An Open Source Cloud Gaming Testbed Using DirectShow

Hamed Ahmadi[1], Mahmoud Reza Hashemi[1], Shervin Shirmohammadi[1,2]

[1]*Multimedia Processing Laboratory (MPL), School of Electrical and Computer Engineering, College of Engineering, University of Tehran, Tehran, Iran*

{ha.ahmadi | rhashemi | sshirmohammadi}@ut.ac.ir

[2]*Distributed and Collaborative Virtual Environments Research Laboratory (DISCOVER Lab), School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, Canada*

Abstract— **Despite its challenges, cloud gaming is growing its share in the gaming market by attracting more players. This has led to an increasing number of researches trying to overcome cloud gaming's challenges, including the required high bandwidth and low latency, to make cloud gaming more practical and profitable. To perform this research, researchers need a testbed to evaluate their ideas and find the best solutions. Currently, GamingAnywhere is the only open source platform and testbed to serve this goal. However, it cannot be used to stream all video games, since it depends on hooking APIs which might be incompatible with some video games. In this paper, we introduce a new open source cloud gaming testbed. In this testbed, the screen capturing module is fundamentally a DirectShow filter and, hence, can be tuned for any DirectShow compatible video game. The testbed also facilitates the measurement of delay and quality as the video is processed through its modules.**

***Keywords*— Cloud Gaming, Evaluation Testbed**

I. Introduction

Cloud gaming is an alternative way to deliver high-quality gaming experience. The concept of cloud gaming is to render a video game in the cloud and stream the game scenes as a video to the players over a broadband network [1]. In a cloud gaming system, the gameplay control commands (mouse, keyboard, joystick, or other types of game controller events) are captured at the client and sent to the cloud to be processed by the game application. As the game engine receives these commands, it updates the game state and renders the next game frame. This game frame is then encoded by the video encoder and sent over the network to the client's device where the frame is decoded and shown on the display.

Cloud gaming has many advantages for users as well as game developers. On the one hand, users no longer need to purchase high end graphical hardware to run new games and can play on virtually any device that can run video. On the other hand, developers no longer have to fear software piracy, as the software never leaves the cloud. Furthermore, this approach can reduce development costs by focusing on one specific platform. These advantages, among others, have lighten up a promising future for cloud gaming as the market analysts predict a nine-time increase in this section of game industry by 2017 [2].

Cloud gaming, however, has its own challenges. First, it requires a high bandwidth network to simultaneously stream the game as a video sequence to multiple players. Second, it is sensitive to network latencies since a long latency seriously impairs the interactive experience of a video game. These restrictions make cloud gaming unavailable to users who have lower bandwidth connectivity, such as mobile users. Beside gamers' need, there are other elements such as efficiency, error resiliency, scalability, and resource allocation that must be taken into considerations while designing a cloud gaming system which exposes an enormous cost to service providers.

Because of its prospering potential, not only cloud gaming's challenges have not deterred its market growth, but also have motivated a great deal of competition among service providers to acquire the largest share in the market by being the first to conquer these challenges. It has also levered a lot of research on both software and hardware aspects of cloud gaming systems.

One of the key needs of any research pertaining to cloud gaming, is to have a testbed on which a researcher could run his experiments and assess the efficiency of his proposed solutions on practice. The testbed should be close enough to the real cloud gaming systems so that the results of the experiments could be applied to the real world with as little adjustment as possible. In order for researchers to conveniently utilize their ideas into the framework, the testbed's license should also allow free usage and modification, at least for academic purposes. Being extensible and configurable are among the other crucial features of such a testbed.

GamingAnywhere [3] is the first open source cloud gaming testbed on which researchers can implement and test their new ideas. It has been written in C/C++ and it leverages several popular open source libraries such as *ffmpeg* and *libSDL.* GamingAnywhere can be compiled from scratch. One exception is its dependence on Windows libraries. Since building them can be tricky, the pre-built Win32 libraries have been included in the software pack.

