Evaluation of 3D applications on mobile gaming consoles using Client-Server architecture

Alejandro Onatra Caro
Multimedia Research Group
Universidad Militar Nueva Granada
Bogotá, Colombia

Wilson J. Sarmiento
Multimedia Research Group
Universidad Militar Nueva Granada
Bogotá, Colombia

Abstract

Within the mobile devices, the gaming consoles have the greatest graphic processing capabilities, in this category can be find the Nintendo DS and Sony PSP. Despite their resources there is minimal use of client-server architectures. This situation opens the possibility of testing this kind of platforms using client-server based applications. In this work was evaluated the performance and development possibilities of a gaming console using a client-server system that updates a 3D world in real time and displays it in the graphic terminal, with the geometric information transmitted by the server in response to the user actions, it is also presented a benchmark test to verify the performance capabilities of gaming consoles.

Keywords:: 3D World, Client-server systems, Mobile Gaming Console, Play Station Portable

1 Introduction

For many years desktop computers had been the main 3D development platforms in the world. But a few years ago that began to change with the development of mobile devices with greater graphic capacities [Chang and Ger 2002] and efficient libraries for 3D mobile development. This context encouraged science and industry to develop and research in different areas of IT based on mobile devices [Canali et al. 2009]. The main reasons for this are: mobility and connectivity[1], because unlike desktop computers [Cha et al. 2009], mobile devices allow users to carry them all day long without being an obstacle to their activities and they can be connected to different platforms, including other mobile devices or desktop-like platforms [Gui et al. 2009]. But only until 2007 mobile applications and libraries development began to growth in a representative rate [Tseng et al. 2007], both on gaming consoles and mobile phones in industry and science [Noguera et al. 2010]. With better tools on gaming devices, appeared new ways of developing application based on client-server architecture, allowing consoles to process more tasks in parallel because the server handle partially or totally the graphic processing.In order to implement a client-server architecture in mobile devices, two methods had been used, the first one render the scene on the server and streams it to the client [Gu and Xie 2010], the second one render the scene on the client using the information provided by the server [Lamberti and Sanna 2007].

This work evaluates the performance of a gaming platform using client-server architecture, in order to explore the possibles applications that can be develop in such systems. The evaluation process use a test application focusing the graphic processing on the gaming platform that acts as a client, allowing it to storage only the currently visualized scene, and the probably next scene. For this purpose we used the platform created by Sony Corporation, called *Play Station Portable*, this one has the highest graphic processing potential, and that is why it was chosen for the construction of the system proposed in this work. Unlike previous work the client had had low graphic processing power compared to the PSP(Play Station Portable), but in this project all the advantages given by the high graphic processing capabilities of the client were used.

2 Related Work

Nishino et al. develop an Ubiquitous 3D Graphics Modeler for Mobile Devices using the first of the two approaches mentioned in the section 1, that allows the user to display virtual models in real spaces on the screen, where the server renders the scene and send it frame by frame to the client [Nishino et al. 2008]. Lamberti and Sanna develop a system that sends the renderized scenes trough compress images to the client. The data processed is streamed to the client, and then it is decompressed by it [Lamberti and Sanna 2007]. One of the main restrictions of this method is the bandwidth to achieve at least 30 FPS(Frames per Second), that is the minimum frame rate recommended for the human eye to see continuous image visualization. If the bandwidth decreased it would be harder for the system to achieve 30 FPS.

Sterk and Palacio compared also two related methods, the first one render a scene on the server and streams it to the client, and the second one renders a fraction of the scene on the client [Moser and Weiskopf 2008], and use a impostor box or skybox to replace the landscape. Their main conclusion is that each method has its advantages and disadvantages, but they determine that the graphic properties of the mobile devices are decisive for choosing which method is the right one for the project. In their case, when the render was made in the client, the visualization was smoother, but maintaining a system like that, was more expensive and less scalable than the system rendering the scene in the server[Sterk and Palacio 2009].

