A Context-Aware Adaptive Rendering System for User-Centric Pervasive Computing Environments

N.A.Nijdam, S. Han, B. Kevelham, N. Magnenat-Thalmann

MIRALab, University of Geneva
7, rte de Drize, 1227 Carouge, Geneva, Switzerland
{nijdam, han, kevelham, thalmann}@miralab.unige.ch

***Abstract*— In user-centric pervasive computing environments where users can utilize nearby heterogeneous devices anytime and anywhere, context-aware remote rendering is essential. It is impractical not only to manually copy 3D content from one device to another whenever a user moves, but also to render complex 3D data locally on resource-limited devices, such as mobile phones and PDAs. In this paper, we propose a context-aware adaptive rendering system which visualizes 3D content with customized user interfaces dynamically adapting to current device contexts such as processing power, memory size, display size, and network condition at runtime, while preserving the interactive performance of the 3D content. To increase the responsiveness of remote 3D rendering, we use a mechanism which temporally adjusts the quality of visualization, adapting to the current device context. By adapting the quality of visualization in terms of image quality, the overall responsiveness and frame-rate are maintained no matter the resource status. In order to overcome inevitable physical limitations of display capabilities and input controls on client devices, we provide a user interface adaptation mechanism which dynamically binds operations provided by the 3D application and user interfaces with pre-described device and application profiles. We prototyped our adaptive rendering system and experimented with a specific scenario in user-centric pervasive environments.**

I. INTRODUCTION

Recent advances in mobile devices and networks have enabled a new wave of user-centric multimedia convergence which puts the mobile users in the center of a rich and interactive world with 3D content. They can utilize diverse devices anywhere and anytime [5]. Considerable efforts have been made for nomadic access of multimedia in mobile computing [1]. This nomadic access however may limit users in that it forces them to use their mobile devices as a single point of multimedia access. It is also impractical to manually transmit 3D contents from one device to another whenever a user switches devices in order to make users to utilize diverse device because not only does it distract users to copy 3D content but also some resource-limited devices, such as mobile phone and PDA, may not render complex 3D data locally. To overcome resource limitations of mobile devices, a data-centric approach [8], with which a client device used semantically equivalent 3D data depending on different device capabilities, has been used. However, this approach cannot guarantee consistency of modified 3D content while a user is switching across diverse devices. Adaptive remote rendering approaches [4][11] have been proposed to seamlessly access 3D data with heterogeneous devices. They adaptively generate size of images and frame rate based on the preconfigured client rendering capabilities, display size and network conditions on a dedicated server and stream to the client devices. They, however, decrease interactive performance of 3D content not only because it takes long response time to get feedbacks if 3D content is fully rendered remotely and streamed back to a client but also because they do not support to adapt dynamic context changes such as network bandwidth at runtime. Hybrid rendering approaches [3][16] maximize the utilization of resources on the server-side by delegating the rendering of highly complex 3D scenes to a dedicated server. They can support the client's interaction with 3D content even in the case of temporal network failure. However, they don't enable the users to exploit diverse devices adapting to the current contexts at runtime. Clients therefore still need to transfer large amounts of 3D data to the server on-demand. They also do not consider client device capabilities, such as display size and input controls, to represent the user interface. It is impractical to provide complex full-featured controls on resource-limited devices such as PDA and smart phone with small display.

In this paper, we propose a context-aware adaptive rendering system which visualizes 3D content and a user interface, while dynamically adapting to the current context such as device availability in the current location and their capabilities, i.e., processing power, memory size, display size, and network condition, while preserving interactive performance of 3D contents. To increase responsiveness of remote 3D visualization, we use a mechanism which adjusts quality of visualization adapting to the current device contexts. By adapting the quality of visualization in terms of image quality, the overall responsiveness and frame-rate is maintained no matter the resource status. To overcome inevitable physical limitations of display and input controls on client devices, we provide an interface adaptation mechanism which dynamically binds operations provided by the 3D application and user interfaces with pre-described device and application profiles.

The remainder of this paper is organized as follows: In section 2 we analyze the technical requirements of context-aware adaptive rendering with a specific scenario. In Section 3 we analyze the pros and cons of related work in remote and adaptive rendering systems. Section 4 describes our proposed adaptive rendering system which meets the requirements

978-1-4244-5794-6/10/$26.00 ©2010 IEEE

described in Section 2. Section 5 describes our implementation and discusses the results from our experiments. Conclusion and future work are discussed in Section 6.