Despite its excellent usefulness, one of the drawbacks of GamingAnywhere is the mechanism it uses to capture game screens. It directly hooks into game executables. If successfully attached, the hook would highly outperform other alternative such as periodically capturing the desktop or a window via its handle. However, very frequently, this hooking mechanism fails. For example, GamingAnywhere is unable to successfully hook into the video games which acquire the Direct3D device for more than once during gameplay. Unfortunately, this is a common case, since most games usually start with showing some interesting graphic artworks while loading the required game data such as textures and meshes. Once loaded, such games close the current Direct3D device and acquire it again with different configurations which better suit the ongoing parts of the game. In fact, in practice, the hooking challenges differ from game to game, depending on the technologies used for developing each game.

In this paper, we introduce a new open source cloud gaming testbed which is based on DirectShow filter graphs [4]. Filter graphs allow incorporating various filters in an arbitrary order to complete a multimedia task. Hence, they offer a great deal of extensibility for the testbed. Moreover, each filter can be configured before and while the filter graph runs. Therefore, similar to GamingAnywhere, the introduced testbed is both extensible and configurable. The testbed is not, however, portable, since DirectShow filter graphs only execute on Windows operating systems. Albeit, it should be noted that multimedia filter graphs can also be built on Linux platforms using GStreamer [5].

The main advantage of the introduced testbed is the interdependence among its constructive modules (i.e. filters). More specifically, researchers can modify a filter's implementation to address their research requirements without impairing any other filter's functionality. For example, the hooking mechanism can be tuned for each game by only modifying the capturing filter which is a clearly distinct module in the proposed framework, compared to GamingAnywhere. Furthermore, the researcher can expand the testbed's functionality by adding and/or removing specific filters. As an instance, by adding a delay-measurement filter, it is possible to analyse and compare various methods to speed up video encoding and video streaming at the cloud side. As another example, employing a video quality measurement filter would be of great help to investigate the efficiency of perceptual video coding on decreasing the video bandwidth [6].

The remainder of this paper is organized as follows. The architecture of our cloud gaming testbed is explained in the next section, while its deployment is explained in section III. Section IV described the directory structure of our testbed, followed by discussions in section V. Finally the paper ends with concluding remarks.

II. Testbed Architecture

A cloud gaming testbed contains two key parts: server and client. Rendering, video encoding and streaming are performed on the server side. On the other side, the client decodes and displays the video stream on the player's screen while capturing and sending to the server the gameplay control commands.

A. Server side

Our testbed's server program has been written in C# .NET. It has two threads. One thread is in charge of receiving user inputs from the client side and delivering them to the game engine. The other thread builds and runs a DirectShow filter graph.

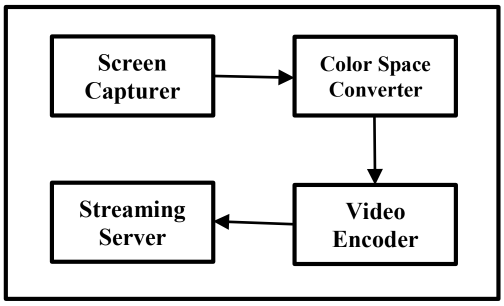

Fig. 1 The DirectShow filter graph at the server side including four filters

DirectShow divides a complex multimedia task into a sequence of fundamental processing steps known as filters. Each filter, which represents one stage in the processing of the data, has input and/or output pins that may be used to connect the filter to other filters. The generic nature of this connection mechanism enables filters to be connected in various ways so as to implement different complex functions. More specifically, to convey a media from a filter to its next filter in the graph, an unconnected output pin on the upstream filter is connected to an unconnected input pin on the downstream filter. Pins may support different media types, but upon connection they must agree on a common media type through a negotiation process. To implement a specific complex task, a developer must first build a filter graph by creating instances of the required filters, and then connecting the filters together.

Note that, as mentioned before, there is no restriction on the number of filters. Users can simply add or remove the filters and even modify them to adapt the testbed to their specific academic purposes. However, in our implementation, the DirectShow filter graph includes four filters to adequately act as a basic cloud gaming server. Fig. 1 shows the built-in graph in the testbed's server program. Each filter, in this figure, has its own responsibility, as described next.

