MPEG-4-based adaptive remote rendering for video games

Nicolas Tizon* Télécom SudParis
Christina Moreno† Télécom SudParis
Mihai Cernea‡ Télécom SudParis
Marius Preda§ Télécom SudParis

Abstract

This paper proposes a complete framework for remote rendering of 3D and complex 2D applications. The retained approach is based on standardized video encoding and interactivity streaming methods and was developed within the framework of the Kusanagi project [1]. The complete architecture of the gaming platform is described and especially we have developed an advanced "lobby server" which acts as an interface between the users and the applications and manages the resources sharing. This server component is also designed in order to build a strongly pervasive environment allowing users to run their 3D applications everywhere and on every device. In this perspective, the MPEG-4 format support is one of the main features that permits to diffuse easily and rapidly our solution. In addition, our gaming platform has been tested by a panel of beta testers. The users experience feedbacks are summarized and give relevant information for the future works. To improve the adaptability of our system, an optimized encoding an streaming method is proposed and the simulation results show that we can significantly improve the user QoE without adding complexity on the server side.

CR Categories: I.3.2 [Computer Graphics]: Graphics Systems—Remote systems;

Keywords: 3D graphics, streaming, MPEG-4, interactivity, multi-user

1 Introduction

Taking off on the early 80s (preceded by the adoption acceleration of personal computers), the industry of video games is one of the most profitable and stable industries in the entertainment market. In 2002, for the first time, the videogame revenue exceeded the one corresponding to movie theaters entries in the United States. Since that date, there has been an increased interest in the study of this industry. Videogames are not longer considered as minor sector having just passionate gamers as target. Even if the console segment represents the most important economic segment, the Internet access development and the digital convergence change gradually the entertainment consumption [De Prato et al. 2010]. In this context cloud gaming (GaaS Gaming as a Service) offers new possibilities integrating hardware, software and service. This paper is organized as follows. The first section is an overview of the different existing remote rendering solutions. Then, we describe the Kusanagi framework and detail the role of the different functional modules. In the last section, we present our experimental works and we provide an adaptive approach to fit with various network constraints. Finally we conclude and provide perspectives for the future works.

*e-mail: nicolas.tizon@it-sudparis.eu
†e-mail: christina.moreno@it-sudparis.eu
‡e-mail: mihai.cernea@it-sudparis.eu
§e-mail: marius.preda@it-sudparis.eu
[1]http://www.kusanagi.eu/

I3D 2011, Paris, France, June 20 – 22, 2011.
© 2011 ACM 978-1-4503-0774-1/11/0006 $10.00

2 Remote rendering

Various remote rendering solutions already exist, but a few of them are well suited to the high requirements of 3D gaming or Ultra-High Definition visualization: latency is a major problem in both cases and requires specific optimizations of the whole chain, from the rendering server to the display device. These solutions pursue the "commercial-off-the-shelf" (COTS) hardware instead of dedicated hardware. In the same way, in order to build strong pervasive environments one must pay a particular attention to provide standardized and multi-platform solutions.

In the scope of remote rendering techniques, the literature is very rich and a large number of previous works could be mentioned. With respect to the nature of transmitted data, one can distinguish between six main families of solutions: Graphics commands, 2D Pixels, 2D Primitives, 3D Vectors, Single 3D object, Multiple 3D objects.

In the first case, the graphics commands which are exchanged between the application and the graphics card on the server are intercepted and sent to the client [Humphreys et al. 2001; Nave et al. 2008]. Then, the rendering is performed by the client GPU. In Humphrey et al. [2001] the solution was originally proposed to distribute rendering tasks over a cluster of computers. The application needs to be parallelized in a first time and the rendered data needs to be centralized in the last time, in order to be displayed. This last operation, from cluster to display, is the sort of remote rendering we are targeting in our project. It allows visualizing 3D contents on a light terminal. In Nave [2008], a graphic commands solution is proposed to reach satisfactory performances in terms of latency and to serve many users without consuming too much resources on the server. The main drawback of such a solution is the requirement of a powerful GPU on the client side limiting the advantages of executing the application remotely.

