GTS GPU. GPRS (2G) wireless connection. Puget Sound Dataset. High quality. Without panoramas. User motion at jet speed (750 km/h).

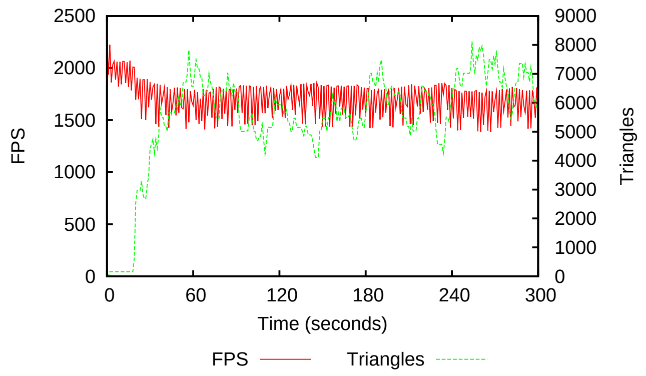


(a)

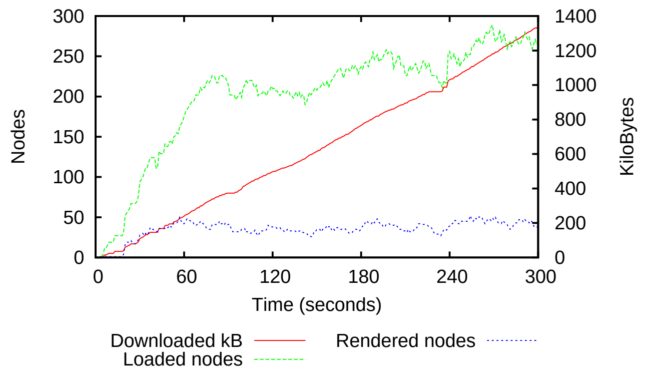


(b)

Figure 82: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 26. Desktop PC, GeForce 8800GTS GPU. GPRS (2G) wireless connection. Puget Sound Dataset. Low quality. With panoramas. User motion at jet speed (750 km/h).

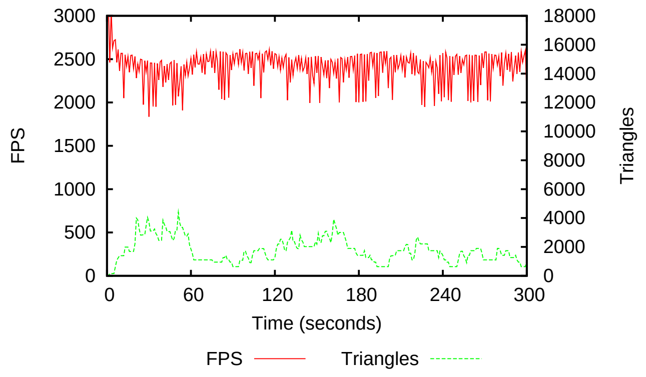


(a)

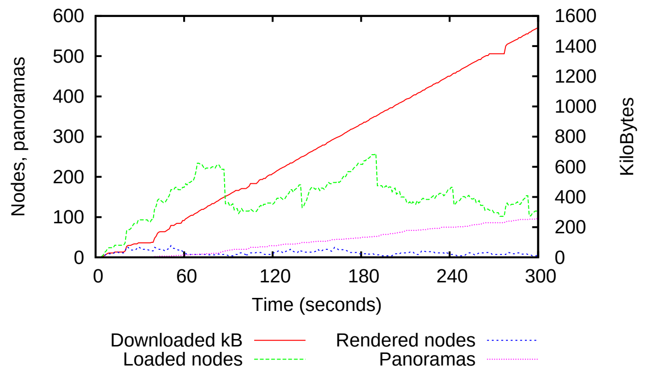


(b)

Figure 83: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 27. Desktop PC, GeForce 8800GTS GPU. GPRS (2G) wireless connection. Puget Sound Dataset. Low quality. Without panoramas. User motion at jet speed (750 km/h).