II. Technical Requirements

While there are many possible scenarios, we present a specific scenario in this section in order to make the problems more concrete and to describe key challenging issues regarding interactive 3D content visualization in user-centric pervasive computing environments.

"Chloe, who is an undergraduate student, is attending a course at the university. After the class, and while on her way to the library, she starts to review the lesson by manipulating 3D course material on her PDA. After arriving at the library, a high-end desktop PC is at her disposal to review 3D content without reinitializing her 3D manipulation session. On her way back home she continues to browse through the 3D course material with her smart phone. Once she comes back home, she completes the 3D manipulation on her PC." (Fig. 1).

Fig. 1 Envisioned scenario.

Two key technical issues should be addressed to enable this scenario. The first issue is a polymorphic presentation adaptation to overcome resource heterogeneity of the client devices while providing suitable responsiveness to users. This means that the presentation of 3D content, i.e., frame size and frame rate, has to be dynamically adapted to the current device context such as processing power, memory size, display size, and network condition at runtime. Presentation on client devices can be adjusted with pre-configured device profiles but the contexts of the devices vary over time and this variance is nondeterministic, which results in clients being under- or overloaded. Therefore, it requires a presentation adaptation scheme that dynamically adapts the current presentation context by continuously observing the underlying device context such as available network bandwidth and computational resources. The second issue comes from inevitable physical heterogeneity in the capabilities of display and input controls on the client devices. This requires dynamic interface adjustment based on the current display instead of providing complex full-featured controls on resource-limited devices such as PDAs and smart phones with small display. There are typically three types of adaptation. First, user inputs with devices (mouse, keyboard, pen, etc.) should be dynamically mapped to user interface controls. It implies that user inputs are interpreted differently based on input devices. Second, different user interfaces could be represented depending on client capabilities. A limited set of functionality can be displayed on the PDA compared with high-end devices. Third, user interfaces (controls) can be dynamically mapped to simulation and rendering functionality.

III. Related Work

Several adaptive rendering mechanisms [3][4][8][10][11][16] have been proposed to overcome resource heterogeneity. Krebs et al. [8] proposed a data-centric approach where users are using the same or semantically equivalent 3D data with heterogeneous devices. It is composed of three tiers. The presentation tier contains the controller and view parts of the MVC paradigm [7]. The domain tier contains application semantics as well as data. The manifold tier glues the presentation and domain tier. Device heterogeneity is handled by pre-described profiles in XML/XSL. The XSL document maps elements in the XML document to nodes in the result tree so that the renderer knows how to render them. Parsing the common XML file and the local XSL file generates the view at a particular user's machine. However, this approach is not able to guarantee the consistency of modified 3D data. It furthermore takes a long time to transfer the 3D data when a user moves across diverse devices. Preda et al. [11] proposed a formal model of adaptive rendering for multi-user 3D games. They defined a set of transformations for adaptation to heterogeneous client devices, rendering, coding, simplification, and modeling and possible process chains for visual adaptation. However, they did not provide any mechanism to dynamically adapt the current context at runtime and to increase interactive performance for users. To increase interactive performance of 3D simulation and rendering, a hybrid rendering approach in which a common subset of 3D models are rendered on both the client and server sides, could be utilized. Engel et al. [3] proposed a system which maximized utilization of resources on server-side by delegating rendering of highly complex 3D scene to a dedicated server. It, however, still limited users to exploit diverse devices adapting to the current contexts because a client needed to transfer large amount of 3D data to server on-demand. Weaver et al. [16] proposed a perceptually adaptive rendering system for immersive virtual reality which reduced the computational burden by rendering detail only where it is needed. Eccentricity from the user's point of gaze is used to determine when to render detail in an immersive virtual environment, and when it can be omitted in order to display higher quality environments without reducing interactivity.

