Visualization of Multidimensional Data on distributed mobile devices using interactive video streaming techniques

M. Pańka[1], M. Chlebiej[2], K. Benedyczak[2,3] and P. Bała[2,3]
[1] UCNTN, Nicolaus Copernicus University, Torun, Poland
[2] Faculty of Mathematics and Computer Science, Nicolaus Copernicus University, Torun, Poland
[3] ICM, University of Warsaw, Warsaw, Poland
maciej.panka@umk.pl

Abstract - Remote visualization of large datasets has been a challenge for distributed systems for a long time. This challenge is getting even bigger when visualization refers to devices with limited capabilities, like CPU and GPU power, number of RAM or screen size. In this paper we present a distributed system we have developed for interactive visualization of remote datasets on variety of modern mobile devices, including laptops, tablets and phones. In our system all the data are rendered on dedicated servers, compressed on-the-fly using a video codec and pushed to client as a single video stream. Based on this model we have taken off most of the computational power from client's devices, leaving them with a video decompression. We were also able to achieve very high frame rates and video quality, dynamically adapted to device capabilities and current network bandwidth of a client. Our system can be used with almost any kind of data, including 2D, 3D and even animated 3D data. All of them are being processed in real time based on user inputs, with minor latency, allowing interactive visualization. At the end of this paper we also present some preliminary results of system performance gained using sample, multidimensional medical datasets.

I. INTRODUCTION

Many scientific experiments, simulations and measurements of today produce large amount of digital data, usually stored in different formats specific to each discipline. Most of these scientific activities are very often generated by dedicated laboratories, technically prepared to handle large data volumes. For example, numerical computations are usually so complex that they must be realized on dedicated clusters. On the other hand, some empirical experiments require long-term measurements by means of variety of digital sensors connected to the storage system.

Obtained data are usually so large that they cannot be easily transferred between systems and have to be stored centrally in a place where they were generated. On the one hand this model simplifies data management, browsing and searching, but also can cause many difficulties, mostly because of the size of the data. One of the biggest challenges is remote visualization of large data without need of downloading the whole sample to a local machine. Internet visualization can be difficult not only because of the size of remote data, but also because of computational power needed to process the information in real time.

There are many effective ways to visualize remote 2D data, like for example compression or image streaming, but the problem is getting more complex when it comes to 3D or animated 3D data. Every frame of a 3D object consists of thousands of pixels, while in animations these frames additionally evolve in time, giving millions of elements that have to be visualized on remote machine in real time. Desktop computers and cable networks are already advanced enough to process these data, although if volumes are large there still could be some sort of latency in client-server communication, even if compression is used. The problem is getting much bigger when visualization is done on mobile devices with the use of a wireless connection. Even though modern laptops, tablets and cell phones have gone a long way in the last couple of years, they still have many limitations compared to desktops computers. The most important are CPU and GPU power, number of RAM, small screen sizes or battery capacity. Moreover, wireless networks, especially 3G standard, are still much more vulnerable on interferences than cable connections. Both limitations cause huge drawbacks in adapting interactive visualization techniques to mobile devices.

In this paper we propose a sample approach to the mobile visualization problem by means of interactive video streaming techniques. The whole system consists of two parts: a distributed server side application and a client program which runs on mobile device. Server modules are responsible for handling user inputs, data processing, video encoding and transferring compressed stream through the Internet. The client's applications' only job is to receive these data, decompress them and display on the screen. This model takes off most of computational power from mobile users, allowing thereby running a client application even on very thin devices. Based on this architecture we were able to achieve very high levels of video quality and frame rates. We were also able to significantly reduce network latency, which makes remote visualization interactive for mobile users. The general idea has been explored in the past by various

research groups but no effective implementation has been yet developed.

The rest of this paper is organized as follows. Section 2 briefly overviews related works covering remote visualization of large datasets. In the section 3 we are describing system architecture, including technologies that we have used for its implementation. In the section 4 we are presenting some preliminary results of system performance obtained using medical datasets. The last section concludes the paper and draws up further work.