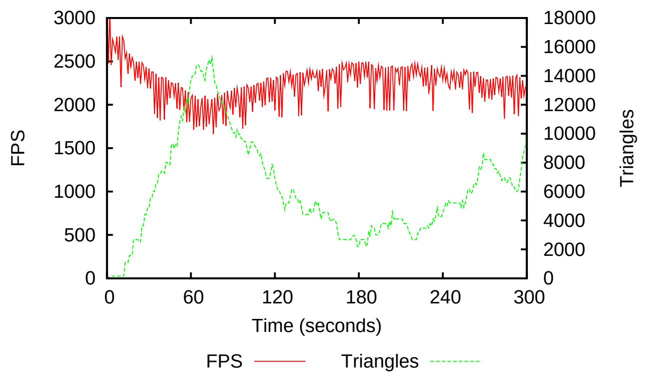


(a)

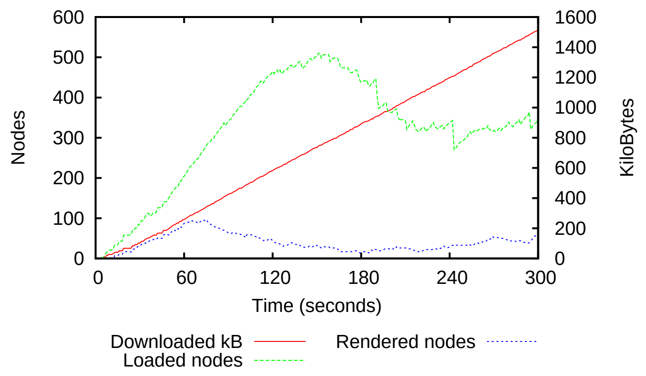


(b)

Figure 84: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 26. Desktop PC, GeForce 8800GTS GPU. GPRS (2G) wireless connection. Puget Sound Dataset. High quality. With panoramas. User motion at jet speed (1000 km/h).

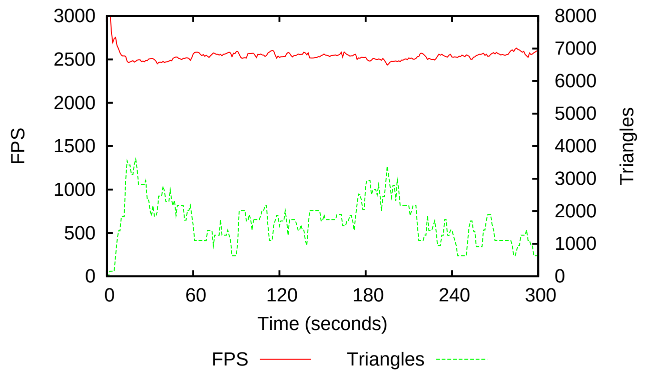


(a)

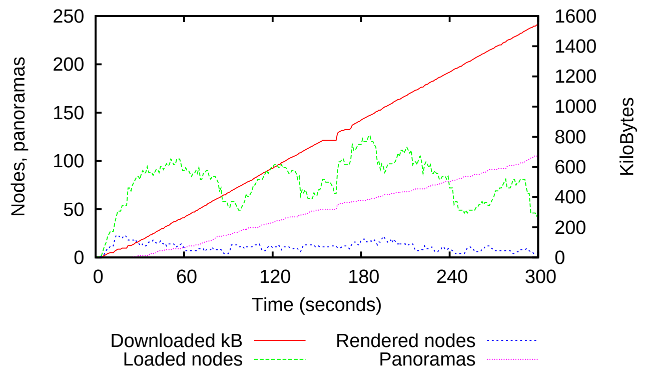


(b)

Figure 85: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 27. Desktop PC, GeForce 8800GTS GPU. GPRS (2G) wireless connection. Puget Sound Dataset. High quality. Without panoramas. User motion at jet speed (1000 km/h).