Riley and Decker build a client-server system where a mobile client can received repetitions of a baseball game [Riley and Decker 2006]. This application doesn't stream images from the client to the server, instead the images were sent one by one and displayed when they were all loaded on the client. This kind of system its not real time based but show other possibilities that can be explored with the client-server architecture.

In the next section details of the system architecture are exposed, the section 4 presents a benchmark evaluation and the last section includes the conclusions of this work an recommendations.

3 System Architecture

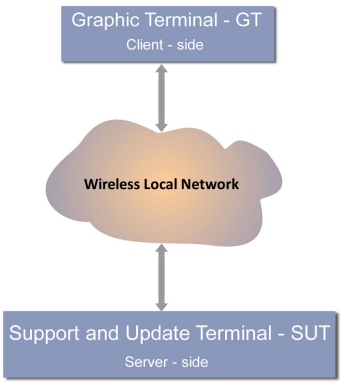

Figure 1: *Basic diagram of a Client-server architecture.*

In the figure 1 it is displayed a basic model of the client-server architecture used in this work, the client is named *Graphic Terminal* (GT), PSP in this case, its job is to process polygons and lighting. And the server is called *Support and Update Terminal* (SUT), this one has to send the positioning information requested by a client each time its needed. In the next subsections the morphology of the system, modules, functionalities and relevant technical data are shown and described.

[1] Wi-Fi, Bluetooth, IrDA, etc.

3.1 Multiple client to server design

The system proposed in this work use a client-server architecture, where the client acts as a GT. The graphic terminal process most part of the graphical data, storage the current scene and the possible next scene, render the current scene including polygons and lighting, track the user movements, request information from the server when is needed to display the next scene. The server acts as a SUT, its functions are: Send the 3D positioning data to the client when requested verifying its integrity when send or received, Storage the different worlds composing the application 3D Universe. The SUT can serve many clients at a time so it is very important to verify the data coherence. In the figure 10 it is displayed the system architecture.

The client-server communication is implemented trough *Sockets* technology using the protocol *TCP/IP*, so its important that the client knows the server IP direction to initialize the conversation. The information exchange between the client and the server is managed trough a simple custom protocol.

3.2 Scene structure

The *World W* used in this work is compose of scenes called *Box Q*, this boxes are spatially arrange like a $L_{world} * L_{world}$ size matrix where L_{world} is the size of the side of the square matrix. The number of boxes Q composing the world W is L^2_{world}. Each box Q is composed of divisions called *Square C* spatially arrange like a matrix of size $L_{box} * L_{box}$ where L_{box} is the side of the matrix size. Also has an associated integer *Matrix M* describing the type of visualization needed in the related square. Each square C has a size of l^2_{square} where l_{square} is side of the squares size measured in PSP virtual space units. In the figures 2 and 3 it is shown the geometric distribution of the scene.

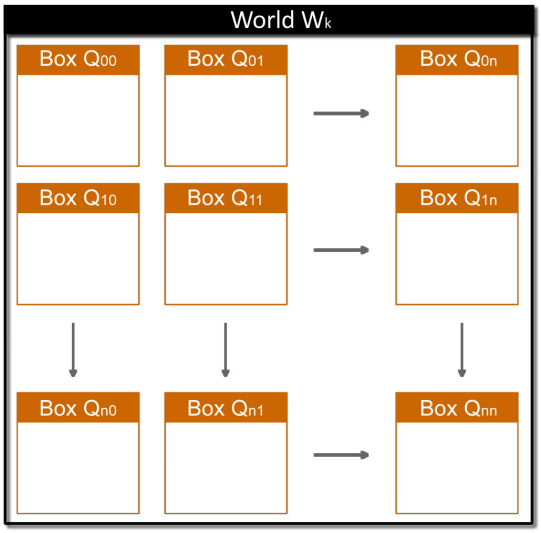

Figure 2: *Structure of a world W_k with $n * n$ boxes Q.*

Every square C has an associated integer value in the *visualization Matrix M* of the box Q, the value represents the way a model would be positioned on the square C. In this work there is only one type of model that can be visualized on the scene, it is a wall-shape model composed of 6 rectangular faces, and each rectangular face is composed by two triangles. This approach allows better visualization features. The visualizations modes are:

- Mode *0* , don't visualize the model on this square C.
- Mode *1* , visualize the model in this square C.