1) *Screen Capturer*: Since accessing the source code of commercial video games is typically restricted to companies that license the game, researchers almost never have the opportunity to modify a video game such that it directly provides the game frames for the video encoder. Therefore, in these cases, there is a need to run the original video game on a machine and then capture game frames from the graphics buffer. If the game uses a graphics application programming interface (API), such as DirectX or OpenGL, a more efficient alternative would be to grab the game frames directly from the video memory. Fig. 2 shows the data flow between an application and a graphics card through different graphics APIs [7]. As can be seen, unlike GDI, DirectX and OpenGL

have direct access to the video memory through the Hardware Abstraction Layer (HAL) device driver. A HAL device provides hardware acceleration to graphics pipeline functions, based upon the feature set supported by the graphics card.

In our implementation, we incorporate a C# based virtual video capture source filter which is written by Maxim Kartavenkov [8] and licensed under the *Code Project Open License (CPOL)*. This filter has been rated five out of five and downloaded over 13500 times by Code Project users. It grabs the desktop via calling *CreateDC* API function nested in "gdi32.dll". The user can simply modify the filter to acquire the desired video game window handle by using the *FindWindow* API function included in "user32.dll" library.

In some situations, researchers have to design a game from scratch to investigate a specific behaviour among players. These kinds of games are not usually as complex as commercial games since they are designed to serve a pre-defined academic goal rather than entertaining the players. Thus, the whole game, in these situations, can be designed as a DirectShow source filter which reduces the burden of capturing the screen.

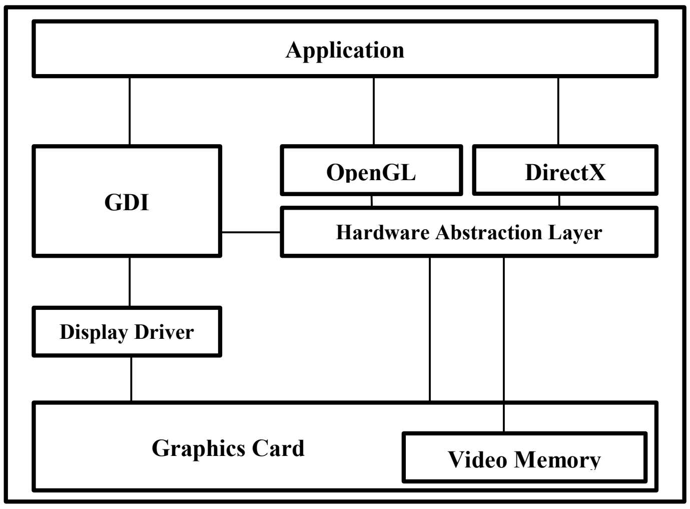

Fig. 2 The relationship among graphics APIs, applications and graphics hardware

2) Color Space Converter: Most video encoders require their uncompressed input stream to be in YUV color space. Therefore, based on the Screen Capturer filter's output color space, there might be a need for a color space conversion. In our implementation, since the incorporated Screen Capturer filter streams video in RGB format, we uses an RGB to YUV filter before sending the stream to the video encoder.

3) Video Encoder: Currently, most cloud gaming companies use H.264/AVC as their basic video encoding standard. Thus, we utilize an H.264/AVC DirectShow filter in our implementation. However, it is possible to use any other video encoding standards, such as High Efficiency Video Coding (HEVC). Moreover, by using modified filters, video coding researchers have the opportunity to propose potential enhancements to the current standards and evaluate new ideas.

4) *Streaming Server:* Once the video has been encoded, it should be delivered to a streamer server so that the clients can connect and acquire the stream. In our implementation, we use an Real-Time Streaming Protocol (RTSP) server DirectShow filter written by Geraint Davies [9], which supports H.264/AVC and AAC media types. When the filter graph is running, the input is fed to an RTSP server, which will transmit as a live stream. The filter decouples the sequence and picture parameter-set (i.e. SPS and PPS) Network Access Layer (NAL) units from the encoded video sequence to grasp the media type, format, and all associated persistent properties of the sequence to prepare a session profile. Upon playing the stream, the filter encapsulates the media packets into Real-Time Protocol (RTP) transport packets and sends them to the client.

B. Client side

The ultimate goal of cloud gaming is to help players play their favorite video games wherever they are, whenever they want, and on any device they wish. Unquestionably, if the client side could run in a web browser, cloud gaming would rapidly approach what it is meant to be. More specifically, wireless handheld devices are increasingly becoming widespread among people. Nearly all these devices enjoy pre-installed and powerful web browsers which support the HTML5 video tag. Supporting this tag allows playing videos in webpages without requiring a plug-in. Although, different browsers have support for different video formats, the latest versions of all popular web browsers, be it a desktop or a mobile browser, support H.264/AVC.

