ROI based video streaming for 3D remote rendering

Nicolas Tizon, Christina Moreno, Marius Preda

ARTEMIS Dept., Télécom SudParis
9 rue Charles Fourier, 91011 EVRY,France
nicolas.tizon, christina.moreno, marius.preda@it-sudparis.eu

***Abstract*—This paper proposes a low computational method to perform ROI (Region Of Interest) based video encoding and adaptive streaming for remote rendering applications. The main objective of the proposed solution is to minimize the latency in the interactive loop even when facing poor transmission conditions. In order to do that, the knowledge of the depth map information provided by the rendering engine is exploited by the real-time video encoder to adapt the bitrate of the transmitted stream. Especially, thanks to an efficient coupling between the rendering and the video encoding stages, the macroblocks of each video frame are encoded with different quantization steps that follow an ROI partitioning. The details of this partitioning algorithm are provided as well with some implementation considerations. The simulation results demonstrate the benefit of our adaptive approach from the user experience point of view.**

I. Introduction

Thanks to the continuous increase of the mass market electronic devices processing capabilities, the access to 3D and complex 2D applications has been generalized well over the frontiers of the professional practices. However, even if the multimedia platforms (PC, tablet, smart-phone...) are more and more powerful, especially those with advanced GPUs, the 3D graphic application designers tend to exploit their capabilities to the edge and thus condemn these devices to a very short life expectancy. In addition, the computing domain met last years a growing tendency to externalize the application execution process, from the client to the network. This so called cloud computing approach allows to benefit from new released applications, even with thin-end devices, and also favors the users access, "every where and every time", to their data and applications. One example among others, cloud gaming offers new possibilities integrating hardware, software and service. Several companies such as OnLive, OTOY, Gaikai begin to propose services that involve game streaming technologies.

Likewise, our proposed remote rendering solution is based on standardized video encoding and interactivity streaming methods and was developed within the framework of the Kusanagi project [1]. 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. The latency is a major problem in both cases and requires specific optimizations of the whole chain: from the rendering server to the display device. Moreover, even if an efficient latency control is a necessary condition to run an application remotely with an acceptable QoE, one must pay a special attention to the way the application is executed on the server. Indeed, in order to minimize the latency during the signal processing and communication stages, a safe strategy would consist in over provisioning the resources (CPU, GPU, bandwidth) allocated to each user. In this case, the multi user requirement is very difficult to achieve and not really efficient in terms of pooling of resources.

[1] http://www.kusanagi.eu/

II. Remote rendering

A. Previous work

In the scope of remote rendering techniques, two main families of solutions could be mentioned: Graphics commands, 2D Pixels steaming. In [1], [2], the graphics commands which are exchanged between the application and the graphics card on the server are intercepted and sent to the client. Then, the rendering is performed by the client's GPU. In [1] 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. In [2], a graphic commands solution is proposed to reach satisfactory performances in terms of latency and to serve many users without consuming too many 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 [3] and [4] the 2D rendered pixels are sent through a video stream and in [5] they are coded as still images. In [6], a 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. Finally, in [7], [8], mixed approaches are proposed to build efficient multi-environment remote rendering platforms, the latency being restricted below a certain level.

In the category of 2D pixel streaming based solutions, a few works concerning video encoding optimizations by using side information from the 3D rendering exist. In [9], the authors propose an optimized MPEG-4 video encoder that benefits from the available model of the scene to better

978-1-4577-1434-4/11/$26.00 ©2011 IEEE

compress the sequence. Two main contributions are described. The first one is a MPEG-4 based object segmentation that distinguish two regions (background and foreground) when applying the quantization factor. In addition, an accelerated motion estimation algorithm is described. In this algorithm, the optical flow is calculated by using the scene model and the camera parameters. In [10], Herzog et al. present a framework where the video encoder provides perceptual error thresholds to the renderer. Those threshold errors are used to control the rendering quality by considering the information loss introduced by the video encoder. This strong coupling between rendering and encoding aims at accelerating the overall frame processing. In the same way, an accelerated video encoding algorithm is described in [11] by using render context information. The general principle here consists in using both the projection information and the z-buffer in order to calculate the motion vectors. This rendering information is integrated in a rate-distortion based decision that allows providing an optimized H.264/AVC encoder.