On the opposite, when streaming 2D pixels, all the rendering tasks are performed on the server and the output images are sent to the client. Different approaches have been considered to encode these output data. In Lamberti [2003] and Humphreys [2008] the 2D rendered pixels are sent through a video stream and in Aranha [2007] they are coded as still images. In Simoens [2008], an hybrid approach in which the 2D pixels are sent by using the X11 protocol for almost static contents and through an h.264 video stream when the motion increases, is proposed. Thanks to a switching mechanism the solution allows minimizing the latency for highly animated contents while saving CPU resources in the others cases.

In Diepstraten [2004], an alternative solution is presented. The rendering process is split between the client and the server by transferring couple of 2D line primitives over the network. Feature extraction algorithms are used to calculate the 2D vectors which are sent to the client and then rendered. Hence, this solution can be used on devices that do not support 3D graphics. In Quillet et al. [2006], a the authors propose a remote visualization system where 3D vectors are extracted and sent per object and an object is sent only once.

In the literature, there is little work addressing error-resilient coding of meshes. A solution proposed by Yan et al. [2005] is to divide the 3-D mesh to be transmitted in small pieces of uniform size that are coded and protected individually. On the decoder side, the pieces that cannot be recovered are replaced using an error concealment technique. In Al-Regib et al. [2005], a more advanced scalable representation of the objects is used allowing an optimized bit allocation for the transmission. In the scope of urban navigation, an optimized progressive 3D model download scheme for mobile devices is presented in Nurminen [2007]. In this case the amount of data is substantial and the 3D objects are transferred one by one. The transmission order depends on the user position.

Finally, in Jurgelionis et al. [2009] and Nijdam [2010] mixed approaches are proposed to build efficient multi-environment remote rendering platforms, the latency being restricted below a certain level. In the Kusangi project, the "2D pixel" (video streaming) approach has been chosen for its ability to optimize the user experience in a majority of cases, especially when facing poor degraded transmission conditions and considering an heterogeneous set of terminal devices.

3 The Kusanagi framework

Kusanagi project is an end-to-end seamless infrastructure to develop, publish and play High-Definition 2D and 3D Real-Time Networked Multimedia Content. The project focus on 2D and 3D interactive content required for both gaming and professional applications. In both domains, image quality and concealed delays are critical success factors, for which the Kusanagi project will research to maximize the output on high bandwidth and low latency networks. Today streaming is mainly focused on video-like images and does not take into account network latencies. In order to achieve its ambitious vision, Kusanagi project aims improving the streaming technologies by taking into account real-time interaction. An overview of the Kusanagi framework is given in Fig 1. The three main components of this architecture are: the Kusanagi application, the lobby server and the MPEG compliant client. The Kusanagi application is the new application obtained after integrating the audio/video capture and the Kusanagi plug-in with the original graphic application.

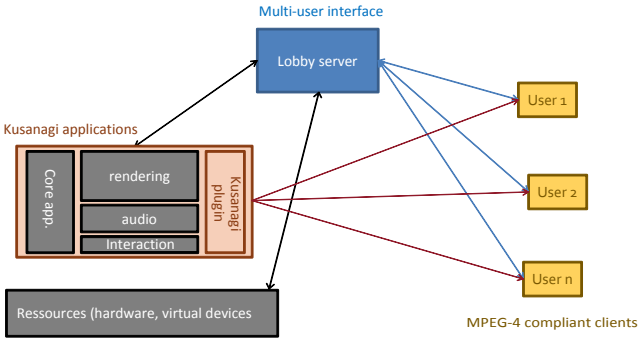


Figure 1: *Kusanagi framework.*

3.1 The Kusanagi plug-in

The Kusanagi plug-in is a piece of software that allows turning any 3D graphic application into a remote controlled software. This transformation requires being able to stream two different kind of data:

- interactivity commands from client to server,
- audio and video data from server to client.

Hence a net-input HTTP server is used to transmit clients' commands to the game. The port number is negotiated by the lobby server and there is no interaction with the lobby server during the session (transport). Concerning the audio and video data, a library has been developed and can be used very easily thanks to a very basic API (launch(),write(),close()).