IV. Context-Aware Adaptive Rendering System

In order to support polymorphic visualization of 3D content on heterogeneous devices, our adaptive rendering system exploits the PSLA model [6], in which an interactive 3D application can be typically divided into four parts; presentation, semantics, link, and adapter. A presentation supports the interface between a user and the shared semantics as well as the visualization of 3D contents. The semantics are organized into software modules which encapsulate the 3D content. The presentation and semantics are dynamically

bound with adapter at initialization as well as runtime. Our context-aware adaptive rendering system is composed of five logical layers as shown in Fig 2. The network layer provides not only the low level connection between the server and client using TCP or UDP but also the transparent change of subscription end-points for seamless subscription to 3D content when a user switches one device to another. The communication abstraction layer, which is composed of two major components, being the communication manager and event manager, abstracts low-level events to high-level (application-level) events in order to mask low-level events generated by heterogeneous devices. It also marshals/un-marshals incoming events and redirects them to the appropriate components. The adaptation layer on the client side has two components, the context manager and resource manager. The resource manager constantly monitors the current state of the resources in terms of application performance. Based on the information provided by the resource manager, the context manager can determine the optimal performance by taking adaptation decisions. The decisions are translated into events and sent to the adaptation manager on server-side so that it executes the appropriate adaptation strategy. The presentation layer contains the render engines. On the server-side, a 3D rendering engine renders 3D content and generates images of different quality based on current client resource capability. To transmit images to a client, the frame-buffer is taken into the compression manager. This generates a stream of compressed frames (similar to a video stream) that is being decompressed on the client side and displayed using a 2D rendering engine. The dynamic state management on the server-side maintains the current state of the 3D simulation in order to switch from devices without losing the current simulation. In the application layer the application logic is responsible for client side handling of input and overall client side functionalities. On the server side it is the "data storage" containing the 3D content that is needed for the 3D rendering and simulation.

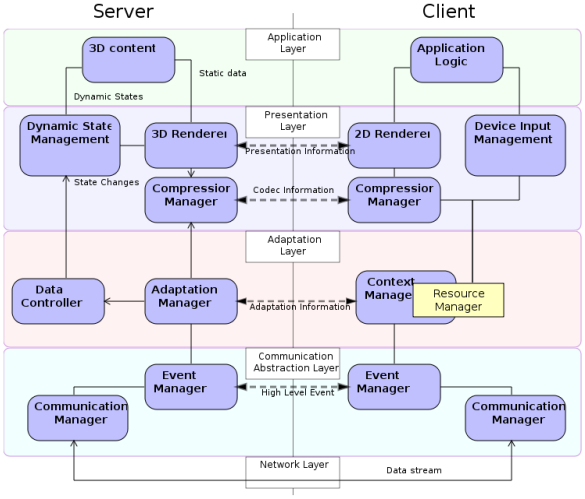

Fig. 2 Overall architecture of context-aware adaptive rendering system.

A. *Run-time Presentation Adaption Control*

We mainly focus on increasing responsiveness of 3D content taking into account the current context of devices. This means that whenever the user performs an action, the response to this action should be shown to the user within a certain amount of time. Fig. 3 shows the typical flow of the remote rendering with interaction. The time taken between the creation of an event and getting the results back to the user is the time that we reduce with our adaptation manager. The three main stages are: send the events, perform simulation cycle and send the results.

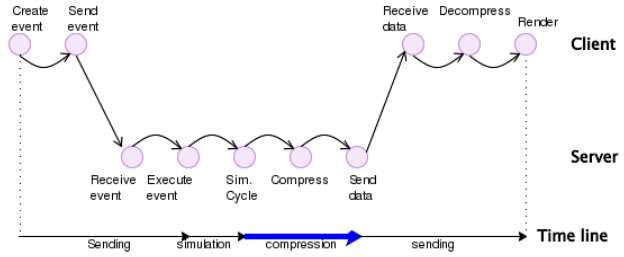

Fig. 3 Round trip time, from input to visual response.

Assuming that the simulation is fast enough to deliver a response within a certain amount of time, the main adaptation takes place after the simulation step. Here a compression algorithm is used in order to reduce the amount of data that needs to be transferred. For a thin client this is the compression of the image-data into an image/video stream. For hybrid or full client rendering, the system compresses the actual 3D data (e.g. lossless compression of vertices). In Fig 4, we can see the several adaptation points given by diamond shaped objects.

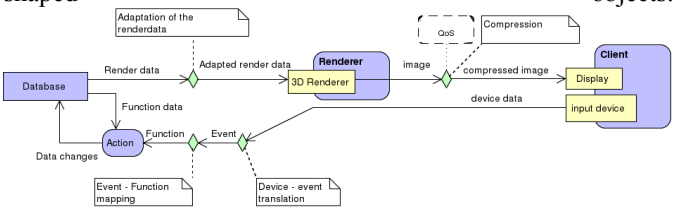

Fig. 4 Adaptation points, denoted by the diamond shaped objects.