To the best of our knowledge, these works represent the principal contributions in the the field of video streaming based remote rendering optimized systems. However, while these studies mainly aim at decreasing the overall processing load of the system, here we propose a solution to improve the encoding and streaming adaptivity by exploiting the depth map information from the rendering engine.

B. *Overview of the kusanagi platform*

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. 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.

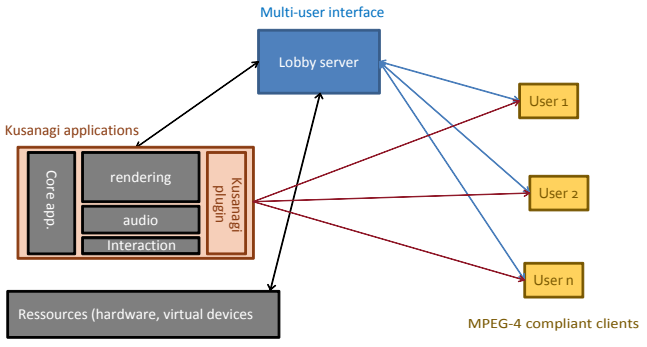


Fig. 1. Kusanagi framework.

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 kinds of data:

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

The lobby server is the central program that manages the resources sharing and acts as an interface between the application and the users. 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). At the beginning of the session, the lobby server can also provide some user based information to the Kusanagi plug-in like the initial round trip time (RTT), measured between the server and each client. This RTT value will be used further (in III-B) to initialize one parameter of the adaptation algorithm.

At client side, a full multimedia player with networking, media, user interactions management and rasterizing is used [12]. The MPEG-4 scene description language allows managing an interactive multimedia session and supporting any kind of graphic applications. The client supports also a wide range of video and audio codecs (H.264, AAC, among others) and is a complete network client. Especially, it supports RTP/RTSP streaming including RTP/UDP streaming and RTP over RTSP(TCP). Jointly with RTP, the client implements the control protocol RTCP. In the next section, an algorithm that exploits the RTCP reports from the client (RTCP-RR) to adapt the transmitted bitrate is presented.

III. ROI AND LATENCY BASED BITRATE ADAPTATION

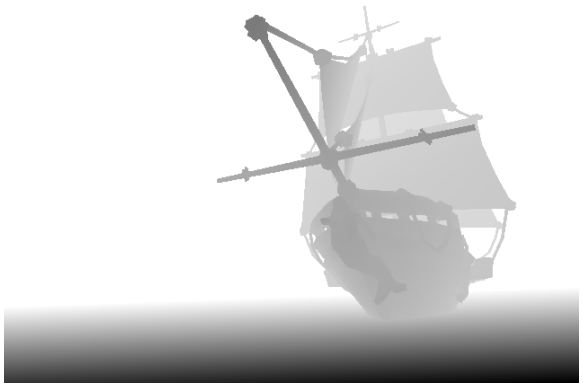
During the rendering process, for each frame, a depth map is generated on the GPU through the well known z-buffer. As previously mentioned, this information can be very useful to improve the video encoding process [9], [11]. In this section, we propose to use this information in order to divide the image into different regions and to adapt the bitrate next with a finer granularity. The first issue addressed here is the ability to obtain an automatic and relevant partitioning of each frame without overloading the processors. Then, we improve a bitrate adaptation algorithm previously described in [13], by taking into account of the differentiated regions of the picture.

A. *Low computational ROI partitioning*

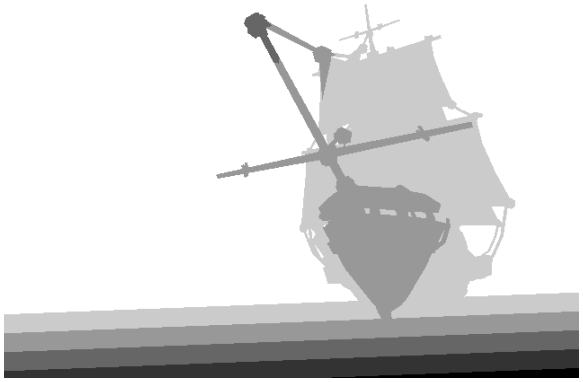
In many practical cases, the objects that belong to the scene's foreground are subject to a higher visual attention from the user and are more relevant in terms of QoE. From this assumption, one can consider that the z-buffer used to render a scene, contains the main information when defining regions of interest. In the Kusanagi platform, we use the Direct3D9 texture interoperability with CUDA [14] to capture and to store the depth map on the GPU. Next, we process this depth image on the GPU and finally obtain a macroblock based partitioning of each frame as described in Fig. 2.