3.2 The lobby server

The purpose of the Kusanagi project is to have end-user devices being able to "run" applications that would normally require a high-performance computer to run. This simulation is accomplished by running the desired application on a powerful machine, but letting the client be able to control the application by sending its inputs and receiving the video output in real time on its display. If this were to be done on a one-on-one relation it would be almost the same as sharing control of the powerful machine with the client, but if more clients need to use different applications or different instances of the same application at the same time on the same machine we would need a tool to manage each client. As described in Fig 2, the Kusanagi lobby does exactly this: it establishes connections with each client and ensures that each client is only sending its inputs only to its own application and each application video output is sent only to the corresponding client. This way a client is able to control his application and his application only even if several clients are working on the same server at the same time. In order to do this, unique ports are used for each resource for each client connection and the lobby is responsible for managing available ports, attribute them to each application and inform the client witch ports to use for each task. Like the name of this application implies, it is intended to

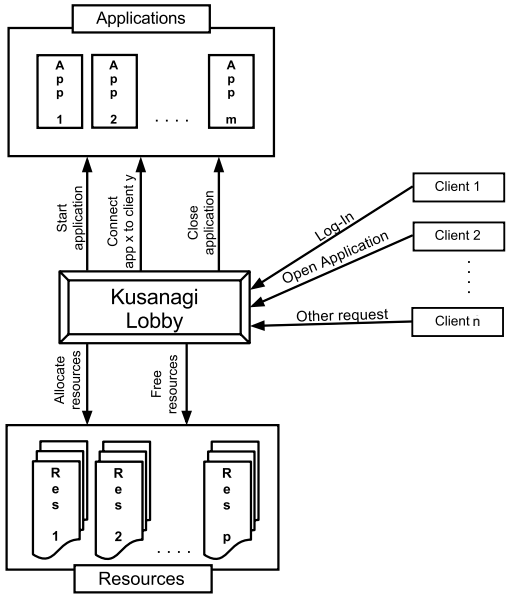


Figure 2: *Lobby server principle.*

be a "lobby": a place where all clients should have their first contact with the server. For this reason one of the responsibilities of the lobby is to monitor all incoming traffic, interpret the HTTP requests and if the request is valid take the necessary steps and reply with the corresponded answer. In order for this to work only one port is monitored and this is why a port should always be mentioned when the application is started or the default port (which we selected as 5114) will be monitored.

Kusanagi lobby is much more than a simple gateway, it is actually the resource manager entity of the Kusanagi platform since other responsibilities of the lobby include: client login, start the clients' desired applications, limit the number of simultaneous connections, create a "link" between the client and its desired application, constantly checking the state of current connections and finally closing the connections. In order to perform all these tasks efficiently the lobby should be launched with the following information: server port, maximum number of clients, stream folder and applications list options customized to fit the desired utilization. We will further try to explain how the lobby achieves its management goals.

As it is normal for this kind of client-server relationship the first step of the communication is the login. The client will send a login request and details regarding the user-name and password, the lobby will try to find the username in its database and if it is found and the password for that user proved to be valid, the client is informed that he can continue the communications with the lobby. In any others cases the lobby will inform the client that the information provided could not be validated in the database.

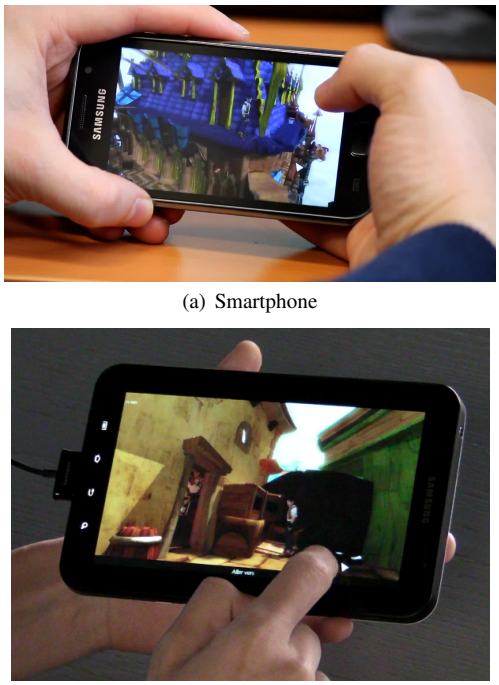
A next natural step after the login, is the request from the client to start the desired application. If the request is valid, the lobby starts working. First it has to check if the maximum number of clients allowed for this instance of the lobby had been reached. A limitation is needed in order to prevent too many applications to run at the same time. Overloading the server will cause reducing the user satisfaction either by having low frame rates, slow response time or even disconnections due to crashes. The maximum number of clients should vary in accordance with the servers computing power and applications demands for computing power. When the maximum number of clients has been reached, the lobby informs the client and shuts down the connection. When, the connection is closed the ports that were used are inserted back into available list and will be able to be used for further connections. If a new connection is allowed the server should prepare the channels through which the client should interact with its application and its application only.

