Enhanced Video Streaming for Remote 3D Gaming

Benjamin Prestele[1], Philipp Fechteler[2], Arto Laikari [3],Peter Eisert[4], Jukka-Pekka[5] Laulajainen [5]

[1,2,4]Fraunhofer Institute for Telecommunications, Berlin, Germany; [3,5] VTT Technical Research Centre of Finland, Espoo, Finland;

E-mail: {[3] arto.laikari, [5] jukka-pekka.laulajainen}@vtt.fi,

{[1] benjamin.prestele, [2] philipp.fechteler, [4] peter.eisert}@hhi.fraunhofer.de

Abstract: **Computer games are getting continuously more demanding in terms of CPU and graphics performance. At the same time, mobile lifestyle and the always-on connected experience are becoming increasingly important to many people. They want to enjoy digital entertainment anywhere inside their homes and also on the go. Therefore low cost and modest performance consumer electronics (CE) devices are gaining high popularity, but providing the same high quality gaming experience on those devices is still a real challenge. This paper describes a novel gaming system, called Games@Large, which enables heavy PC game play on low cost CE devices without the need of game software modification. The key innovations of the system are distribution of game execution, streaming of graphics, video, audio, and game control as well as network quality of service management.**

Keywords: games, graphics, rendering, video coding

1 INTRODUCTION

Electronic entertainment integrates more and more naturally into our always-connected life style: TVs connect to Internet video platforms and social networks, WiFi-connected electronic photo frames show the latest holiday pictures from our home computer, and new devices, such as the iPad, make web surfing a casual couch experience anywhere in the house or garden. Computer games are no exception in this regard. Games have been using network infrastructure already for a long time to keep players connected all over the world. However, with the advent of new gaming concepts, such as the motion-controlled Nintendo Wii or the Second Life virtual world, electronic gaming is transforming more and more from single-user play into a multi-player social experience. Modern games have become highly realistic and a wide population, not only youngsters, consumes them. Mobility and digital home entertainment appliances have additionally generated the desire to play not only in front of a home PC, but everywhere inside the house and also on the go.

As games have turned to be realistic connected virtual worlds, they have also become more demanding towards the computer hardware. High CPU processing power and graphics performance is required to play these games. As the result of TV evolution, Set-Top Boxes (STBs) have entered homes and mini-laptops have gained popularity. Several such low cost consumer electronic end devices (CE), e.g. electronic photo frames or iPods, are already available. Although these devices are capable to execute software, the latest 3D computer games are too heavy for them. The Games@Large system provides a unified solution for targeting a wide variety of end devices and enhances the usage of modest execution power CE devices also to heavy PC game playing terminals. Commercial games are supported without the need of modification and regardless of the used 3D API used by the game and the end device, since the Games@Large platform provides cross-streaming from DirectX to OpenGL and vice-versa. For devices without 3D graphics hardware, the game is rendered from a video stream.

The Games@Large project has developed a system for gaming both for homes and for enterprise environments, like hotels, Internet cafés and elderly homes [1]. .

Key concepts in the Games@Large system are execution distribution, audio and graphic streaming and rendering and decoupling of input control commands. Games are executed in one or more servers and the game display and audio is captured and streamed to the end device, where the stream is rendered and the game is actually played. Game control is captured at the end device and streamed back to the server and injected to the game.

There is a number of commercial Gaming on Demand systems, overviewed in [1], that have been presented to the market. More recently, there have been some new announcements about upcoming systems, such as Playcast Media Systems, Gaikai's Streaming Worlds technology, Onlive and GameTree.tv from TransGaming Technologies. However, there is very little detailed technical information publicly available about the commercial systems.

This paper will give a brief review of the overall architecture of the Games@Large system, presented in more detail in [2], and concentrate mainly on the novel advancements achieved recently in the accelerated video streaming techniques used in the project. In [1] advanced methods for video encoding suited for gaming were presented as a research proposal, and this paper presents more details of the implemented solution and recent research results.