II. RELATED WORK

There are many different approaches to the remote visualization problem. One of them is the idea of building dedicated visualization rooms where users can manipulate distance data in an interactive and collaborative way [1]. These rooms are always equipped with dedicated hardware and software, and have access to the broadband Internet connection. The biggest inconvenience of this approach is expensiveness and lack of mobility. With a variety of mobile devices commonly available today, users are used to have unlimited access to global information resources and they expect the same from the visualization system. Therefore one of the biggest challenges of today is to develop more ubiquitous solutions for remote visualization.

Existing techniques for remote visualization can be divided into four main categories. The first one is image streaming technique. In this model all the data are rendered remotely on a dedicated server and are sent to user as a series of digital images. Every modern mobile device is able to read commonly used digital image formats. To reduce network bandwidth it is recommended to use some sort of compression methods instead of sending RAW data. An exemplary approach to this problem would be making use of one of available algorithms, like ZLIB, LZO, BZIP2 or RLE [2, 3]. Some authors are additionally choosing compression algorithm dynamically, depending on current data source [2].

On the other hand [3] suggests using dedicated image compression algorithms, like JPEG or JPEG 2000, which has already become one of the most popular standards on the Internet. D. Dragan and D. Ivetic introduced an exemplary system which derives from this model to visualize remote medical images on mobile devices [4]. Based on JPEG 2000 standard the authors were able not only to increase compression level, but also to improve overall system efficiency with the use of a JPIP image streaming technique. Image streaming is a great solution when data are visualized on the mobile devices, which are usually equipped with very small screens. Image based visualization could also be adapted to visualize 3D and animated 3D data with the use of Motion JPEG 2000 standard [5], although in this area there are available more effective techniques.

The second approach to remote visualization is to make use of 3D objects streaming [6]. In this concept, dependent on a chosen algorithm, single objects or whole scenes are progressively sent to users' devices where they are successively rendered in real time. This model allows for user interaction with remote data even without the need of waiting for an entire scene to be downloaded. To reduce network traffic it is also possible to replace the client-server communication with peer-to-peer connections, which in some situations can increase an overall efficiency [7].

Unfortunately, in most cases the computational power required to render 3D scenes excludes this technique from being effectively used on the mobile devices because of the lack of enough CPU and GPU resources. It is also difficult to adopt this method into scientific visualization, mostly because of large sizes of data volumes, which have to be transferred to clients' devices without any latency.

A much better way to visualize scientific 3D data on the mobile devices is use of Virtual Reality Modeling Language (VRML). The VRML is a standard for representing 3D scenes using text files, where every object can be described by means of specially formatted markups. This approach was effectively used to visualize data on the mobile devices [8, 9]. K. Engel and T. Ertl showed how performance of this technique could be additionally improved with the use of GZIP compression and advanced data clipping algorithm [10]. Even though the VRML can be successively adapted to mobile devices, it still has some major limitations constraining visualization interactivity, mostly because of its network bandwidth consumption and computational power needed to process the data.

The fourth category of the remote visualization techniques involves systems that use a video codec to compress the rendered data. In this model everything is rendered on dedicated servers, compressed on the fly and broadcast to users as a single video stream. This solution does not involve objects rendering on the clients' devices, leaving them only with video decompression. Most of the modern mobile devices have built-in support for common video formats available today. That is probably one of the main reasons why recent videos streaming techniques became a very natural and promising solution to the remote visualization problem.

An exemplary system implementing this model was introduced in [11]. The authors presented a framework for Open Inventor application, which allows interactive visualization of remote datasets using a video codec. This solution has been additionally extended in [12] where most of the data were generated on a client's device using a low profile configuration, and only the last frame was rendered server-side and sent to a client in higher resolution. A generic solution to this problem by means of GLX, dynamic linking and VNC protocol has been proposed [13]. Thanks to using one of the commonly available VNC clients their system could still cooperate with most of today's mobile devices. Unfortunately, one of the biggest inconveniences of their system was poor frame rate level, which had bad impact on visualization interactivity.

According to the current studies it seems that a much better solution would be to make use of a dedicated video codec for data compression. With the use of a video

codec every frame in a sequence is encoded using some sort of prediction, relevant to a chosen algorithm. This technique allows achieving much better quality, frame rate and compression level of an output stream. The video sequence is afterwards streamed to a client's device, where it is decoded and displayed on a screen. Sample applications based on this model are described in [14], where authors used an H.261 standard to encode the data. Another approach to this problem would be to use an MPEG-4 format [15, 16], which is also supported by many mobile devices available today. Unfortunately, one of the main problems with the MPEG-4 is its very complex motion estimation algorithm, which lengthens the encoding time and can have a very bad impact on system interactivity.