First of all, the depth values are quantized over a small number of integer values (6 values in Fig. 2 (c)). This processing is implemented with reduction based programming methods, well suited to be efficiently executed on the GPU.

(a) YUV frame



(b) Depth map



(c) Quantized depth map

(d) Quantized and rescaled depth map

Fig. 2. Depth map processing steps (partitioning: 6 regions).

The integer value of each pixel, from 0 to 5 in the given example, represents its depth level. The lower values (black) correspond to the foreground and the higher values (white) to the background. As this ROI partitioning is used next to perform the bitrate adaptation during the video encoding stage, the next processing step consists in downscaling the quantized depth map values (d). This downscaling is basically obtained by averaging the depth values for each macroblock (16×16 pixels) and by dividing this mean value by the number of different depth levels in the macroblock. After this last GPU processing, each macroblock is represented by an integer value $(0..N)$ which is supposed to indicate its level of importance. The lower levels correspond to the higher interest for the user and this value will be used as a quantization factor offset during the video encoding stage (see next section). This last operation on the depth map allows a better discrimination of homogeneous areas from regions with high frequencies (object borders). For instance in the given example, some object borders on the boat are almost in the background, however thanks to the division by the number of different levels, they will be better taken into account during the video encoding.

B. Adaptation algorithm

In [13], 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 (RTT) measurement. The proposed measurement consists in separating the latency into an intrinsic component RTT_i and a congestion based component RTT_c:

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

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 $RTT(t)$ and with the intrinsic delay RTT_i measured by the lobby, the server can deduce the congestion based latency $RTT_c(t)$. 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. Hence, 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, our adaptation algorithm makes an update with the corresponding value. 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:

$$RSD = \sigma/\mu, \tag{2}$$

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.

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, in [13] the server adapts the quantization step value as follows:

$$Q_p^n = Q_p^{n-1}(1 + \alpha X^\beta), \tag{3}$$

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:

$$X = (RTT - RTT_i)/RTT \tag{4}$$

This 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. 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 3 is no longer valid as X given in Eq 4 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.

In [13], the quantization factor Q_p^n used in Eq 3 and Eq 5 has the same value for all the macroblocks of the picture. In addition, in the previous section we described a depth map based process that provides, for each macroblock, an integer value from 0 to N that indicates its level of importance. As the lower values correspond to regions of higher interest, we can directly map these integer values to quantization factor offsets. Let's denote $\Delta\mathbf{Q}_p^n$ the matrix containing these offset values and represented by picture (c), in Fig. 2. The coefficients of this matrix belong to the integer interval $0..N^n$, where N^n is the higher offset value for the n^{th} iteration of the algorithm. Then we can write $\mathbf{Q}_p^n$, the matrix of the quantization factors for the entire picture as follows:

$$\mathbf{Q}_p^n = \mathbf{Q}_{p\,b}^n + \Delta\mathbf{Q}_p^n, \tag{6}$$

where $\mathbf{Q}_{p\,b}^n$ is the matrix that contains only one value $Q_{p\,b}^n$, which is the base quantization factor common to all the macroblocks.