The dynamic adaptation is aimed at the regulation of the output data. We increase the perception of visual interaction with a remote rendered 3D environment by introducing a 'temporal adjustment of presentation quality' adaptation mechanism. By decreasing the quality, the decoding speed is increased on the client side. Additionally for thin-clients the frame-rate is increased. Whenever the user interacts with a 3D object, the actions are displayed faster but at the expense of image quality. The decrease of image quality is achieved by switching to a stronger compression algorithm (if the decoding side can keep up), by changing compression parameters and/or if the client supports fast image scaling the actual image resolution can be reduced and up-scaled on the client side. We also use another approach,which detects changes in the rendered frame and compares it with previous rendered frames. If the amount of difference from the previous frame is greater than a certain threshold the adaptation control changes the compression strategy by reducing the quality temporarily. The third approach is the encapsulation of

expected load for functions or performance indicators. This can be done manually or at run-time. Manually by reading the load factor for a function from a predefined configuration file. This is the same configuration file that defines which events are bound to specific functions. The run-time approach is a profiler algorithm that, whenever a function is executed, (based on an incoming event) measures the time that is needed for full execution. The difficulty here is that a function itself can be very fast, but the effects of the function (influencing the simulation) can result in longer times. Either by predefining the load factor or by adjusting the load factor through the measurement of function execution another feature can be introduced, namely load predection.

By anticipating the actions and their resulting timeframe for user feedback the level of adaptation can be controlled to achieve a constant time cycle. Another, more straight-forward, approach to keep a constant time cycle is to keep a constant data rate. This does not take into consideration the quality of the network but only the amount of data being sent. It can be achieved by fixing a specific frame rate and keeping a threshold on the data size for each frame. If the size of a frame after compression is higher than a certain threshold the compression quality is lowered until the threshold is met again. By default, we use this passive adaptation rule. However, depending on the device profile a preferred frame-size is given, which is the actual data size for a frame (after compression) to be sent to the client. An extension to the dynamic adaptation control is based on the network load and should be balanced together with the frame rate. The whole procedure is shown in Fig. 5.

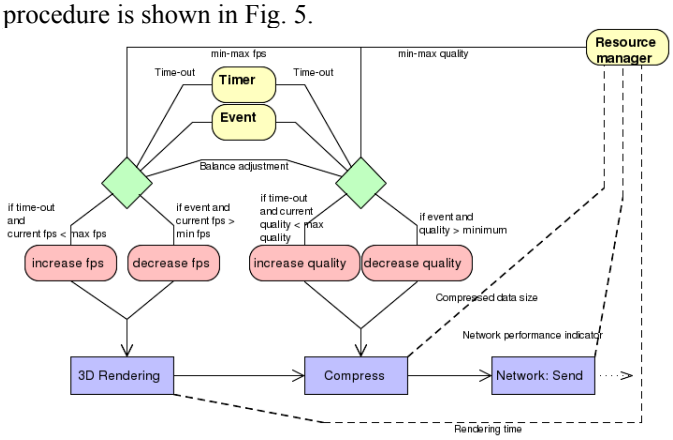

Fig. 5 Adaptation manager: the adaptation control.

B. *Dynamic Interface Adaptation*

In order to overcome the physical heterogeneity limitations in display capabilities and input controls on client devices, we provide a dynamic user interface reconfiguration mechanism for interaction with 3D content. It means to change the way the interface is presented to the user (big screen or small screen brings several design issues with it) and to adapt to input capabilities of the client device.

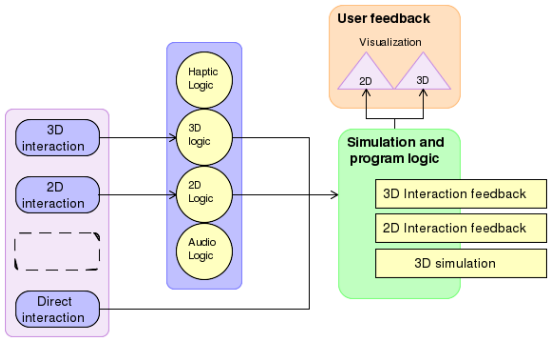

Fig. 6 Dynamic mapping of user interface.