The solution we propose in this paper derives directly from the fourth category of remote visualization techniques. Being aware of potential problems we have decided to use much less power consuming video codec and distribute most complex tasks between different servers. Based on this model we are able to effectively visualize 2D, 3D and even animated 3D data on variety of mobile devices, including cell phones.

III. SYSTEM ARCHITECTURE

A. Overview

Our system consists of a lightweight client program and a distributed server application. The server side application is divided into modules, each of them playing a different role in the system, as shown in the Fig. 1. Visualization session begins when the user launches the client's application on his device. It automatically connects at the startup to a session management module of a server. This module is responsible for handling all user activities during a session, including zooming, moving and rotating of the remote objects. Users' inputs intercepted on their devices are automatically sent to a session management module, where they are translated into adequate server directives and passed to a rendering module. Depends on the received data the rendering module generates successive frames of the visualized data and passes them to an encoding module. At this point every received frame is encoded in real time and is broadcasted to the client's device as a single video stream. The only job left for the client's application is to receive this video stream, decode it and display on the screen. Fig. 2 presents in details a complete workflow of our system.

B. Server modules

All server modules can be run on a single machine, although the system yields the best results when it is launched in a distributed environment, where all its main modules are deployed on the different computers.

Rendering module is completely transparent in the sense of source data, which means that they can be either read from disk, database or rendered in real time using a dedicated machine. Depends on the user's choice the rendering module generates successive frames of a visualized object and sends them using socket communication to the encoder, without any compression. We have developed our own robust protocol to stream raw byte arrays between these two servers, which minimizes the latency of network traffic. Because we are not using any compression at this point, it is recommended that both servers communicate using broadband connection.

Encoding module receives these data, compresses them using a video codec and forwards to the client using a dedicated streaming protocol. Every user receives their own video stream, individually customized to suit capabilities of the user's device and currently available network bandwidth. Based on this information, the encoding module can automatically adapt video broadcast using varying quality, frame rate and resolution of the output stream. Video bit rate is automatically decreased during user interaction with an object (zooming, spinning, rotating), as there is no need to overload network communication at this point. Only the last frame in each sequence is sent to client being compressed using a higher bit rate. This way user saves network bandwidth and device's battery lifetime.

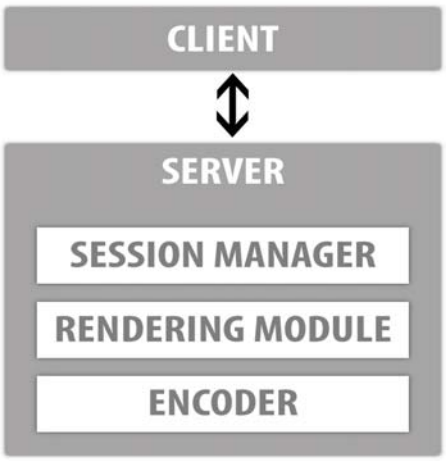

Figure 1. A general schema of the system architecture

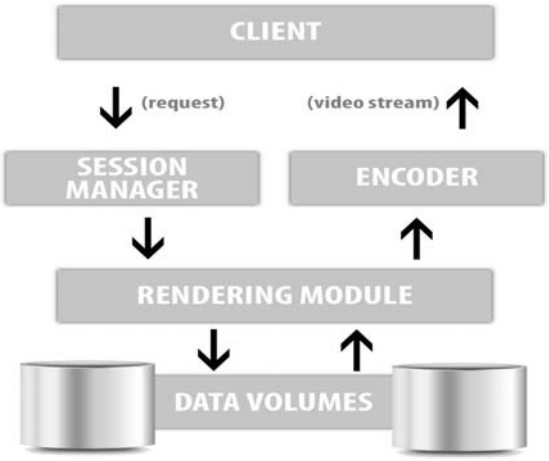

Figure 2. Communication workflow between the client's application and server modules

C. *Visualization interactivity*

In our system the term interactive visualization means that user can manipulate remote video streams in real time, using mouse or touch gestures, and receives almost immediate feedback from a server, automatically adapted to taken action. Our system can work this way with 2D, 3D and even animated 3D data, all of them being processed on a remote server in real time and broadcast to user as a single video stream. Depends on the source data user can take different actions, like zooming, moving, rotating and animating a remote object.

