Open access • Proceedings Article • DOI:10.1109/ISCE.2008.4559473

Games@large graphics streaming architecture — Source link

I. Nave, Haggai David, A. Shani, Y. Tzruya ...+3 more authors

Institutions: VTT Technical Research Centre of Finland, Heinrich Hertz Institute

Published on: 14 Apr 2008 - International Symposium on Consumer Electronics

Topics: DirectX, Systems architecture, Software architecture, Bill of materials and IPTV

GAMES@LARGE GRAPHICS STREAMING ARCHITECTURE

Itay Nave, Haggai David, Alex Shani — Exent Technologies Ltd.
Arto Laikari — VTT
Peter Eisert, Philipp Fechteler — Fraunhofer HHI

ABSTRACT

In coming years we will see low cost networked consumer electronics (CE) devices dominating the living room. Various applications will be offered, including IPTV, VoIP, VoD, PVR and others. With regards to gaming, the need to compete with PlayStation and Xbox will require a radical change in system architecture. While traditional CE equipment suffers from having to meet low BOM (bill of materials) targets, dictated by highly competitive market and cable companies targeted costs, consoles enjoy superior hardware and software capabilities, being able to offset hardware and BOM costs with software royalties. Exent Technologies is leading the European FP6 Integrated Project Games@Large , whose mission is to research, develop and implement a new platform aimed at providing users with a richer variety of entertainment experience in familiar environments, such as their house, hotel room, and Internet Café. This will support low-cost, ubiquitous game-play throughout such environments, while taking advantage of existing hardware and providing multiple members of the family and community the ability to play simultaneously and to share experiences.

This paper focuses on one of the innovative aspects of the Games@Large project idea – the interactive streaming of graphical output to client devices. This is achieved by capturing the graphical commands at the DirectX API on the server and rendering them locally, resulting in high visual quality and enabling multiple game execution. In order to support also small handheld devices which lack hardware graphics support, an enhanced video method is additionally provided.

INTRODUCTION

Computer games constitute nowadays one of the most dynamic and fastest changing technological areas, both in terms of market evolution and growth and technology development [1]. Market interest is now revolving around capitalizing on the rapid increase of always-on broadband connectivity. Broadband connection drives to a new, digital, "Future Home" as part of a communications revolution, which will affect every aspect of consumers' lives and change the way consumers enjoy entertainment. Taking into account that movies and music provided by outside sources were at home long before the Internet and Broadband, the challenge is to invent new content consumption patterns of existing and new types of content and services.

Games offer a leisure time activity for every member of the household – from avid gamers to kids, as well as allowing whole families to play together.

Running an interactive content rich multimedia application (such as video games) requires high performance hardware available on a PC or on a dedicated gaming device. Other devices such as Set Top Boxes (STB) or Handheld Devices (HD) lack the necessary hardware and adding such capabilities to these devices will cause their prices to become prohibitive.

The Games@Large project intends to develop a system that is not available yet in the market that enables rendering of PC games on the next generation STB devices without causing a significant increase in their price.

In general, the system will execute games on a server PC, located at a central site or at home, capture the graphic commands, stream them to the STB and render the commands on the STB allowing the full game experience. For end devices that do not offer hardware accelerated graphics rendering, the game output is locally rendered at the server and streamed as video to the client [7]. Since computer games are highly interactive, extremely low delay has to be achieved for both techniques. This interactivity also requires the controller's commands to be captured on the STB and streamed to the server in order to inject them into the game process.

The Games@Large platform implements all the technologies required for that process focusing on:

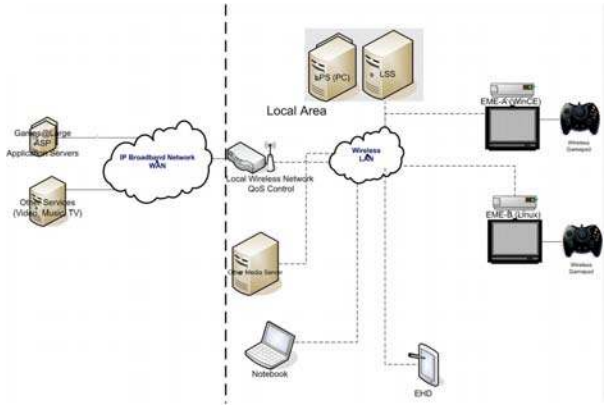
- Graphic commands capturing on the server, streaming and rendering on the STB.
- Low delay video streaming
- Audio streaming
- Running multiple games on the server and managing computer and games resources.
- Controller commands capturing on the end device, streaming to the server and injection into the game process
- Performance optimizations in order to achieve a great gaming experience.