Fig. 6 shows the details of dynamic mapping. Each Element is an event handler, and is listening to certain events to which it can respond by creating new events. Each event received from an input device is first handled by the Client Event handler, this module determines if the event should be handled by the client or the server. Then on the server or client side, the raw event is sent to the corresponding Element that is listening to the device. An Element can have a representation, which can be a form of visualization in 2D or 3D, but also in the form of audio or any other form of feedback. The handling and control of the representation is performed by the logic that is assigned to the specific Element. The representation of the element is loosely coupled and therefore it may reside on the server or client side. According to the Element logic, the raw event can be translated into higher level events. These can be a representation update, simulation, adaptation or any other application specific event. This makes an Element a dynamic building block that is used by the User Interface adaptation manager to construct and modify the user interface whenever needed. For example a combo box on a pc is displayed in 2D using Windows, GTK or QT native widgets, but it can also be displayed as a ring selection in 3D, using a different representation but the same logic, and accepting the same events, or accept different events but with the same logic.

Currently there are three forms of user-interface-interaction implementations, two with a visual representation and one with no “interface” visualization as follows:

Interaction with 2D interface: The default 2D interface providing windows, buttons, text fields and other widgets with its entire well established visual feedback mechanism. This is an intermediate layer between input device and the 3D simulation, providing the means to perform complex operations and provide the necessary data for it. For example a textbox and a button, might update simulation parameters, which will be more dynamic than an “increase” and “decrease” button, but at the expense of more user interactions (typing and confirmation with a mouse-click or keyboard command, instead of a single click).

Interaction with 3D interface: This form of interaction is integrated into the 3D environment and therefore provides the user with direct interaction with 3D objects This is usually achieved by means of a 3D pointer, which is able to hover over 3D objects and whenever the user executes a command to select or perform some other action it is directly executed on the specific object. Aside from the input device, this

interaction mechanism boils down to ‘see the 3D object’, ‘select the object’, ‘execute operation on object’ and ‘wait for visual feedback’. The visual feedback should be direct if the simulation behind is a real-time simulation. In essence this approach is similar to the 2D interaction mechanism, but by adding one dimension more it provides us with new abilities as to how we perceive the interaction and handling of virtual objects. The implementation of a 3D interface feedback may have similar features as its 2D version; for example selected objects should be highlighted, or just by hovering over objects information on them should be displayed. This feedback is different from the 3D simulation itself, as it doesn’t intervene with the simulation, but just provides visual 3D feedback.

Interaction with Implicit Inputs: It has input device bindings directly to an event that changes simulation parameters without showing it in the sense of an interface. For example 3D movement, the camera is being moved around in the 3D world. The visual feedback is that the user gets the impression of moving in a virtual world, but there is no direct feedback from any object in the world. It is exactly the same as the 3D interaction, but without a direct 3D “interface” implementation. All of these interaction mechanisms can be used with any kind of human interface device (HID), such as a pointing device (mouse) or keyboard etc. Each input device can be coupled differently according to its input capabilities, the application capabilities and user preferences. The binding between the application and device can be hardcoded and to some degree the user can set its own preferences. However in order to switch from one device to another, the interface mapping needs to adapt accordingly. Another form of adaptation is dynamic coupling in which coupling changes depending on the current context. For example if the connection between device and server is very bad, than 3D rotation can be limited from smooth rotation to fixed rotation (for example front, side, top view).

V. Implementation and Experiment Result

A. Implementation

We used a seamless user mobility support mechanism [12] in the network layer and a customized version of the Atlas framework [9], which is a scalable network framework for distributed virtual environments, for the communication abstraction layer in order to handle the image/video stream and events. Inputs from a client device are, depending on the application, translated into local events or server events. For example, a simple rotation of a 3D object, which is done by clicking and dragging the mouse, is executed locally or remotely depending on the device used. If the device has render capabilities and is able to render the 3D object locally, then the rotation can be performed locally, otherwise the event is sent to the server and rotation takes place in the remotely rendered 3D environment. If the action is performed locally then the results of the action still need to be synchronized with the server, however this can be achieved by synchronizing based on end result, instead of micro updates for each device event (like mouse click, move etc).