3.3 Support and Update Terminal -SUT

For the server implementation the Java programming language was used, mainly because of its multi-platform characteristics, it is based on objects and has a extensive standard API with many purpose utilities. In general the server must have the capacity to run Java applications and the next minimum features:

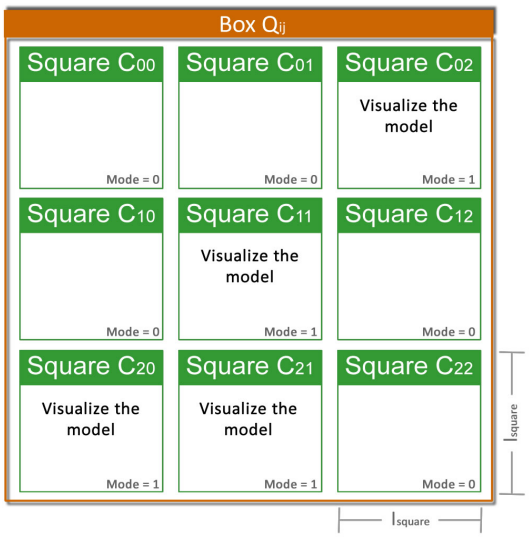

Figure 3: *Structure of a Box Q of size $3 * 3$, 4 squares C are in mode 0 and 4 in mode 1.*

- Processor Pentium IV of 1.5GHz.
- Memory RAM of 510MB.
- Hard Drive with 20GB of capacity.
- Network card with Wi-Fi IEEE 802.11b capacity.
- OS Windows XP, Linux Ubuntu 8.10 or similar, MAC OS X.

The server does the following tasks:

1. **Connect to the local Wi-Fi network**: The server connects to the local wireless network and allow access for multiple clients identified as PSP devices, trough the TCP/IP protocol and *Sockets*.
2. **Send the information requested**: Send specific information to the client when requested, using the specified requesting protocol GT-SUT.
3. **Storage the 3D worlds geometric information**: The geometric information of the worlds W structure is storage within the server, so that the client don't have to had all that information, allowing the system to change the worlds in real time, updating the client when this apply for new information.
4. **Verify the integrity of the data send and received**: Since the system can respond to multiple clients, the data integrity needs to be verified each time is sent or received, so that in case the data match with the server reference, this will send confirmation to continue to the client, if the data doesn't match, it would be necessary to request the data again.

The figure 4 shows the structure of the server.

3.4 The graphic terminal - GT

The graphic terminal chosen for this work was the Sony mobile gaming platform, PSP. This platform main chip is composed of a CPU based on MIPS32 Rk-4 of 32 bits with a speed of 333MHz, a FPU for operations on floating point numbers, a VFPU for vector processing, a DSP for signal processing and a special processor known as *Media Engine* also based on MIPS32 Rk-4 of 32 bits dedicated to audio and video processing trough hardware. It has a 32MB main memory capacity and 4MB of embedded DRAM. The graphic chip has a speed of 166MHz using 2MB of the 4Mb of DRAM and Bus of 512 bits, can process up to 33 millions of polygons with