Corresponding author: Benjamin Prestele, Fraunhofer Institute for Telecomm., Berlin, Germany, Benjamin.Prestele@hhi.fraunhofer.de

Figure 1: Input frame-buffer from game (left), corresponding depth map with blue colored skybox region (middle) and difference image of input frame-buffer and per pixel prediction based on previous frame (right).

2 GAMES@LARGE ARCHITECTURE

Three major element classes, as depicted inFigure 2, form the basis of the Games@Large system: servers, end devices and access points. Games are executed on the servers and played on the end devices. The Local Processing Server (LPS) runs the games and utilizes also the Local Storage Server (LSS). In the home version, these logical entities will be located on the same physical computer. In an enterprise version, the server entities are distributed over several physical computers. End devices, like Set-Top-Boxes (STB) or Enhanced Multimedia Extenders (EME), Enhanced Handheld Devices (EHD) and notebooks are connected to the server either with a wireless or wired connection.

The system exploits an adaptive streaming architecture and uses Quality of Service (QoS) functionalities to ensure good quality gaming to a variety of devices connected to the wireless network. The details of each component have been presented in earlier publications [1].

The Games@Large platform interacts with the games without the need to make any modifications to the application. This has the big advantage that there is no need to create specific Games@Large-designed games; instead almost any existing game can be used with the platform. The platform provides also a cross streaming option from DirectX to OpenGL and vice-versa, which makes it possible to play the games on end devices with various operating systems. Because the games are executed in the server, the end device does not need to spend the processing power, which the game would require, if it were running natively on the client.

The objective of the streaming architecture is to support various end devices with an efficient and high quality game experience, independent of software or hardware capabilities. To meet these demands, a streaming architecture has been developed that is able to support two streaming strategies to display the game remotely: graphics and video streaming.

Graphics Streaming is used for end devices with accelerated 3D graphics support, like desktop computers or set-top-boxes, typically having screens of higher resolution. Here the graphics commands from the running game are captured, transmitted and rendered locally on the end device. In this way, high image quality is achieved, since the scenes are always rendered for the desired screen resolution. Further details on Graphics Streaming can be found in [3].

The alternative approach, video streaming, is used mainly for end devices without a dedicated GPU, like handheld devices, typically having screens of lower resolution. Here the graphical output is rendered on the game server and the frame-buffer is captured and transmitted as H.264/AVC encoded video stream. The bit rates in Video Streaming are in general much more predictable in comparison to Graphics Streaming. However, H.264/AVC encoding on server side is computational demanding and needs to be accomplished in parallel to one or more game sessions running on the same server. Several techniques have been developed in this project to accelerate the H.264/AVC encoding of games and are described in the next sections.

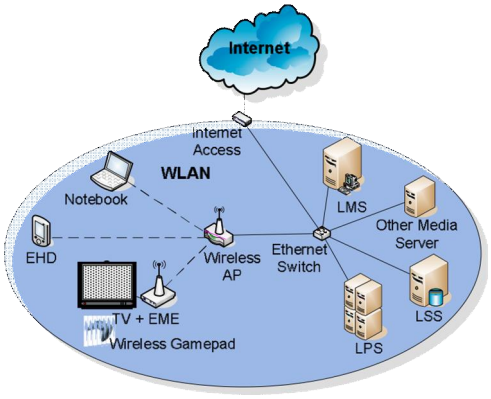

Figure 2: Games@Large Architecture.

3 ACCELERATED VIDEO STREAMING

The core idea of the implemented H.264/AVC encoding acceleration techniques is to exploit additional information accessible from the game's render context. For this purpose the original graphics library is overloaded by a proxy library (see [4] for details on this method for OpenGL on Linux), thereby eliminating the need to modify the game. As each graphics command

issued by the game passes through the proxy library, the commands are intercepted and parameters are adapted to respect the specific capabilities of the client: for example the viewport is adapted to the client's screen size, thereby enforcing rendering at the native resolution of the end device. This is in contrast to other adaptive streaming techniques, such as [5], where the visual output is changed only after rendering. The Games@Large approach in turn reduces the computational load and processing delay on the server and provides maximum image quality since e.g. no rescaling is required.