The 3D rendering engine is based on Open Scene Graph (OSG) renderer [18], It is based on a scene graph approach which makes it possible to dynamically change the rendering procedure based on the client device capabilities. This can range from partial rendering (part on server and part on client), representation (different shading style, e.g. realistic, cell shaded etc) and specific adaptation algorithm that are tightly coupled with the rendering (off-screen rendering, layered rendering, color channel manipulations). On the client side, a similar engine is used, if the client device is capable of rendering 3D data. However it also provides other means to render the image/video, where it simply is "blitting" the incoming image data onto the screen. The image can contain meta-information about its size and location on the screen, in order to only update a smaller region on the screen. This mechanism makes it possible for low end devices to view complex data that otherwise could not be rendered. Currently the blitting operation is implemented in three ways, using a “software” algorithm (SDL) [20] using OpenGL with the glDrawPixel command and OSG by rendering to a texture, which is used on a full screen quad. The Resource manager and context manager are handling the adaptation. The resource manager is registering the network bandwidth and, based on speed and packet throughput, determines the time needed to render the incoming data and the time needed for decompressing the image data. The context manager uses this information and generates, according to a set off heuristic rules, adaptation events. An event contains information about a function or action that has to be executed. An example of the heuristic rules is the following: based on network throughput the optimal size for a compressed image is calculated, then an event is generated and sent to the server. On the server side the event is forwarded to the compression module, which will adapt to the new required size by changing the compression parameters or even switch between compression algorithms. Fig. 7 shows a sample application using our proposed system.

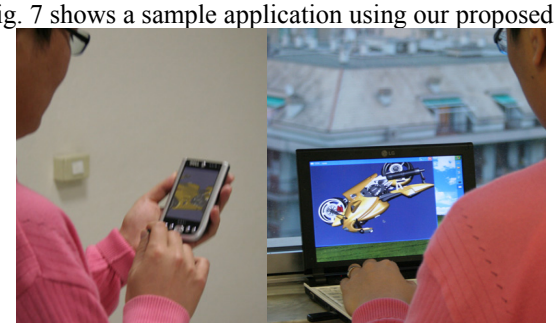

Fig. 7 Use case, left on a pda, right on a laptop.

B. Experiment Results

For the experiment we used a UMPC(Intel Celeron 900 Mhz, 512MB RAM, Intel graphics for mobile) and PC (Intel Duo-core, 2 GB RAM, NVidia Geforce7) as client devices connected to a high-end rendering server. The time measured is the response time in a local LAN environment. We used high quality medical 3D data for rendering. When we use our proposed algorithm the response time becomes faster, as shown in Fig. 8.

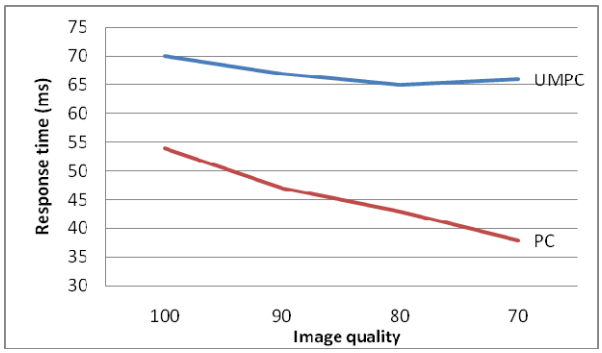

Fig. 8 The response for the UMPC and PC clients.

VI. Concluding Remarks

Interactive performance in terms of responsiveness is one of key challenging issues for interactive 3D applications. In this paper, we introduced run-time presentation adaption and dynamic interface adaptation mechanisms which aim to preserve the real-time interactive performance of 3D content, taking into account heterogeneous devices in user-centric pervasive computing environments. To support perceptual real-time interaction with 3D contents, temporal adjustment of presentation quality adaptation is used. It dynamically adjusts the quality of presentation on client devices according to the current device context. To overcome the inevitable physical heterogeneity in display capabilities and input controls on client devices, we provided a dynamic user interface reconfiguration mechanism for interaction with 3D contents. It can change the way how the interface is presented to the user (big screen or small screen bring several design issues with it) and adaptation to the user device input capabilities. In addition, functionality of 3D contents and rendering are dynamically bound with user interfaces at runtime according to profiles. We have built an initial prototype of our context-aware adaptive rendering system using Atlas framework for the handling the image/video stream and events, seamless session mobility mechanism in the network layer, and OSG as rendering engine. We also experimented with a specific scenario in user-centric pervasive environments. Experimental results show that the proposed system increased interactive performance of 3D contents.