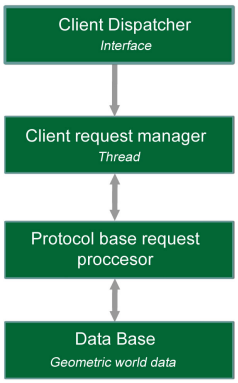

Figure 4: *Structure of the Server.*

transformations and lighting, with a rate of 664 Megapixels per second. Also has a sound chip with 3D capacity and y 7.1 audio channels, 4.3 inch Widescreen TFT LCD, Memory card reader *Memory Stick*, Supports Ad-HOC and AP connection trough Wireless LAN card IEEE 802.11b, Bluetooth and Mini USB 2.0. In the gaming mobile console world it has bigger processing power than any other and all basic multimedia equipment for a basic entertainment system. In the figure 5 it is shown the client structure.

The client does the following tasks:

1. **Connect to the local Wi-Fi network**: The GT connects to the local wireless network and let the user choose the server IP to begin communication and the data request.
2. **Request information of the scene**: When needed send a request to the server, for geometric information about the scene to be rendered using the protocol GT-SUT.
3. **Render the current scene**: With the information of the box Q obtained from the server, render and display the scene on the screen.
4. **Updates the current scene**: Evaluate the user movements using the tracking method and request the data from the next scene when needed.

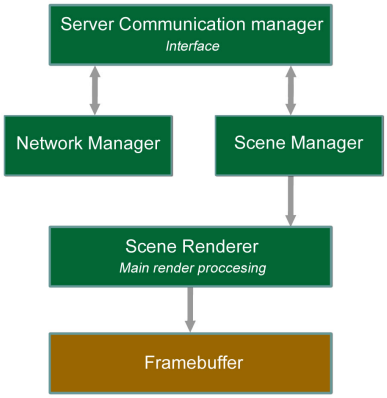

Figure 5: *Structure of the Client.*

For implement the client it was used the free SDK *Creative Commons*, *minPSPW*[2], created by Paulo Lopes. Also the Eclipse Galileo IDE, with the C++ programming language.

3.5 User position evaluation - Tracking Algorithm

To evaluate the user position inside a box $Q_{i,j}$ (where i,j are coordinates on (x,y) on a discrete space) currently rendered, a simple tracking algorithm was used. The box $Q_{i,j}$ will have maximum 8 neighbour boxes Q, but the corner boxes were not taken into account in this work, like can be seen in the figure 6 corners , so each box have maximum 4 neighbour boxes. The algorithm detects when the user is within a predefined $load\_distance$ distance from a border of the box $Q_{i,j}$ associated to the green area of the figure 6 of size $e * l_(square)$, and which border is. The border options are: $Q_{i+1,j}$, $Q_{i,j+1}$, $Q_{i-1,j}$ and $Q_{i,j-1}$. Then every time a user is within a distance $load\_distance$ from any border of $Q_{i,j}$, the client will request the information of the closest border to the user and loads it into memory, in case the user position is within a $load\_distance$ distance of a scene, the latter will be rendered by the system with the advantage of being previously loaded in the PSP memory. In the figure 6 there's a scheme of the geometry taken account for the tracking algorithm.

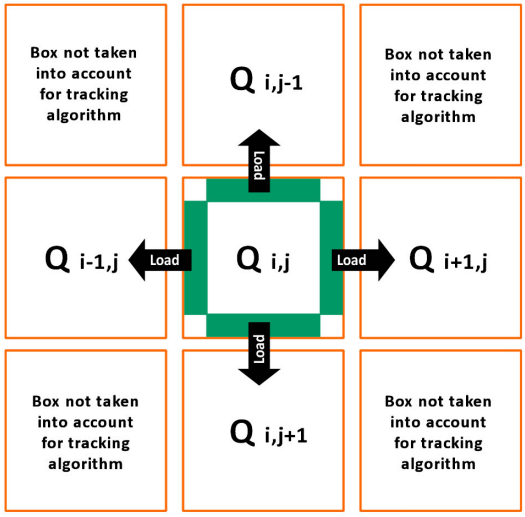

Figure 6: *A box Q neighbourhood showing loading area and paths that can be taken to go to another box.*