Additionally, the interception of drawing commands makes it possible to analyze and exploit knowledge about the scene geometry in order to reduce the complexity of the computationally most expensive part in H.264/AVC video coding, which is motion compensation. In this process, the encoder exploits spatial and temporal redundancies by predicting small areas (macroblocks) in the current video frame from already encoded pixels in the current or previous frames. If a good match is found, a macroblock can be encoded very efficiently, using a motion vector pointing to the matched area and the residual error from this prediction. For maximum encoding efficiency, H.264/AVC additionally allows for the use of variable block sizes, i.e. each macroblock may be split into several partitions, with each partition using a different motion vector for prediction. The motion compensation process is therefore a very time-consuming search procedure and the encoder may test many different combinations until it finds the most efficient variant for encoding. To reduce complexity of this step, the H.264/AVC video encoder in the Games@Large system has been enhanced to exploit the graphical context for the direct calculation of motion vectors and to enforce a particular partitioning in the so-called skybox parts of the rendered scene (see Section 3.2). The resulting H.264/AVC bit stream is fully standard compliant and can be played with any compliant video player.

3.1 Efficient Direct Calculation of Motion Vectors

In order to calculate motion vectors from the graphical context, projection information for corresponding scene objects in successive frames is required. For this purpose, the projection matrices of all intercepted drawing commands are captured, stored and analyzed. Our investigations have shown that major portions of a scene will typically be rendered with very few different projection matrices. By choosing only the most-used projection matrices, motion vectors can be predicted efficiently for large parts of the rendered frame. A threshold on the rate-distortion-cost is used to detect cases, where this prediction is wrong as scene elements were rendered using a different projection or 2D overlays cover the 3D scene. In this case, the enhanced encoder will fall back to the generic H.264/AVC motion search.

As detailed in [6], to directly calculate the motion vector for a given macroblock (partition), the corresponding 3D location is computed by back-projecting the centre location of the macroblock, using screen coordinates and the depth buffer value together with the projection matrices from the current frame. The resulting 3D location is then re-projected to the pixel location in the previous frame, this time using the projection matrices of the previous frame. The motion vector is finally calculated as the difference of both 2D locations, scaled by four, to account for quarter-pixel resolution of H.264/AVC.

In contrast to [7], we do not use gluUnProject() and gluProject() of the OpenGL Utility Library, nor their counterparts in DirectX, i.e. D3DXVec3Unproject() and D3DXVec3Project() for projection and back-projection. Instead, we have implemented the motion vector calculation by assembling all pixel independent calculations, including normalization, matrix-inversion and matrix-matrix-multiplication into a single matrix, which is constant for the entire frame. As a result, the calculation of a motion vector MV between two corresponding pixels p_{cur} and p_{prev} in the current and the previous frame, respectively, is reduced to a single multiplication of a 3 × 4-matrix with a 4-vector:

$$p_{prev} = N^{-1}P_{prev}P_{cur}^{-1}Np_{cur} = Mp_{cur}$$

$$MV = 4\cdot(p_{prev} - p_{cur}) = 4\cdot(M-1)p_{cur} = \overline{M}p_{cur}$$

with N being the matrix for normalizing screen coordinates to [-1, 1] space and P being the matrix containing all projection information (model-view, projection, world transform). All these pixel independent computations are finally merged into a single matrix $\overline{M}$, which also compensates for the different structure of OpenGL and DirectX specific matrices P and N, as well as for the different coordinates systems being used.

Care has to be taken for motion vectors pointing outside of the frame, e.g. by falling back to generic H.264/AVC motion search algorithms. Uncovered regions or object boundary crossings could be detected by comparing the calculated depth value for the previous frame and the measured one. Our experiments have shown though that handling these effects explicitly does not improve the overall performance and are already handled well by the rate-distortion-cost threshold mentioned above.