After the channel between the client and its application has been set and launched, the lobby only has to check from time to time that the application and client are still running and the channel is alive. If for any reason, the application does not run anymore, the channel is released. The client is informed and it is removed from the lobby so a new client can connect. In the same way, if for any reason the client does not give a signal that it is still running, the channel is released and the application is closed (to release computing power).

The client can also ask the lobby to close the connection, for any given reason. This could happen if the client wants to close or when the user leaves the client for a long period of time (and marked as "away").

3.3 The MPEG-4 client

At client side, a full multimedia player with networking, media, user interactions management and rasterizing is used [Le Feuvre et al. 2007]. The MPEG-4 scene description language allows managing an interactive multimedia session and to support any kind of graphic applications. The client supports also a wide range of video and audio codecs (H.264, AAC,...) and is a complete network client. Especially, it supports RTP/RTSP streaming including RTP/UDP streaming and RTP over RTSP(TCP). In addition, this MPEG-4 compliant client is already available on different platforms (Windows, Linux, MacOS, Android) as described if Fig 3 and should be available on any devices which is assumed to be compatible with MPEG standards.



(a) Smartphone

(b) Tablet

Figure 3: *Examples of the multi-platform client.*

In the next section, some user experience feedbacks are presented and commented. Then, an adaptation algorithm is proposed in order to improve the user QoE.

4 Experimentation and optimizations

4.1 User experience

In order to test the Kusanagi gaming platform, a Beta test was performed from the first of February until the 4th of March in the THD platform [2]. The game used to test the remote rendering was a point and click adventure game: "Jack Keane". The recruitment was performed by the THD organizers and was composed by 53 test users from different ages, mostly living in the Parisian region.

The users were invited to play to the game Jack Keane and answer a short questionnaire about the experience. The users had also access to a forum where they can post the problems encountered. Once identified, the Kusanagi staff corrected them and prevented the users. The most significant error encountered deal with the saving mode.

A total of 37 questionnaires were usable, that is a response rate of 69.81%. In an average of 80% users did not found problems during the installation and starting with the game. Concerning the experience, the opinion is split, half of the users had a good experience without any flaws and the other one complain about some pixelation of the image as well as some latency. This problem can be explained by the type of connection used, the gaming time and by a lack of adaptability in the system. The lack of sound seamed not bother the gamers in the direct question, nevertheless almost of the users mentioned it in the commentary section.

Concerning the Attitude regarding this type of rendring solution,

[2] http://www.portailthd.fr

the questions were adapted from the TRA (Theory of Reasoned Action, Fishbein and Ajen). The results indicate that in a 92% users have a positive attitude towards Kusanagi and are willing to adopt it once it will be available. Among the reasons given to use a platform like Kusanagi the portability and cost stand out, followed by the enlargement of the market target. The only condition for this is an appropriate bandwidth. The platforms that are most likely to be used by this kind of service would be the Television and the Mobile Phone. Given the small number of usable questionnaires, the results are not conclusive but give us a glimpse of the direction and focus to make.

In order to achieve an acceptable quality of experiment (QoE) for each user, various resources management functionalities were implemented in the lobby server as previously described in 3.2. Moreover, to provide the best quality to each user in function of its reception conditions, a more refined mechanism must be implemented. In the next section, we describe in which conditions and how, the measured network latency can be used to adapt the streamed video bitrate

4.2 Latency measurement and adaptation

Basically, when measuring the latency in the Internet, one can distinguish between two different components. The first one is the intrinsic delay introduced by the network infrastructure (routers, transport channel...). For a given point-to-point communication, this delay is quite constant and cannot be easily attenuated by acting outside of the network. On the other hand, an important contribution to the latency can be caused by congestion. To limit this latency, some solutions can be implemented directly in the server which transmits the data and they can be very efficient, especially when the congestion is due to a saturation of the transport channel caused by the server itself. For example, one can experiment this latency when trying to watch a streamed video at a bitrate higher than the download bandwidth. Next, we assume that the congestion based latency is due to an over-estimation of the video bitrate compared with the reception bandwidth of the user.