Let's define now $Q_{p_{max}}$ the maximum value, acceptable for the quantization factors :

$$Q_{p\,b}^n + N^n \leq Q_{p_{max}}. \tag{7}$$

This maximum value is fixed in function of the application requirements and highly depends on the content of the video. Then, the adaptation algorithm when receiving a RTCP-RR is summarized in 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

$Q_p^{n-1} = Q_{p_b}^{n-1} + N^{n-1}$
if $RTT \geq RTT_i$ **then**
  $X = (RTT - RTT_i)/RTT$
  $Q_p^n = Q_p^{n-1}(1 + \alpha X^\beta)$
  if $Q_p^n > Q_{p_{max}}$ **then**
    $Q_{p\,b}^n = MIN(Q_{p_b}^{n-1} + 1, Q_{p_{max}})$
  else
    $Q_{p\,b}^n = Q_{p_b}^{n-1}$
  end if
  $N^n = Q_p^n - Q_{p\,b}^n$
else
  $Q_p^n = \theta.Q_p^{n-1}$
  if $Q_p^n \geq Q_{p_b}^{n-1}$ **then**
    $Q_{p\,b}^n = Q_{p_b}^{n-1}$
    $N^n = Q_p^n - Q_{p\,b}^n$
  else
    $N^n = N^{n-1}$
    $Q_{p\,b}^n = Q_p^n - N^n$
  end if
end if

The first part of the algorithm concerns the computation of Q_p^n and is detailed at the beginning of this section. Next, if the increase of Q_p^n (when $RTT \geq RTT_i$) leads to a value that exceeds $Q_{p_{max}}$, hence the base quantization factor $Q_{p\,b}^n$ is incremented by one (or saturated to $Q_{p_{max}}$). In this case, the higher offset value is obtained by calculating the difference: $N^n = Q_p^n - Q_{p\,b}^n$. Inversely, if the decrease of Q_p^n (when $RTT < RTT_i$) leads to a value lower than $Q_{p_b}^{n-1}$, the higher offset value does not change ($N^n = N^{n-1}$) and the base

quantization factor is obtained by calculating the difference: $Q^n_{p_b} = Q^n_p - N^n$.

The general principle of this heuristic computing is to use, as much as possible, the variations of the quantization offsets to perform low scale bitrate adaptations and to only use Q^n_p to perform large scale adaptation on the entire image, when it is necessary. In the next section, some simulation results demonstrate the interest of such an approach, in terms of image quality, when facing degraded transmission conditions.

IV. 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. The application used here to test our solution is a point and click adventure game called "Jack Keane". To control and to monitor the network we have chosen the network emulation approach by using the WANem software [2]. 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 adaptation algorithm is initialized with the following parameters: $Q^0_p = 20$, $N^0 = 0$ and $Q_{p_{max}} = 35$. 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. On the opposite, a value of 35 for the quantization factor corresponds to PSNR values around 30dB and average bitrates lower than 2Mbps for our gaming content. In H.264, the quantization factor can take its value in the interval $0..51$. By assigning high or low values to $Q_{p_{max}}$ one can turn the streaming more or less adaptive.

In Fig 3, the RTT values estimated by the server and the PSNR evolution of the encoded video are depicted. The PSNR values are computed for two regions of the frames. The first region corresponds to the macroblocks encoded with the maximum quantization factor offset N, and all the others macroblocks are constituting the second region which can be seen as the region of interest. For this simulation, the adaptation algorithm is used with the following parameters: $\alpha = 0.2$, $\beta = 1$ and $\theta = 0.9$. Practically, this comes to increase Q_p of 20% when RTT is high and to decrease Q_p of 10% 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

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

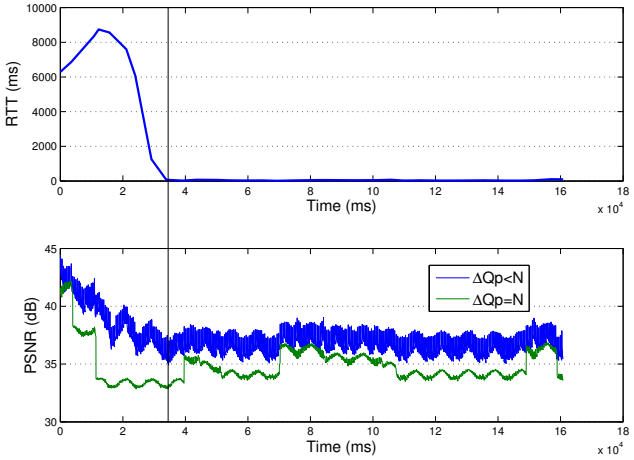


Fig. 3. Measured RTT and PSNR ($RTT_i = 20$ms, bandwidth: 2.5Mbps).

that the server spends around 30s to adapt the video bitrate and a stable state ($RTT = RTT_i$) is reached after 40s.