An example of a per-pixel prediction is given on the right of Figure 1: pure green indicates a perfect prediction while lighter and darker green show prediction errors. Please note that the blue area is treated differently, as it contains the so-called skybox region, which is discussed in the following section. The acceleration achieved by integrating the direct motion vector calculation into the H.264/AVC encoder can be seen inFigure 3: compared to (non-optimized) generic encoding, the direct calculation of motion vectors for the main scene part (labelled "scenePred") speeds up encoding for the given test sequences by 8-10%, while increasing bit rate by typically only slight amounts. Please note that this optimization does not consider the skybox region, which will be detailed below.

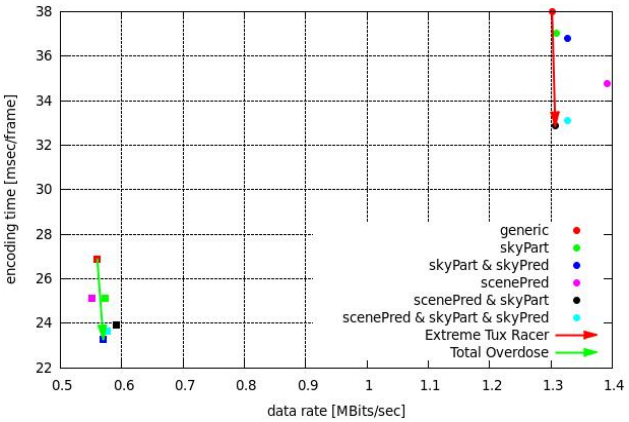

Figure 3: Results of H.264 encoding based on skybox and scene enhancement

3.2 Skybox Optimization

Many 3D computer games use a rendering technique called skybox/skysphere to draw the far away scene environment, such as sky and mountain panoramas. Skybox regions are commonly rendered at the beginning of a frame-buffer update and depth testing is typically disabled for those parts of the image. This permits to unambiguously detect skybox regions by checking the z-buffer values, as can be seen by the blue region in the middle ofFigure 1. As discussed in more detail in [8], the use of this particular rendering technique can be exploited to achieve acceleration in H.264/AVC encoding. Specifically, since skybox regions are homogenous moving textures, corresponding macroblocks need not to be split into smaller partitions, and assigning one common motion vector for this region is sufficient. Further checks for partitions with finer granularity can therefore be omitted for macroblock partitions residing completely within skybox regions. The technique to calculate direct motion vectors is the same as already discussed above, but using the projection information for the skybox region. In comparison to the generic H.264/AVC macroblock partitioning and motion search process, this can accelerate encoding for large portions of the image.

Results of a per pixel prediction in the skybox region is shown as blue area on the right of Figure 1. In this example, prediction errors occur in only small areas, where foreground scene objects overlay the skybox, and naturally at the border to non-skybox regions, where previously occluded foreground areas get uncovered. The impact on coding complexity from exploiting the skybox region and corresponding projection information for macroblock partitioning and direct motion vector calculation is depicted in Figure 3. In combination with direct motion vector calculation for the non-skybox parts, coding complexity can be reduced by approximately 10-25% with our test sequences, depending on the size of the skybox region in the rendered image.

3.3 Client for Low Delay Video Streaming

The H.264/AVC encoded video is then transmitted together with AAC audio to the client, using standard compliant RTP over UDP streaming protocols. While any standard compliant client software will be able to play back the streams, most media players do not fulfil the requirements of very low delay decoding and rendering. They are usually implemented with the focus on smooth and reliable playback of accurately synchronized audio and video, even in the case of heavy network jitter. Most players therefore use generous buffering at many stages of the processing chain, which in turn leads to typical delays of approx. 100ms to few seconds. Such large delays on the client side would obviously not be acceptable for gaming and also cancel out much of the effort put into low delay encoding and streaming on the server side and the network protocol layer.