A common way to estimate the latency is to measure the round trip time between the server and the client, by sending a ping command for instance, from one machine to the other. In the sequel, let's denote RTT this parameter. By calling RTT_i the intrinsic delay and RTT_c the congestion based component, we can write:

$$RTT(t) = RTT_i + RTT_c(t), \tag{1}$$

where RTT and RTT_c are time dependent variables.

For each user, the RTT_i variable can be easily estimated by the lobby server by sending a ping command at the beginning of the session. Then during the session if the video bitrate is too high, the RTT_c component will increase due to congestion. In our platform, RTCP-RR (Receiver Report) are sent periodically ($\sim$ every seconds) from the client to the server. Thanks to this information, the server is able to compute the round trip time $RTT(t)$. Moreover, thanks to the intrinsic delay RTT_i measured by the lobby, the server can deduce the congestion based latency $RTT_c(t)$ and adapt the video bitrate in order to reduce this delay.

During the encoding process, the more efficient way to decrease the bitrate consists in increasing the quantization step Q_p. Hence, when receiving an RTCP-RR and given the estimated round trip time RTT, the server adapts the quantization step value as follows:

$$Q_p^n = Q_p^{n-1}(1 + \alpha X^\beta), \tag{2}$$

where Q_p^n is the new quantization step, Q_p^{n-1} is the old value, α and β are positive constant variables experimentally fixed in the next section and X is given by:

$$X = (RTT - RTT_i)/RTT \tag{3}$$

These heuristic computing allows decreasing the quantization step when a congestion based latency is detected i.e. when $RTT_c = RTT - RTT_i > 0$. Hence, when the estimated RTT is high, Q_p increases dramatically while it is almost constant for low values of the congestion based delay RTT_c i.e. when RTT is close to RTT_i.

Moreover, in order to better take into account of long term congestion variations an additional control must be done to verify if the intrinsic latency has still the original value provided by the lobby at the beginning of the session. Indeed, as we mentioned previously, this component depends mainly on the network infrastructure and on the global utilization rate of this infrastructure and does not vary a lot through the time. Therefore, when the congestion based latency is attenuated ($RTT_c \approx 0$) thanks to the adaptation mechanism described in Eq 2, the estimated RTT value correspond to the intrinsic latency of the network. In others words, if we detect that the variability of the estimated RTT is low enough then we can deduce that we are measuring this intrinsic latency RTT_i. In this case, if the new estimated RTT_i is different from the current one given, our adaptation algorithm updates the corresponding value in Eq 2. To measure the variability of RTT, we compute the relative standard deviation (RSD) over a large set of estimated RTT values. This parameter is computed as follows:

$$RSD = \sigma/\mu, \tag{4}$$

where σ is the standard deviation and μ the mean value of RTT.

If this value is lower than a given threshold RSD_{th}, the algorithm considers that is measuring RTT_i. Finally, in order to be able to converge locally to an optimized value of the video bitrate the server needs to decrease sometimes the quantization step when the measured $RTT < RTT_i$. In this case, Eq 2 is no longer valid as X given in Eq 3 becomes infinite for very low values of RTT. Basically, in our algorithm, when $RTT < RTT_i$ we compute the new quantization step as follows:

$$Q_p^n = \theta.Q_p^{n-1}, \tag{5}$$

where θ is a decreasing rate fixed empirically in the next section. The adaptation algorithm when receiving a RTCP-RR is summarized ing Alg. 1.

Algorithm 1 Adaptation alg. when receiving a client feedback

Update RTT, σ and μ

$RSD = \sigma/\mu$
if $RSD < RSD_{th}$ **then**
  $RTT_i = \mu$
end if

if $RTT \geq RTT_i$ **then**
  $X = (RTT - RTT_i)/RTT$
  $Q_p^n = Q_p^{n-1}(1 + \alpha X^\beta)$
else
  $Q_p^n = \theta.Q_p^{n-1}$
end if

In the next section, some experimental results are provided in order to validate our proposed bitrate adaptation method.