The cost of this adaptation is a significant PSNR decrease of the non-ROI region, mainly during the first iterations, whereas the quality of the rest of the frame is relatively well conserved. Indeed, the PSNR of the ROI never fall down below 35dB. Finally, we can say that the non-ROI area is used by the adaptation algorithm to "learn" the transport channel profile. In Fig. 4, the evolution of the quantization factor components: $Q_{p_b}(t)$ and $N(t)$) is depicted. We can clearly identify the end of the adaptation phase around the fifth iteration, where $Q_{p_b}(t)$ increases and $N(t)$ decreases. After the tenth iteration, the strong adaptation phase is finished and the values are more constant.

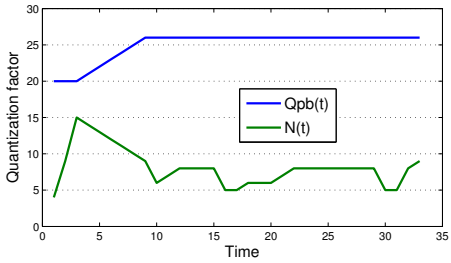


Fig. 4. Quantization factors variations ($RTT_i = 20$ms, bandwidth: 2.5Mbps).

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. 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 10% ($RSD_{th} = 0.1$).

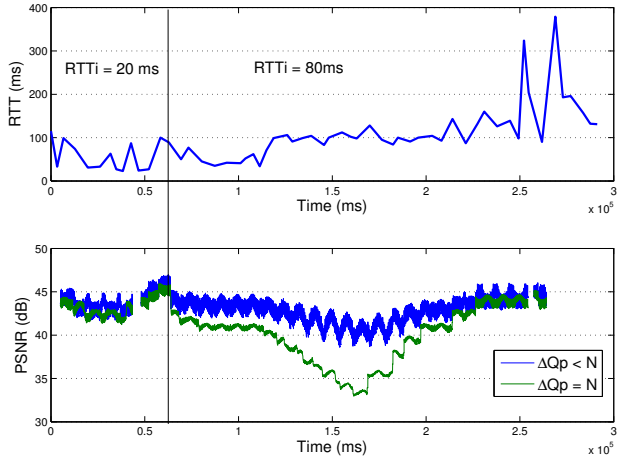


Fig. 5. Measured RTT and PSNR (RTT_i = 20ms and 80ms, bandwidth: 5Mbps).

The server spends around one minute to update its estimated RTT_i value. During this period, due to the congestion control, the algorithm decreases mainly the quality of the non-ROI macroblocks in order to "learn" the new RTT_i value. When this value is updated, the quality of the entire image recovers its initial level and the adaptation period is almost transparent for the user.

V. Conclusion

This paper proposes a low computational ROI based adaptive streaming algorithm for remote 3D applications. The macroblock differentiation used during the video encoding process is obtained thanks to an efficient processing loop of the depth map on the GPU. This coupling between the rendering engine and the encoder has been implemented here in a video game architecture but can be obviously generalized to any 3D graphics application. The proposed partitioning algorithm provides a powerful tool to discriminate automatically the video frame regions in function of the visual content and that is completely executed on the GPU.

In addition, the resulting matrix of this GPU processing is used as an input parameter in our previously proposed bitrate adaptation algorithm. Thanks to this knowledge of the ROIs, the server is able to face bad transmission conditions, in terms of latency, without scarifying the QoE, in terms of image quality.

Finally, the future works will investigate more relevant ROI partitioning by taking into account of interactivity feed-backs. A finer object based partitioning could be used also to improve the video encoding efficiency and more specially to decrease the required processing time.

Acknowledgment

The presented work 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 German video game company Deck13.