However, the specific scenario and design of the Games@Large system allows relaxing some of the aforementioned restrictions. Firstly, the network is a dedicated and controlled environment with QoS optimizations put in place to reduce network jitter and delay. Secondly, many games use sound only for background music or to provide aural feedback to user interactions, which usually is not vital to the game's functionality. Since audio and video streams are encoded and transmitted with very low delay, and since there is no increasing drift between the streams over time, we found it suitable for this scenario to waive perfect synchronization in favour of instantaneous playback.

Custom client software has been implemented for this purpose, relaxing on the synchronization requirements and focusing on minimal buffering and instant playback of video and audio instead. While each video frame is being decoded and displayed immediately after the required data packets have arrived at the client, the decoded audio frames undergo an additional pre-processing step prior to being rendered on the sound card. This showed to be necessary, as even small temporal interruptions of the audio bit stream may result in very distracting clicking noise in the rendered audio. Therefore an adaptive retiming algorithm is being applied to the yet unplayed audio samples, which smoothly stretches or compresses available samples to reflect the current arrival speed of new audio samples. This allows the client to compensate for low to medium jittering on the network with almost no noticeable distortion of the audio, while on average still maintaining minimal buffering and perceived delay.

4 CONCLUSION

Games@Large has implemented an innovative architecture, transparent to legacy game code, which allows distribution of a cross-platform gaming and entertainment on a variety of low-cost networked devices. Virtually extending the capabilities of such devices, the Games@Large system is opening important opportunities for new services and experiences in a variety of fields and in particular for the entertainment in the home and other popular environments. This paper has presented the advancements in the field of video encoding optimized for game use.

Acknowledgements

This work has been carried out in the IST Games@Large project (http://www.gamesatlarge.eu) , which is an Integrated Project under contract no IST038453 and is partially funded by the European Commission.

References

[1] A. Jurgelionis, P. Fechteler, P.Eisert, et al., “Platform for Distributed 3D Gaming”, in Intern. Journal of Computer Games Technology, Article ID 231863, 2009

[2] A. Laikari, P. Fechteler, P. Eisert, A. Jurgelionis, F. Bellotti, A. De Gloria, “Games@Large Distributed Gaming System”, in Proc. of Networked & Electronic Media Summit (NEM2009), Saint-Malo, France, Sept. 2009

[3] P. Eisert, P. Fechteler, “Low Delay Streaming of Computer Graphics”, in Proc. of IEEE Int. Conf. on Image Processing (ICIP2008), San Diego, USA, Oct. 2008, pp. 2704-2707

[4] S. Stegmaier, J. Diepstraten, M. Weiler, T. Ertl, “Widening the Remote Visualization Bottleneck,” in Proc. of 3rd Int. Symposium on Image and Signal Processing and Analysis (ISPA2003), Rome, Italy, Sept. 2003, vol. 1, pp. 174– 179.

[5] J. Brandt, L. Wolf, “Adaptive Video Streaming for Mobile Clients”, in 18th Intern. Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV2008), Braunschweig, Germany, May 2008

[6] P. Eisert, P. Fechteler, “Accelerated Video Encoding Using Render Context Information”, in Proc. of IEEE Int. Conf. on Image Processing (ICIP2010), Hong Kong, China, Sept. 2010

[7] L. Cheng, A. Bhushan, R. Pajarola, M. El Zarki, “Realtime 3D Graphics Streaming Using MPEG-4,” in Proc. of IEEE/ACM Workshop on Broadband Wireless Services and Applications (BroadWise ’04), pp. 1–16, San Jose, Calif, USA, July 2004.

[8] P. Fechteler, P. Eisert, “Depth Map Enhanced Macroblock Partitioning for H.264 Video Coding of Computer Graphics Content,” in Proc. of IEEE Int. Conf. on Image Processing (ICIP2009), Cairo, Egypt, Nov. 2009, pp. 3441-3444