PC games are typically executed on Windows platforms (XP, Vista) with DirectX API and require a high performance CPU and graphics hardware.

The Games@Large supported end device requirements are:

- Graphics hardware: According to research performed to date, and as described in the Games@Large architecture, we believe that a relatively low bill-of-material graphics hardware component can be added to such consumer electronics devices.
- Rendering capabilities: OpenGL or DirectX.
- Limited CPU power

GAMES@LARGE STREAMING ARCHITECTURE



The Games@Large streaming architecture includes the following main components.

1. Storage Server: The storage server holds all the game files as well as user specific files.
2. Processing Server (LPS): The Processing Server is responsible to launch the game process, manage its performance, allocate computing resources, file system and I/O activities and capture the game graphic commands (or, alternatively, the rendered outpur) as well as managing execution of multiple games. The processing server is further responsible for receiving the game controller commands from the end device and injecting them into the game process. The processing server is also responsible to stream game audio to the client.
3. Graphic Streaming Protocol Stack: The Graphics Streaming Protocol is intended to become a standard protocol used in order to stream 3D commands to an end device allowing lower performance devices such as STB to present high performance 3D applications such as games without the need to actually execute the games on this device. The protocol is focused on the performance optimizations required.
4. Client: The client module is responsible for receiving the 3D commands and rendering them on the end device using local rendering capabilities. For the video streaming approach, H.264 decoding must be supported instead. The client is also responsible for capturing the controller commands and transmitting them to the processing server.

GRAPHICS STREAMING

The Graphics Streaming is the core of the Games@Large innovation. The protocol allows the decoupling of the rendering environment of 3D applications from its original execution environment on a Windows PC to the next generation end devices such as STB and mobile devices.

The graphics streaming provides:

- Full graphics: Unlike video streaming, graphics streaming maintains the original hi-def image quality.
- Low bit rate: Proof of concept implementations of the graphics streaming protocol demonstrate low bit rate and good user experience. The product will implement various types of compression and prediction algorithms that will allow full gaming experience with low bit rates and reduced latency.
- Reduced latency: Since DirectX API is a synchronous API, the protocol stack on the server side translate it into an asynchronous API allowing reduced latency and streaming of commands.
- Real time translation from the captured DirectX commands on the server to OpenGL commands that can be rendered on a variety of end devices (such as Linux based systems).
- Protocol Definition: One of the outcomes of the Games@Large project is a definition of a standard protocol that can be implemented on a variety of end devices in order to support the graphics streaming protocol.

Streaming and remote rendering are achieved by multiple encoding and transmission layers, first of which is the interception and the very last one is the rendering on the client machine. All layers in between these two are independent of any specific graphics API. The latter implies that the 3D data streamed post-interception till the client rendering process is not specific to either DirectX or OpenGL, but rather utilizes higher-level concepts common to all 3D graphics.

Since efficient direct translation from DirectX API commands to OpenGL commands is difficult due to the significant differences between these APIs, a set of common generic concepts may be of assistance. In general, a 3D scene consists of multiple objects that are rendered separately. Before rendering an object, several parameters (states) must be set, and these include lighting, textures, materials, the set of 3D vertices that make a scene (built out of multiple objects), and further various standard 3D transforms (e.g., translate, scale, rotate).

The game running on the LPS is provided with a pseudo-rendering environment that intercepts the DirectX

calls. Instead of executing them on the server, the commands are passed to the next layer – the graphics states book-keeper. That layer keeps track of the object currently being rendered and of its states. When the game/application has provided all necessary data for drawing an object/scene, the states tracking layer passes it further to the brute-force (e.g., ZIP or LZO) compression and from there to the network.

On the client side, after the data has been received and decompressed, it is passed to the rendering component. This component is the last in the 3D streaming pipeline. It receives the states data of a 3D object and renders that object using the desired graphics API of the client. There is a certain overhead in OpenGL rendering because some data (especially color and vertex data) must be reorganized or rearranged in the processing stack before it can be given to OpenGL for rendering. This may result in increased demand of CPU processing and memory transfer between system memory and the GPU.