Two-dimensional images are usually generated only once in the rendering module, and are buffered in encoder for rest of the session. When user wants to zoom in / out or move to the different part of the image he selects an appropriate region on the screen and sends its coordinates to the server. Based on the user's choice an adequate portion of the source image is cropped and added to an output sequence, as shown in Fig. 3. This sequence is encoded as a video stream and broadcasted back to the user. This way a full image is never downloaded to user's device, and only its smaller regions are successively streamed from the server, giving the effect of zooming and movement. This technique gives great results, especially when high-resolution images must be visualized on the mobile devices, which are usually equipped with very small screens.

Remote visualization of 3D data works very similarly to 2D images, however besides zooming and moving, server objects can be additionally rotated along the X and Y axes. The main difference is that successive rotation frames are dynamically generated in the rendering module depends on user's choice, without the need of buffering them in encoding block. Generated frames are streamed in real time to encoder, where they are turned into video sequence and broadcast to client. Only the last frame of each rotation sequence is buffered in encoding module, so it could be later zoomed in / out and moved similarly to 2D images.

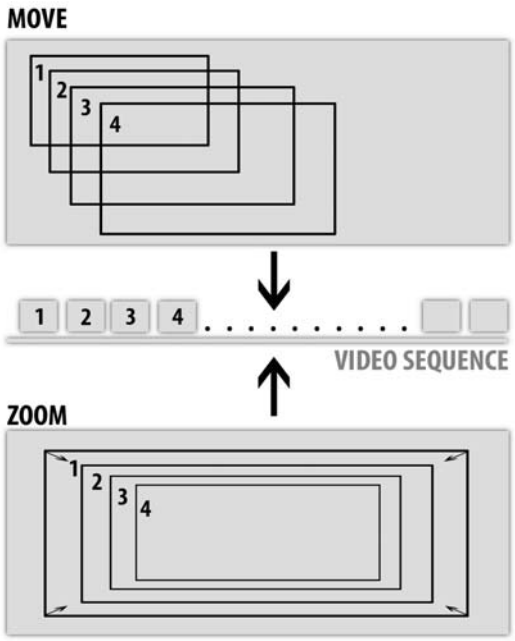

Figure 3. Outgoing video stream is generated from the successive sectors of an input frame, cropped based on user's selection

Visualization of animated 3D data derives directly from the above technique. Animated objects can also be zoomed, moved and rotated, however, every frame can additionally evolve in time. Depending on a current viewing angle, the rendering server passes adequate frame sequence to encoding block, which compresses it and broadcasts to user in a loop, giving the effect of an infinite animation. User can pause the animation at any moment and swap its frames one by one. Every frame explored this way can be additionally manipulated using previously mentioned techniques, including zooming, moving and rotating effects.

D. *Technologies*

We wanted to create a visualization system, which would be completely multiplatform and compatible with a variety of modern mobile devices. That is why we planned to develop client side application in either Java or Adobe Flash technology, which so far has been available on most popular desktop operating systems, including Windows, Linux and Mac OS X. We have tested both solutions extensively, checking their strengths and weaknesses in context of mobile multimedia applications. At last we have decided to use Adobe Flash technology.

For the last couple of years Adobe has been cooperating with top mobile manufacturers under the Open Screen Project [17]. The main goal of this project is to develop a unified runtime environment, which will allow launching the same Action Script 3.0 code on different mobile operating systems. As a result, full Flash Player 10.1 has already been ported to Android and Palm OS devices. According to the project's website, Flash Player should also be available on Blackberry and Windows Mobile soon. Additionally, with the use of Adobe's iPhone packager, Action Script 3.0 code can be very easily run on Apple's iOS devices.

On the other hand, number of mobile Java applications is developed using J2ME SDK, which is completely different from the desktop distribution. Additionally, J2ME virtual machines vary a lot among manufacturers, which accounts for the fact that at least some parts of the Java code needs to be rewritten before being used on different mobile devices.