4.3 Experimental results

To perform our experimental measurements we have built a monitored client-network-server architecture which is basically composed of three computers:

- The server: 3D application and streaming,
- The network emulator,
- The client: video player and interactivity manager.

The server is composed of the rendering engine and the streaming module. The rendering engine is directly connected to a video encoding module based on the x264 encoder and the Darwin Streaming Server (DSS) is then used to provide a complete RTSP based streaming solution. To control and to monitor the network we have chosen the network emulation approach by using the WANem software [3]. By installing it on a computer, the latter is turned into an IP gateway (WANem PC) from which we can increase the intrinsic latency and control the available bandwidth. The video streams are encoded with the following features:

- Average frame rate: 30fps,
- Spatial resolution: VGA,
- H.264 low latency encoding: no B-frames, CAVLC, no rate control buffering.

At the beginning of the session, the fixed quantization parameter ($Q_p = 20$) corresponds to the maximum video quality level achieves to the user. Typically with this high quality level and with the low latency based configuration of the encoder the video bitrate is around 5Mbps and the average PSNR is around 50dB.

In Fig 4, the RTT values estimated by the server are depicted. The network emulator here is configured with a delay of 10ms in each direction ($RTT_i = 20$ms) and a bandwidth of 5Mbps and the adaptation algorithm presented before is not activated. Hence we can note that the variability of the measured RTT is very important and the average measured value is superior to 20 times the intrinsic RTT. In this case, the network bandwidth is equal to the average video bitrate but as we use a variable bitrate encoding, the peak bitrates superior to 5Mbps generate congestion. Practically, this situation makes the playability of the game not really acceptable even if the network conditions are quite good.

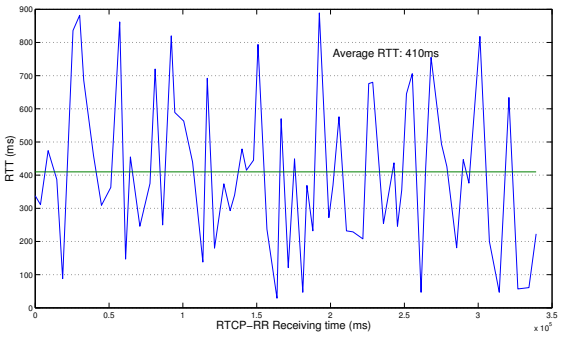


Figure 4: *Measured RTT ($RTT_i = 20$ms, bandwidth: 5Mbps).*

In Fig 5, the RTT values estimated by the server and the PSNR evolution of the encoded video are depicted. For this simulation, the adaptation algorithm is used with the following parameters: $\alpha = 0.1$, $\beta = 1$ and $\theta = 0.9$. Practically, this comes to increase Q_p of 10% when RTT is high and to decrease Q_p of 10% also when $RTT < RTT_i$. In addition, the network emulator is configured with a delay of 10ms in each direction ($RTT_i = 20$ms) and a bandwidth of 2.5Mbps. We can see that the server spends around

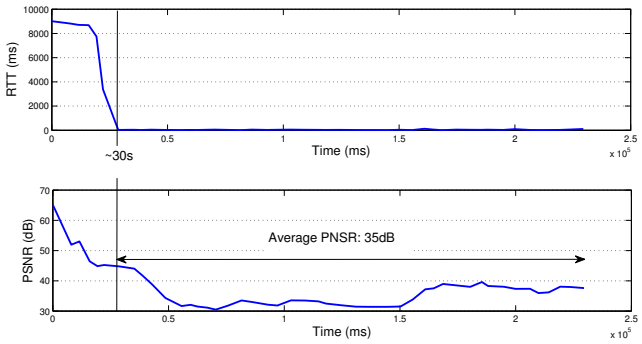


Figure 5: *Measured RTT and PSNR ($RTT_i = 20$ms, bandwidth: 2.5Mbps).*

30s to adapt the video bitrate and to reach a stable state where $RTT = RTT_i$. The cost of this adaptation is a significant decrease of the image quality. However, the bandwidth is much more limited than in the previous simulation and an average of 35dB is still acceptable. Finally, thanks to the adaptation algorithm the period of unacceptable playability is limited to a few seconds at the price of slight decrease of the image quality.