The algorithm 1 or *Tracking Algorithm* evaluates the position of the user frame per frame, calculating the minimum distance $distance_{xy}$ between the user position and the border of the 4 possible borders $border_i$. When the distance is less than $load\_distance$, the system make a request for parameters to the server, and load the data of the Box Q associated to that border, if the loading process ends up successfully, the $load$ variable is set to $true$ if not, to $false$. If the user position is within $min\_distance$ of a border and the Box Q data associated to this is loaded, the system renders this info, and show it to the user.

Algorithm 1 Tracking algorithm

Require: $load\_distance >> min\_distance$
  for $i = 0$ to 3 **do**
    $distance_{xy} = minDistance(position, border_i)$
    if $distance_{xy} < load\_distance$ **then**
      if $load == false$ **then**
        $load = LoadScene(border_i)$
      end if
    else
      $load = UnloadScene()$
    end if
    if $distance_{xy} < min\_distance$ **and** $load == true$ **then**
      $RenderScene()$
    end if
  end for

3.6 Protocol GT-SUT for requesting information

The communication between GT and SUT its based on the exchange of messages in ASCII coded strings. This strings or messages follow the next rules:

- Every platform that wants to connect to the server(in this case

[2]http://minpspw.sourceforge.net/

the PSP)has to be identified with an @ followed by the system name(i.e @PSP).

- Every string should end with a closing character, in this case # was chosen.
- Any exchange of information should be followed by a confirmation from the receiver, that confirmation should begin with the character *C*.

To verify that the strings are well constructed, a simple protocol was designed. The protocol is divided in three parts, the first one find the type of device trying to connect and what kind of information is requested. The second one is focused on the information management, in the figure 9 after the Device-Action-Request sequence, the server reach a *Send World Parameter* state trough the *W#* string, this state keep waiting for a *Wn* string where the W means its a world parameter and n is the number of the parameter, in this case number 0 is L_{world} and 1 l_{box}. After these, the server send the string *CWn#* to confirm the end of the world parameters and go to the *Waiting Request* state. When the client send *B#* to the server it change its state to *Send Box Parameters* this state waits for a *Bn#* string, where the B means its a box parameter and n is the number of the parameter, in this case 0 is l_{square} and 1 is the matrix M, in this case the server keeps sending the information of the matrix one item at a time, all the items are confirmed by the server. If any parameter is unconfirmed, the box o world process is started again. The last section of the protocol is shutting off the connection, in the figure 9 is represented this state as *Terminate Protocol*.

3.7 3D World navigation using a Client-server system

The system proposed in this work is composed of a client *GT* that connects to a server *SUT* using a wireless network and an access point. To display the 3D world on the PSP screen, the *GT* request the geometric information of the current scene to the *SUT* using *Sockets*, this request include: L_{world},l_{box},l_{square} and the matrix M of the box Q. The user can move in the 3D world linearly or angularly in x, y or z directions.

If the user approach to one of the borders of the box Q, the *GT* request to the server the information of the box Q next to the specified border. The tracking algorithm takes into account only one border at a time,that's why the corners where not included in the transition of the boxes. If another client *GT* exist, the *SUT* would create another thread to attend the requests of the new client. The structure of the system is displayed in the figure 10.

4 System Benchmark Tests

In this section its shown the different kind of tests run on the system and the different results obtained trough the different evaluation criteria.

4.1 Performance tests

The performance test are based in *Frames per Second* in different scenarios and the seconds needed to load the basic chain of data needed to render a box Q. The tests were:

1. **FPS vs Size of the box**: This test compare the relation between average FPS and the size a box Q. Given that not all the models are shown in a random scene, the matrix M was modified allowing that all the models could be displayed, so the box Q associated to that matrix will hold the maximum number of polygon possible given our current model.