The second reason why we have chosen Flash over Java was that presently it has a much better support for Internet video streaming. According to different sources over 75% of video clips available on the Internet is published using Adobe's technology [18]. Additionally, Adobe's Real Time Messaging Protocol seems to be much more functional than RTSP supported by J2ME. Java Media Framework, which used to be the best API for video processing in Java unfortunately stopped evolving long time ago, and currently does not support most of modern video formats.

On the other hand, our server side application has been written almost completely in Java. We have also used Wowza Media Server as a video streaming solution, mostly because its built-in support for RTMP, re-streaming functionality and available API, which allows to develop server-side applications using Java. Network communication between Flash clients and server application is realized using Adobe's Real Time Messaging Protocol.

Server's encoding module uses FFMPEG for video compression, which presently is one of the best open source solutions for manipulating video files. We have also used Xuggler framework to call FFMPEG libraries directly from our Java application. Adobe's Flash Player currently supports three video codecs: Sorenson Spark, VP6 and H.264. The current version of our system uses Sorenson Spark, which derives directly from an H.263 standard. We have decided to use Sorenson's solution mostly because of its low CPU requirements for both encoding and decoding, which plays a crucial role in performance of our system. We have run some preliminary tests using X.264, which is an open version of H.264 codec. So far we have left it under experiments because of its high requirements for CPU power. Unfortunately, we couldn't experiment with a VP6 codec because of its license limitations. In the future we are also planning to run tests on other popular video formats, including VP8, which should be supported by one of next releases of Flash Player.

IV. RESULTS

We have already run a series of preliminary performance tests of our system using sample multidimensional medical data. We have tested server's video encoding module efficiency, because this is the most computational power consuming part of our system.

Testing procedure was run in a distributed environment consisting of two different servers and variety of clients' devices. Server modules were launched on dual core Intel Xeon X5355 2.6 GHz, responsible for data rendering, and quad core Intel Xeon E5420 2.5 GHz, which encoded live video streams. On the client side we used desktop computers running Windows Vista, laptop with Mac OS X, as well as tablet and cell phone, both equipped with Android 2.2 system. The number of servers can be increased with the number of clients.

We have measured the CPU usage and encoding speed during object zooming, moving, rotating and animating. Outgoing video parameters were set at 25 fps, and bit rates of 2 mb/s and 4 mb/s respectively for user's interaction and last frame encodings. We have run simulation for three different screen resolutions (320x240, 800x480, and 1366x768) and different number of concurrent connections (5, 10 and 20). Tables 1, 2 and 3 present obtained results.

Our early experiment proved that even while having many simultaneous users connected to a single server,

Table 1. System performance for 5 simultaneous users

Screen size	Zoom fps	Move fps	Rotation fps	Animation fps	Top CPU usage
320 x 240	254.7	271.6	369.5	493.6	3%
800 x 480	70.2	70.9	99.2	186.5	10%
1366 x 768	29.3	21.6	36.0	82.8	23%

Table 2. System performance for 10 simultaneous users

Screen size	Zoom fps	Move fps	Rotation fps	Animation fps	Top CPU usage
320 x 240	252.5	278.9	348.4	491.1	7%
800 x 480	62.2	66.7	82.5	170.2	20%
1366 x 768	20.0	16.0	21.3	60.2	60%

Table 3. System performance for 20 simultaneous users

Screen size	Zoom fps	Move fps	Rotation fps	Animation fps	Top CPU usage
320 x 240	240.0	264.8	331.7	498.0	11%
800 x 480	55.6	53.3	64.9	138.8	42%
1366 x 768	5.8	5.5	17.6	15.1	80%

video compression speed was very fast. Screen resolutions of 800x480 and 320x240 pixels, which presently are the most typical sizes of modern cell phones, produced very promising results, never descending below established 25 fps level. The results could be even better if we set smaller bit rates of the outgoing videos, adequate to the 3G network's capabilities rather than WLAN.

The lowest acceptable encoding speed is 15 frames per second. Below that value video display looses its smooth and could have a very bad impact on the visualization interactivity. In our case the encoding server slowed down below that level only when we set a very high resolution of video stream, which is typical for desktop computers rather than mobile devices. However, even then, despite lower fps results, we were still able to successfully visualize all data, achieving gratifying reception of the whole session.