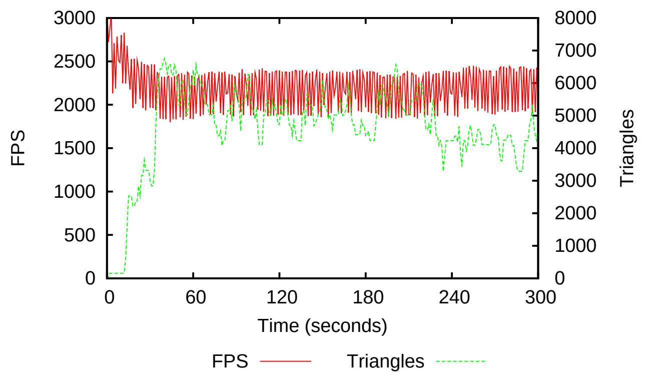


(a)

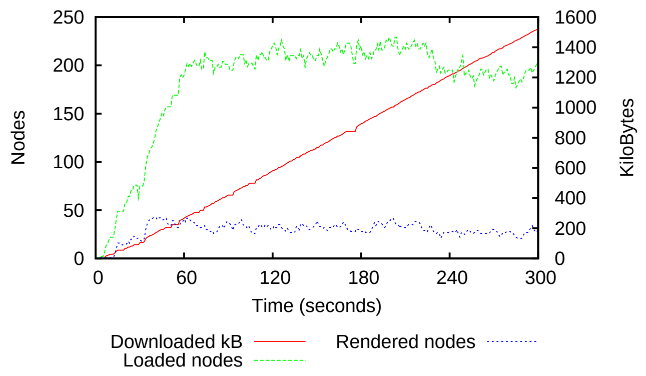


(b)

Figure 86: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 26. Desktop PC, GeForce 8800GTS GPU. GPRS (2G) wireless connection. Puget Sound Dataset. Low quality. With panoramas. User motion at jet speed (1000 km/h).



(a)

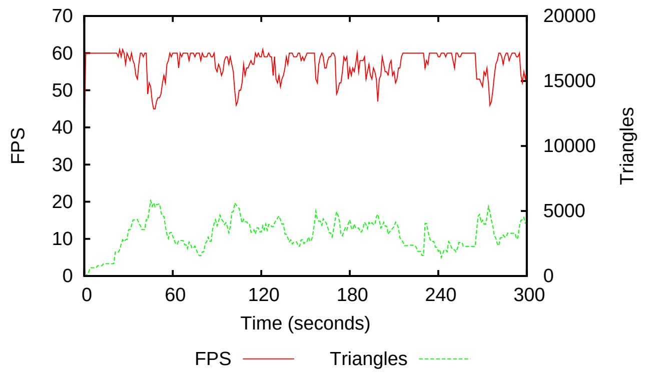


(b)

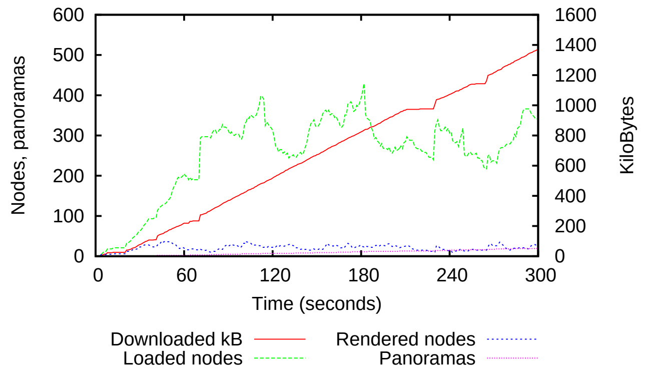
Figure 87: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 27. Desktop PC, GeForce 8800GTS GPU. GPRS (2G) wireless connection. Puget Sound Dataset. Low quality. Without panoramas. User motion at jet speed (1000 km/h).

C.2 Nokia N95 device

C.2.1 GPRS (2G) wireless network with flag to avoid network congestion due to panoramas



(a)



(b)

Figure 88: Perfomance measures for the walkthrough are similar to those that are despicted by Figure 27. Desktop PC, GeForce 8800GTS GPU. GPRS (2G) wireless connection. Puget Sound Dataset. Low quality. Without panoramas. User motion at jet speed (1000 km/h). A flag is used to avoid network congestion due to panoramas. Compare this Figure with Figure 50.