   The maximum number of polygons N_{poly} in this type of boxes is given by the formula:

   $$N_{poly} = 2 * l_{box}^2 * (n_{face} + 1) \quad (1)$$

   Where n_{face} its the number of faces per rendered model, but given that there's only one kind of model in this tests, there's only one formula needed to calculate this quantity. In this test only a value of 30 FPS or higher its an acceptable score. The number two in the beginning of the formula is placed because each square face is composed of two triangles.

2. **Scene time loading vs Size of the box**: This test compared the time(Measured in second) the client take to load the data needed to render a scene in terms of the size of the box Q.

   The size of the data N_{data} sent from the server is l_{box}^2, because only the information that comes from the matrix M its taken into account.

This paper doesn't show the tests using the size of the square C because in earlier stages of the project it was shown that for all sizes needed for this project or higher, doesn't affect the FPS.

4.2 FPS vs Size of the box

For each size of the box there were taken 50 measures of the frame rate. In the table 1 it is shown the data incrementing each time by 5 the size of the box, in the figure 8 it was used an increment of 1. Each FPS score was approximated to the closest integer value:

l_{box}	FPS	N_{poly}
5	60	350
10	60	1400
15	60	3150
20	60	5600
25	30	8750
30	30	12600
35	20	17150
40	15	22400
45	12	28350

Table 1: *FPS vs Size of the box.*

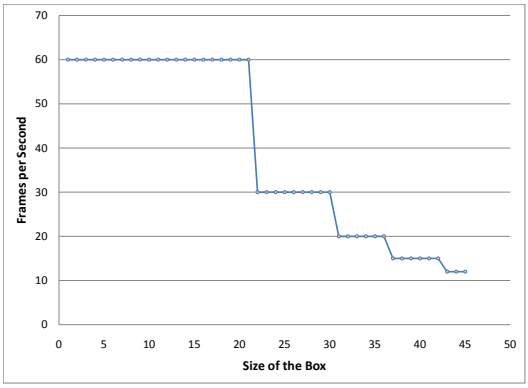

Figure 7: *FPS vs Size of the box*

In the table 1 and in the figure 7, the FPS decrease below the acceptable rate when the size of the box Q reach a value of 31, so, in l_{box} = 21 the scene has an optimum size. It can be extended a little further and use a value of l_{box} = 30 with a frame ratio of 30 that's in the limit of what is visible accepted for image continuity by the human eye.

4.3 Scene time loading vs Size of the box

For each size of the box there were taken 50 measures of the time t_{load} used to load the information of the scene. In the table 2 it is shown the data, incrementing each time by 5 the size of the box, in the figure 7 it was used an increment of 1. Each time the score was approximated to the closest integer value:

l_{box}	N_{data}	t_{load}
5	25	3.1s
10	100	5.4s
15	225	12.3s
20	400	15.9s
25	625	18.3s
30	900	22.8s
35	1225	30.2s
40	1600	41.2s
45	2025	52.4s

Table 2: *Scene time loading vs Size of the box.*

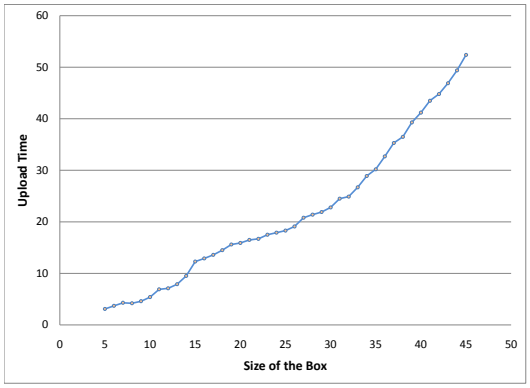

Figure 8: *Scene time loading vs Size of the box.*

In the table 2 and in the figure 8, can be seen a regular increment of the loading times when the size l_{box} of the box is raised. So in the loading times t_{load} between 16.5 and 22.8 seconds the FPS associated is between 60 and 30. These values are indeed a long time to wait for the data to load, but that's why the system used a tracking algorithm, minimizing waiting time.