In Fig 5, the RTT values estimated by the server and the PSNR evolution of the encoded video are depicted. The network emulator here is configured with a bandwidth of 5Mbps and the delay in each direction varies from 10ms ($RTT_i = 20$ms) to 40ms ($RTT_i = 80$ms) during the session. As previously mentioned in this paper, this kind of event is not very usual and corresponds to a deep modification of the network conditions. In order to catch this intrinsic delay variation the algorithm computes the relative standard deviation and update the RTT_i parameter when this deviation is higher than 20% ($RSD_{th} = 0.2$). We can see that the server

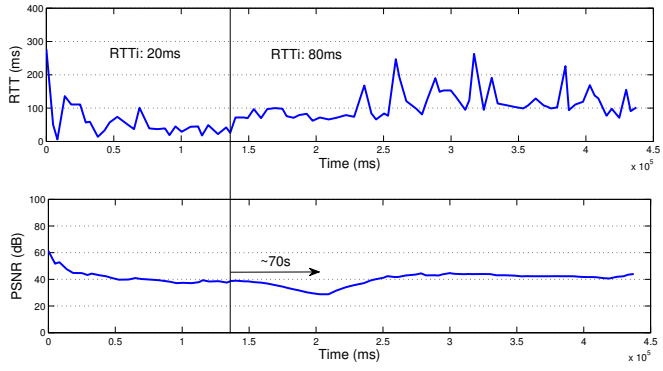


Figure 6: *Measured RTT and PSNR ($RTT_i = 20$ms and 80ms, bandwidth: 5Mbps).*

spends around one minute to update its estimated RTT_i value. During this period, due to the congestion control, the algorithm slightly decreases the quality of the video and when RTT_i is updated, the PSNR comes back to its original range of values. The algorithm does not adapt significantly the bitrate because it detects that the

[3] http://wanem.sourceforge.net/

variation of RTT is not due to congestion. Finally, the algorithm is able to differentiate the two origins of the latency: intrinsic delay or congestion, and then to take the appropriate decision when updating the quantization step.

5 Conclusion

This paper proposes a complete framework for remote rendering applied on video games that can be obviously generalized to any 3D graphics and complex 2D applications. After presenting the state of the art, our approach has been described in details. The retained solution belongs to the family of "2D Pixels" remote rendering and was developed within the framework of the Kusanagi project.

In terms of architecture, our platform is composed of three main modules: the Kusanagi plug-in, the lobby server, the MPEG-4 client. The Kusanagi plug-in turns the core application into a remote controllable software by adding some encoding and streaming functionalities. The lobby server is the central program that manages the resources sharing and acts as an interface between the application and the users. The choice of an MPEG-4 compliant solution allows developing our services very easily and rapidly on a large variety of devices and benefiting from the last advanced of these predominant standards.

The Kusanagi gaming platform has been tested in a very high bitrate optical fiber based network. The results of this test in terms of users experience feedbacks are presented. The conclusions of this benchmark point to a need of more adaptability in order to improve the user QoE. In the last part of this paper, an adaptation algorithm is proposed to optimize the video encoding quality following the network conditions of each user. The proposed algorithm is based on a repeated round trip time measurement. Using standards RTCP reports, this solution can be implemented very easily and without complexity overhead on the server side. The presented simulation results show that our method is very efficient for detecting congestion based latency and to adapt the video bitrate adequately.

Finally, the future works will aim at improving the resource sharing between users by splitting the encoding process into threads and allocating CPU following user constraints. In addition, video encoding optimizations should be obtained by using additional information from the 3D rendering engine.

Acknowledgements

The work presented here is supported by the European project Kusanagi, labellized by the Celtic consortium.
Jack Keane, the game used for the Beta tests, was provided by the Deck13 German video game company.

References

AL-REGIB, G., ALTUNBASAK, Y., AND MERSEREAU, R. M. 2005. Bit allocation for joint source and channel coding of progressively compressed 3-d models. *Circuits and Systems for Video Technology, IEEE Transactions on 15*, 2, 256–268.