Modern computers allow huge (relatively to network) data transfer rates between the game running locally and the video card. In our case, instead of the video card, all that data is sent over a network to the client machine with limited bandwidth and latency; hence, some optimizations are necessary in order to decrease the network load.

The most obvious optimization is based on the fact that video games often set the same state more than once for the same object and only the last change before the rasterization is the one that affects the image. So, only the last change in such case is sent over the network.

Further, measurements on games have shown that large part of the network traffic is caused by the data needed to update the object's list of vertices when it changes shape (human or animal motion, explosions etc.). In order to reduce the amounts of this data, various methods have been investigated. The most promising is reducing the resolution of vertex or texture coordinates, prediction and caching of the vertex data on the client and sending only the vertices that have changed since the last rendering of the object.

The tests performed include multiple runs of games from various genres: First Person Shooter, Business Strategy, Car Racing and some Casual games. The selection of the games made it possible to monitor how different types of games influence bandwidth used and frame rate (FPS) on the client side.

Further, different clients have been investigated: a PC (Pentium-4 3.6GHz, 2GB RAM, NVIDIA GeForce 6600GT video card) running Microsoft Windows XP, same PC running Debian Linux and a Handheld PC (Samsung Q1Ultra portable PC, 800MHz CPU, 1GB RAM, Intel 945gm video chipset with 128MB shared memory) running Linux Ubuntu. The server and client were connected using a 100Mbps LAN.

We measured FPS as a measurable value for gaming experience on the client side. The network usage and utilization was measured also to benchmark network parameters required for 3D streaming.

Typical results of test runs of different games on different client machines are shown below. The tables show average session values for the frame-rate (FPS) and for data sent through the network. The values in parenthesis are the minimum and maximum peaks.

Sprill (casual game)	FPS	Data Sent (Kbytes/sec)
Linux PC	90-120 (60)	2200-2600 (3000)
WinXP PC	170-200 (100)	4300-5500 6000)
Handheld PC	12-18(8)	240-450(620)

Zoo Tycoon (business strategy)	FPS	Data Sent (Kbytes/sec)
Linux PC	35-70(20)	500-1500 (2500)
WinXP PC	60-80(15)	1500-3500(3900)
Handheld PC	7-11(7)	240-450(620)

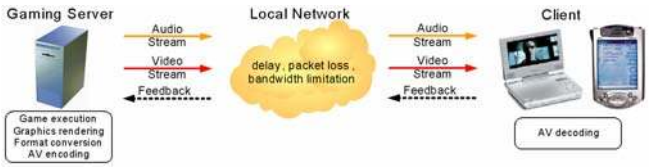
Total Overdose (shooter)	FPS	Data Sent (Kbytes/sec)
Linux PC	7-20(4)	1300-2500(3700)
WinXP PC	9-27(6)	1000-3000(3700)
Handheld PC	*	

* The tablet PC lacks the hardware acceleration features needed to run this game.

The measurements apply to in-game stages, as during loading, lower FPS and higher net loads are observed, lasting a few seconds till the game finish loading. It can also be observed that in general higher FPS increase net load and that the OpenGL conversion lowers FPS by approximately 25-35%.

During game-play, reduction in FPS is observed when there are more viewable objects in the scene or when the scene is changed dramatically (such as explosions). FPS peaks are observed when the scene is relatively simple or when the menus are being displayed.

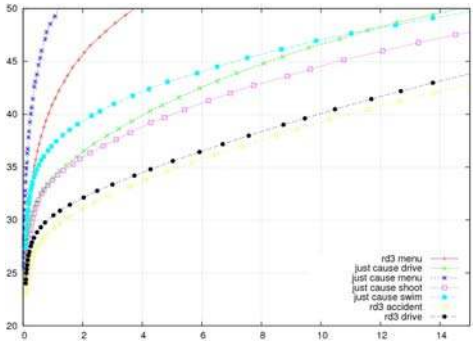
VIDEO STREAMING



Although the graphic streaming approach is the preferable solution since it offers lower latency and enables execution of multiple games on one server, it cannot be used for some small handheld devices like PDAs or smart phones. These end-devices typically lack the capability of hardware accelerated rendering and cannot create the images locally for displaying them. Therefore, we also

provide a fallback solution using video streaming techniques [3,5,6]. Here, the server renders the computer graphics scene, the frame buffer is captured, eventually down-sampled to match the target resolution, and the current image is encoded using a video codec. For video encoding we use H.264 [2], which poses the current state of the art in this domain. Decoding video is usually computationally less demanding and can be performed even on small devices. Furthermore, hardware acceleration for H.264 decoding can be found on an increasing amount of end devices. Also, the bit-rate for streaming the graphics output is rather predictable and not fully influenced by the complexity of the graphics scene.