5 Conclusions and Recommendations

The architecture used in this project was focused to multiple clients and one server to attend to all the requests in the local wireless network. In future work it would be convenient to think of world models with higher complexity, so that different users, navigating in their personalized worlds can interact with other users or their worlds. Such system would need to manage the communication trough the server or using P2P connections, where the mobile platform acts as a server and a client at the same time. In the current system the geometry data its storage in the server, that makes easy the administration of this information, so to manage two worlds coming from two different clients interacting between them, would be better to focused the communication control on the server.

In this project the mobile platform used shows a solid performance using client-server architecture, processing 12600 triangles with a FPS ratio of 30 without complex optimizations. If the code was modified to maximize the processing capacities of the platform, it will provide better results. In the future, is advise using advance predictive algorithms to expect the user movements and load the information of the possibly next scene in parallel. Also, its possible to reduce the time between requests, increasing the value of the square size, so that the user would need more time to approach to the borders of the scene and the predictive algorithm will have more time to calculate the user trajectory without the frame rate being affected. The PSP performance trough the different tests using the evaluation application shows the great potential possessed by this type of platforms when using server-client architectures.

The use of mobile gaming platforms for 3D applications development using client-server is a challenge, because their are closed system, that means that all the official development tools like the device itself are proprietary, so all third-party tools have a lot of technical problems in the documentation above all. But if this is overcome, this type of consoles have a great potential to offer, either the PSP for its graphic processing capabilities or others for their different gadgets(Touch screens, Autostereographic Displays, Built-in cameras, Built-in microphones, facial recognition, etc.) to take the interaction to new levels.

References

CANALI, C., COLAJANNI, M., AND LANCELLOTTI, R. 2009. Performance evolution of mobile web-based services. *IEEE Internet Computing March/April*, 60.

CHA, S., KURZ, J. B., AND DU, W. 2009. Toward a unified framework for mobile applications. In *Seventh Annual Communications Networks and Services Research Conference*.

CHANG, C.-F., AND GER, S.-H. 2002. Enhancing 3d graphics on mobile devices by image based rendering. In *PCM 02: Proceedings of the Third IEEE Pacific Rim Conference on Multimedia*.

GU, Y., AND XIE, M. 2010. A efficient architecture for semi-real-time graphical simulation based on mobile computing devices in wireless wms system. In *Asia-Pacific Conference on Wearable Computing Systems*.

GUI, F., GUILLEN, M., RISHE, N., BARRETO, A., ANDRIAN, J., AND ADJOUADI, M. 2009. A client-server architecture for context-aware search application. In *International Conference on Network-Based Information Systems*.

LAMBERTI, F., AND SANNA, A. 2007. A streaming-based solution for remote visualization of 3d graphics on mobile devices. *IEEE Transactions on Visualization and Computer Graphics 13*, 2.

MOSER, M., AND WEISKOPF, D. 2008. Interactive volume rendering on mobile devices. In *VMV 2008 - 13th International Fall Workshop on Vision, Modelling and Visualization*.

NISHINO, H., SHIIHARA, K., KAGAWA, T., AND UTSUMIYA, K. 2008. A ubiquitous 3d graphics modeler for mobile devices. In *International Symposium on Parallel and Distributed Processing with Applications*.