ARANHA, M., DUBLA, P., DEBATTISTA, K., BASHFORD-ROGERS, T., AND CHALMERS, A. 2007. A physically-based client-server rendering solution for mobile devices. In *Proceedings of the 6th international conference on Mobile and ubiquitous multimedia*, ACM, New York, NY, USA, MUM '07, 149–154.

DE PRATO, G., FEIJO, C., NEPELSKI, D., BOGDANOWICZ, M., AND SIMON, J. 2010. Born digital / grown digital: Assessing the future competitiveness of the eu video games software industry. Tech. rep., European Commission Joint Research Centre.

DIEPSTRATEN, J., GORKE, M., AND ERTL, T. 2004. Remote line rendering for mobile devices. In *Proceedings of the Computer Graphics International*, IEEE Computer Society, Washington, DC, USA, 454–461.

HUMPHREYS, G., ELDRIDGE, M., BUCK, I., STOLL, G., EVERETT, M., AND HANRAHAN, P. 2001. Wiregl: a scalable graphics system for clusters. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, SIGGRAPH '01, 129–140.

HUMPHREYS, G., HOUSTON, M., NG, R., FRANK, R., AHERN, S., KIRCHNER, P. D., AND KLOSOWSKI, J. T. 2008. Chromium: a stream-processing framework for interactive rendering on clusters. In *ACM SIGGRAPH ASIA 2008 courses*, ACM, New York, NY, USA, SIGGRAPH Asia '08, 43:1–43:10.

JURGELIONIS, A., FECHTELER, P., EISERT, P., BELLOTTI, F., DAVID, H., LAULAJAINEN, J. P., CARMICHAEL, R., POULOPOULOS, V., LAIKARI, A., PERÄLÄ, P., DE GLORIA, A., AND BOURAS, C. 2009. Platform for distributed 3d gaming. *Int. J. Comput. Games Technol. 2009* (January), 1:1–1:15.

LAMBERTI, F., ZUNINO, C., SANNA, A., ANTONINO, F., AND MANIEZZO, M. 2003. An accelerated remote graphics architecture for pdas. In *Proceedings of the eighth international conference on 3D Web technology*, ACM, New York, NY, USA, Web3D '03, 55–ff.

LE FEUVRE, J., CONCOLATO, C., AND MOISSINAC, J.-C. 2007. Gpac: open source multimedia framework. In *Proceedings of the 15th international conference on Multimedia*, ACM, New York, NY, USA, MULTIMEDIA '07, 1009–1012.

NAVE, I., DAVID, H., SHANI, A., TZRUYA, Y., LAIKARI, A., EISERT, P., AND FECHTELER, P. 2008. Games@large graphics streaming architecture. In *Proc. IEEE Int. Symp. Consumer Electronics ISCE 2008*, 1–4.

NIJDAM, N. A., HAN, S., KEVELHAM, B., AND MAGNENAT-THALMANN, N. 2010. A context-aware adaptive rendering system for user-centric pervasive computing environments. In *Proc. MELECON 2010 - 2010 15th IEEE Mediterranean Electrotechnical Conf*, 790–795.

NURMINEN, A. 2007. Mobile, hardware-accelerated urban 3d maps in 3g networks. In *Proceedings of the twelfth international conference on 3D web technology*, ACM, New York, NY, USA, Web3D '07, 7–16.

QUILLET, J.-C., THOMAS, G., GRANIER, X., GUITTON, P., AND MARVIE, J.-E. 2006. Using expressive rendering for remote visualization of large city models. In *Proceedings of the eleventh international conference on 3D web technology*, ACM, New York, NY, USA, Web3D '06, 27–35.

SIMOENS, P., PRAET, P., VANKEIRSBILCK, B., DE WACHTER, J., DEBOOSERE, L., DE TURCK, F., DHOEDT, B., AND DEMEESTER, P. 2008. Design and implementation of a hybrid remote display protocol to optimize multimedia experience on thin client devices. In *Proc. Australasian Telecommunication Networks and Applications Conf. ATNAC 2008*, 391–396.

YAN, Z., KUMAR, S., AND KUO, C.-C. J. 2005. Mesh segmentation schemes for error resilient coding of 3-d graphic models. *Circuits and Systems for Video Technology, IEEE Transactions on 15*, 1, 138–144.