Streaming games' output, however, is somewhat different than streaming real video. First, the synthetic content is usually free of noise, shows also high spatial frequencies and has different statistics compared to real video. The following plot shows the RD performance measured in PSNR [dB] over bit-rate [Mbit/s] for different games (first person shooter: Just Cause and car racing: Race Driver 3) in 4CIF resolution and 30 FPS. Different scenes with different amount of motion and scene changes (explosions vs. menus) result in large differences in bit-rate.



The requirements on low delay are also much higher in contrast to video broadcast in order to enable interactive gaming. Typically, users require round trip delays of below 100ms for first person shooter while other games (e.g. strategy, casual, etc.) have less restrictive constraints. For enhaced gaming experience, we investigate new methods to reduce the end-to-end delay by carefully selecting data dependencies and intelligent error concealment. In order to allow encoding in parallel to the execution of a computer game, computational complexity is another important issue, especially in case of several games running on one game server. However, additional information about the scene is available from the render context, like camera data or motion and depth information. We exploit that information to reduce the complexity of the H.264 encoding in order to combine high coding efficiency with lower encoding complexity.

In order to deliver the sound of the game, audio samples are captured at the server and encoded with High Efficiency Advanced Audio Coding Version 2 (HE-AAC v2) [4]. Both, video and audio are streamed from the server to the client by utilizing the Real-Time Transport Protocol (RTP, RFC 3550, 3640, 3984). For synchronized playback of audio and video, the Real Time Control Protocol (RTCP, RFC 3550) is used.

CONCLUSIONS

In this paper, we have presented a new architecture for remote game-play. One core component is the streaming of 3D graphics commands over local area networks. Real-time encoding and intelligent caching allows for interactive frame rates at high visual quality. A generalized protocol supports end devices with both OpenGL and DirectX API's. Clients which lack hardware accelerated rendering can also be used with the video streaming approach. The H.264-video and HE-AAC-v2-audio streams are fully standard compliant and enable high efficiency. Enhancements at the encoder address low delay and low encoding complexity.

REFERENCES

[1] Yoav Tzruya, Alex Shani, Francesco Bellotti, Audrius Jurgelionis. **Games@Large - a new platform for ubiquitous gaming and multimedia**, Proc. BBEurope, Geneva, Switzerland.

[2] MPEG-4 AVC (2003). **Advanced video coding for generic audiovisual services**, ITU-T Rec. H.264 and ISO/IEC 14496-10 AVC.

[3] S. Stegmaier, M. Magallon, T. Ertl, **A generic solution for hardware-accelerated remote visualization**. Proc. of the Symp. on Data Visualisation, 2002.

[4] MPEG-4 HE-AAC, ISO/IEC 14496-3:2005/Amd.2

[5] S. Stegmaier, M. Magallon, T. Ertl, **Widening the Remote Visualization Bottleneck**. Proc. of the 3rd Symp. on Image and Signal Processing and Analysis (ISPA), 2003.

[6] L. Cheng, A. Bhushan, R. Pajarola and M. Zarki, **Real-Time 3D Graphics Streaming using MPEG-4**, Proc. IEEE/ACM Workshop on Broadband Wireless Services and Applications,. ICS-UCI, 2004.

[7] P. Eisert and P. Fechteler, **Remote Rendering of Computer Games**, Proc. Intern. Conf. on Signal Processing and Multimedia Applications (SIGMAP), Barcelona, Spain, July 2007

[8] I. Buck, G. Humphreys, P. Hanrahan, **Tracking graphics state for networked rendering**. In Proc. SIGGRAPH / EUROGRAPHICS Workshop On Graphics Hardware, 2000.

[9] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P.D. Kirchner, J. T. Klosowski, **Chromium: a stream-processing framework for interactive rendering on clusters**. In Proc. Intern. Conf. on Computer Graphics and Interactive Techniques, 2002.

[10] U.S. Patent Application No. 11/204,363, entitled "System and Method for Providing a Remote User Interface for an Application Executing on a Computing Device" and filed August 16, 2005.