NOGUERA, J. M., SEGURA, R. J., OGAYAR, C. J., AND ARINYO, R. J. 2010. A hybrid rendering technique to navigate in large terrains using mobile devices. In *Computer Graphics International. Singapore: 2010.*

RILEY, P. F., AND DECKER, J. C. 2006. Analysis architecture of a mobile sports replay system. In *Proceedings of the 20th International Conference on Advanced Information Networking and Applications (AINA 06).*

STERK, M., AND PALACIO, M. A. C. 2009. Virtual globe on the android, remote vs. local rendering. In *Sixth International Conference on Information Technology: New Generations.*

TSENG, Y.-M., WU, T.-Y., AND WU, J.-D. 2007. A mutual authentication and key exchange scheme from bilinear pairings for low power computing devices. In *31st Annual International Computer Software and Applications Conference(COMPSAC 2007).*

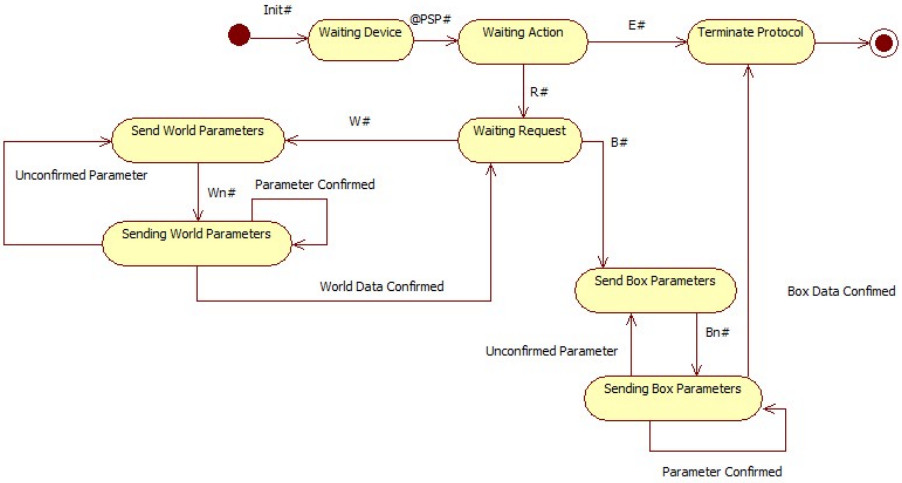

Figure 9: *The server-side protocol state diagram.*

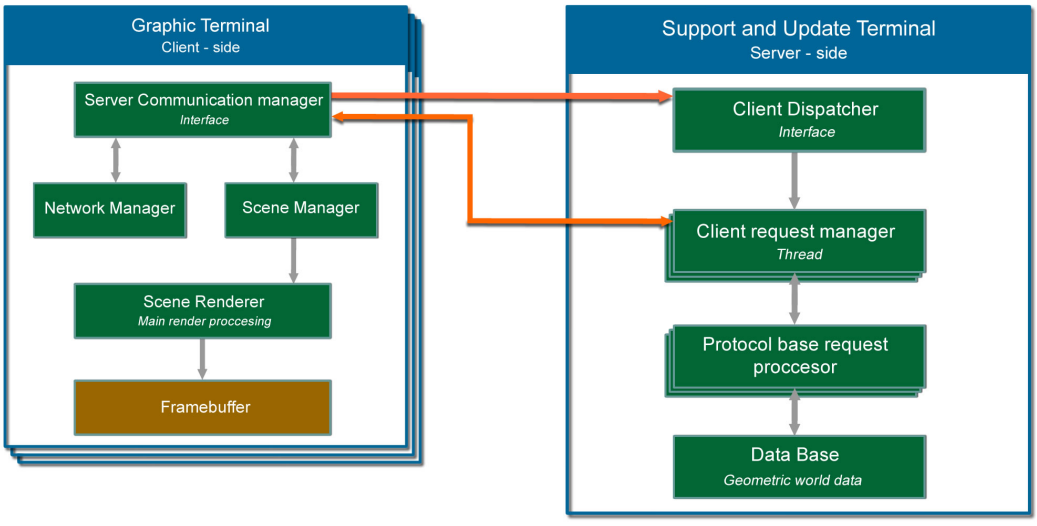

Figure 10: *The proposed system scheme: 3D World navigation using a Client-server system.*