Acknowledgements

The work presented was supported by the EU project INTERMEDIA (38419), in the framework of the EU IST FP6 Programme.

References

[1] Anastasi, G., Conti, M., Gregori, E., Pelusi, L., and Passarella, A., An Energy-efficient Protocol for Multimedia Streaming in a Mobile Environment, International Journal of Pervasive Computing and Communications, Vol.1, issue 4, pp. 301 - 312, 2005.

[2] Chang, S., Anastassiou, D., Eleftheriadis, A., and Pavlik, J., Video on Demand Systems: Technology, Interoperability and Trials, Kluwer Academic Publisher, 1997.

[3] Engel, K., Hastreiter, P., Tomandl, B., Eberhardt, K., and Ertl, T., Combining local and remote visualization techniques for interactive volume rendering in medical applications, proceedings of Visualization 2000, pp. 449 – 452, 2000.

[4] Eisert, P. and Fechteler, P., Low delay streaming of computer graphics, proceedings of the 15th IEEE International Conference on Image Processing, pp. 2704-2707, 2008.

[5] Frohlich, P., Simon, R., and Baillie, L., Mobile Spatial Interaction, Personal and Ubiquitous Computing, vol.13, issue 4, pp. 251 – 253, 2009.

[6] Han, S., Lee, D., Ko, I., A Deputy Object based Presentation Semantics Split Application Model for Synchronous Collaboration in Ubiquitous computing Environments, Proceedings of the third International Conference on Collaboration Technologies (CollabTech 2007), July 2007.

[7] Krasner, G. and Pope, S., A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System, Journal of Object-Oriented Programming, vol. 1, Issue 3, pp. 26 – 49, 1988.

[8] Krebs, A.M., Marsic, I., and Dorohonceanu, B., Mobile adaptive applications for ubiquitous collaboration in heterogeneous environments, proceedings of 22nd International Conference on Distributed Computing Systems, pp. 401- 407, 2002.

[9] Lee, D., Lim, M., Han, S., and Lee, K., ATLAS: A Scalable Network Framework for Distributed Virtual Environments, Presence: Teleoperators and Virtual Environments, Vol. 16, Issue 2, pp. 125-156, 2007.

[10] Lee, S., Ko, S., Fox, G., Adapting Content for Mobile Devices in Heterogeneous Collaboration Environments, Proceedings of the 2003 International Conference on Wireless Networks, June 2003.

[11] Preda, M., Villegas, P., Moran F., Lafruit, G., and Berretty R., A model for adapting 3D graphics based on scalable coding, real-time simplification and remote rendering, the Visual Computer Journal, vol. 24, pp. 881-888, Springer, 2008.

[12] Repetto, M., Mangialardi, S., Rapuzzi R., and Bolla, R., Streaming multimedia contents to nomadic users in ubiquitous computing environments, Workshop on Mobile Video Delivery in conjunction with InfoCom 2009, Rio de Janeiro, Brazil, 2009.

[13] Singhal, S., and Zyda, M., Networked virtual environments: design and implementation, ACM Press/Addison-Wesley, ISBN:0-201-32557-8, 1999.

[14] Verhoeven, R. and Dees, W., Defining services for mobile terminals using remote user interfaces, IEEE Transactions on Consumer Electronics, vol. 50, Issue 2, pp. 535 – 542, May 2004.

[15] Vlissides, J. and Linton, M., Unidraw: a framework for building domain-specific graphical editors, ACM Transactions on Information Systems, vol. 8, Issue 3, pp. 237 – 268, 1990.

[16] Weaver, K. and Parkhurt, D., Perceptually adaptive rendering of immersive virtual environments, Lecture Notes in Computer Science, vol. 4569, pp. 224 – 229, 2007.

[17] Reade, C., Elements of Functional Programming, Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., (1989), ISBN 0201129159.

[18] OpenSceneGraph, http://www.openscenegraph.org/

[19] RTSP, http://www.ietf.org/rfc/rfc2326.txt,

[20] Simple DirectMedia Layer , http://www.libsdl.org/