References

[1] G. Humphreys, M. Eldridge, I. Buck, G. Stoll, M. Everett, and P. Hanrahan, "Wiregl: a scalable graphics system for clusters," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH '01. New York, NY, USA: ACM, 2001, pp. 129–140. [Online]. Available: http://doi.acm.org/10.1145/383259.383272

[2] I. Nave, H. David, A. Shani, Y. Tzruya, A. Laikari, P. Eisert, and P. Fechteler, "Games@large graphics streaming architecture," in *Proc. IEEE Int. Symp. Consumer Electronics ISCE 2008*, 2008, pp. 1–4.

[3] F. Lamberti, C. Zunino, A. Sanna, F. Antonino, and M. Maniezzo, "An accelerated remote graphics architecture for pdas," in *Proceedings of the eighth international conference on 3D Web technology*, ser. Web3D '03. New York, NY, USA: ACM, 2003, pp. 55–ff. [Online]. Available: http://doi.acm.org/10.1145/636593.636602

[4] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, and J. T. Klosowski, "Chromium: a stream-processing framework for interactive rendering on clusters," in *ACM SIGGRAPH ASIA 2008 courses*, ser. SIGGRAPH Asia '08. New York, NY, USA: ACM, 2008, pp. 43:1–43:10. [Online]. Available: http://doi.acm.org/10.1145/1508044.1508087

[5] M. Aranha, P. Dubla, K. Debattista, T. Bashford-Rogers, and A. Chalmers, "A physically-based client-server rendering solution for mobile devices," in *Proceedings of the 6th international conference on Mobile and ubiquitous multimedia*, ser. MUM '07. New York, NY, USA: ACM, 2007, pp. 149–154. [Online]. Available: http://doi.acm.org/10.1145/1329469.1329489

[6] P. Simoens, P. Praet, B. Vankeirsbilck, J. De Wachter, L. Deboosere, F. De Turck, B. Dhoedt, and P. Demeester, "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*, 2008, pp. 391–396.

[7] A. Jurgelionis, P. Fechteler, P. Eisert, F. Bellotti, H. David, J. P. Laulajainen, R. Carmichael, V. Poulopoulos, A. Laikari, P. Perälä, A. De Gloria, and C. Bouras, "Platform for distributed 3d gaming," *Int. J. Comput. Games Technol.*, vol. 2009, pp. 1:1–1:15, January 2009. [Online]. Available: http://dx.doi.org/10.1155/2009/231863

[8] N. A. Nijdam, S. Han, B. Kevelham, and N. Magnenat-Thalmann, "A context-aware adaptive rendering system for user-centric pervasive computing environments," in *Proc. MELECON 2010 - 2010 15th IEEE Mediterranean Electrotechnical Conf*, 2010, pp. 790–795.

[9] Y. Noimark and D. Cohen-Or, "Streaming scenes to mpeg-4 video-enabled devices," *Computer Graphics and Applications, IEEE*, vol. 23, no. 1, pp. 58 – 64, jan/feb 2003.

[10] R. Herzog, S. Kinuwaki, K. Myszkowski, and H.-P. Seidel, "Render2mpeg: A perception-based framework towards integrating rendering and video compression," *Computer Graphics Forum*, vol. 27, no. 2, pp. 183–192, 2008. [Online]. Available: http://dx.doi.org/10.1111/j.1467-8659.2008.01115.x

[11] P. Fechteler and P. Eisert, "Accelerated Video Encoding Using Render Context Information," in *Proceedings of the 17th International Conference on Image Processing (ICIP2010)*, Hong Kong, China, 26-29th October 2010, pp. 2033–2036, iCIP 2010.

[12] J. Le Feuvre, C. Concolato, and J.-C. Moissinac, "Gpac: open source multimedia framework," in *Proceedings of the 15th international conference on Multimedia*, ser. MULTIMEDIA '07. New York, NY, USA: ACM, 2007, pp. 1009–1012. [Online]. Available: http://doi.acm.org/10.1145/1291233.1291452

[13] N. Tizon, C. Moreno, M. Cernea, and M. Preda, "Mpeg-4-based adaptive remote rendering for video games," in *Proceedings of the sixteenth international conference on 3D web technology*, ser. Web3D '11. New York, NY, USA: ACM, 2011.

[14] J. Sanders and E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, 2010.