V. CONCLUSION AND FURTHER WORK

In this paper we have presented a distributed system for remote visualization of large datasets on mobile devices. In the proposed solution all the data are rendered on dedicated servers, compressed using video codec and broadcast to users as an interactive video streams. Users can view and manipulate remote objects using different types of mobile devices. Our system works with 2D, 3D and animated 3D data, which can be zoomed in / out, moved and rotated over X and Y axes in real time. Performance tests showed, that this system is able to effectively visualize remote data on mobile devices, even with many concurrent server sessions.

In the future we are planning to experiment more with different video formats, including X.264 and VP8 standards. We also want to extend functionality of our system with the collaboration module, which should let many concurrent users to cooperate over remote data in real time. All the results obtained from our early experiments prove that we should further split modules over different machines, which should further increase overall performance of our system.

REFERENCES

[1] N. T. Karonis, M. E. Papka, J. Binns, J. Bresnahan, J. A. Insley, D. Jones, and J. M. Link, "High-resolution remote rendering of large datasets in a collaborative environment", Future Generation Computer Systems 19, 2003, pp. 909-917.

[2] Z. Constantinescu, and M. Vladoiu, "Adaptive compression for remote visualization", Buletinul Universitatii Petrol – Gaze din Ploiesti, vol. LXI, No. 2/2009, pp. 49-58.

[3] K. Ma, and D. M. Camp, "High performance visualization of time-varying volume data over a wide-area network", IEEE, 2000.

[4] D. Dragan, and D. Ivetic, "Architectures of DICOM based PACS for JPEG2000 medical image streaming", ComSIS, vol. 6, no. 1, June 2009.

[5] N. Lin, T. Huang, and B. Chen, "3D model streaming based on JPEG2000", http://graphics.im.ntu.edu.tw/docs/tce07.pdf.

[6] S. Hu, "A case for 3D streaming on peer-to-peer networks", Web3D 2006, Columbia, Maryland, 18-21, April 2006.

[7] W. Sung, S. Hu, and J. Jiang, "Selection strategies for peer-to-peer 3D streaming", NOSSDAV, Braunschweig, Germany, 2008.

[8] M. Mosmondor, H. Komericki, and I. S. Pandzic, "3D visualization of data on mobile devices", IEEE MELECON 2004, Dubrovnik, Croatia, May 12-15, 2004.

[9] R. R. Lipman, "Mobile 3D visualization for steel structures", Automation in Construction 13, pp. 119-125, 2004.

[10] K. Engel, and T. Ertl, "Texture-based volume visualization for multiple users on the World Wide Web", http://wwwvis.informatik.uni-stuttgart.de/eng/research/pub/pub1999/EGVE99.pdf.

[11] K. Engel, O. Sommer, C. Ernst, and T. Ertl, "Remote 3D visualization using image-streaming techniques", http://www.vis.uni-stuttgart.de/ger/research/pub/pub1999/ISIMADE99.pdf.

[12] K. Engel, P. Hastreiter, B. Tomandl, K. Eberhardt, and T. Ertl, "Combining local and remote visualization techniques for interactive volume rendering in medical applications", http://www.vis.uni-stuttgart.de/ger/research/pub/pub2000/engel_vis00.pdf.

[13] S. Stegmaier, M. Magallon, and T. Ertl, "A generic solution for hardware-accelerated remote visualization", IEEE TCVG Symposium on Visualization, 2002.

[14] M. Hereld, E. Olson, M. E. Papka, and T. D. Uram, "Streaming visualization for collaborative environments", http://www.mcs.anl.gov/uploads/cels/papers/P1512.pdf.

[15] L. Cheng, A. Bhushan, R. Pajarola, and M. E. Zarki, "Real-time 3D graphics streaming using MPEG-4", July 18, 2004.

[16] Y. Noimark, and D. Cohen-Or, "Streaming scenes to MPEG-4 video enabled devices", IEEE Computer Graphics and Applications, January / February 2003.

[17] http://www.openscreenproject.org/

[18] Comscore – Video Metrix Report, August 2009

[19] M. Chlebiej, K. Benedyczak, and P. Bała, "Technologie strumieniowe", November 2009.

[20] M. Chlebiej, "Urządzenia mobilne w szpitalach – wizualny system multimedialny", December 2008.