In addition to playing video, the client side should be capable of gathering the game controls from the web browser and sending them to the server. In our implementation, we use AngularJS [10], an open source JavaScript web application framework, to capture mouse and keyboard events while the player interacts with the game's video through the web browser. Touch events can also be captured using its ngTouch module [11]. Furthermore, in order to send the captured game controls to the server, we use Web Sockets technology which operates over a single socket and is exposed via a JavaScript interface in HTML5 compliant browsers. Another point to mention is that web applications which are developed in HTML5, JavaScript and CSS can be easily converted into mobile apps [12]. These apps are run through an invisible browser that is packaged into a native application.

III. DEPLOYMENT

There are two key considerations to run the server on a machine. First, .Net Framework 3.5 or later must be installed on the machine. Second, if there exists a pre-compiled or third-party DirectShow filter that has been called inside the server's source code, it must be registered on the machine. Afterwards, the server program can be run easily. Fig. 3 shows the main form of the server program.

As can be seen, it comprises two parts: DirectShow Filter Graph Manager and Gameplay Manager. The former provides an interface to initialize, play, pause and dispose the filter graph. The latter is in charge of receiving the gameplay commands and simulates them on the machine via calling the *SendInput* API function packed in "user32.dll" library. In this manner, the game engine accesses the synthesized mouse and keyboard events while the player generates them on the client device.

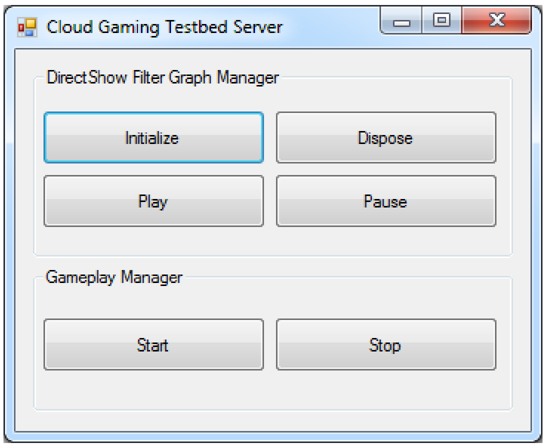

Fig. 3 The main form of the server program

The filter graph is responsible to capture, encode and stream the game frames. Once initialized, the user can play or pause the filter graph. If the video game itself is not implemented as a DirectShow filter and runs as a separate application, it should be executed before initializing the filter graph. After finishing the experiment, the user is required to dispose the filter graph to release the allocated resources.

Fig. 4 shows the testbed in our lab. In this figure, the PC on the right-hand side runs the "My Beautiful Doll, Somi" video game and the developed server application. The PC on the left-hand side receives the stream and displays it.

Fig. 4 A demonstration of the testbed. The server (on the right) is streaming "My Beautiful Doll, Somi" video game and the client (on the left) is decoding and displaying the stream.

IV. Directory Structure

The testbed is released with an all-in-one software package which includes the testbed source code, third-party library source code, and pre-compiled binaries and libraries. The package is available on the Multimedia Processing Laboratory (MPL) website here[1] . There are three subdirectories in the package, which are:

- **Executables**: contains all the required run-time files and their dependent shared libraries.
- **Libraries**: contains the pre-compiled versions of all the required libraries.
- **Source code**: the source code of the testbed along with its dependencies have been placed here.

V. Discussion

In cloud gaming, the cloud not only processes the game logic, but also the video rendering. Nonetheless, in order for this promising paradigm to become fully practical, researches are required to overcome its shortcomings. The potential solutions would cover a broad spectrum including methods to speed up video encoding and video streaming at the cloud side, live and real-time parallel video encoding in the cloud, methods to decrease video bandwidth while maintaining visual quality, and energy-efficient cloud computing for video rendering at the server side, among many others. However, no possible solution is incorporated into real practical systems unless it has been thoroughly evaluated and has shown significant enhancement over the existing utilized solutions. Therefore, researches must be provided with evaluating tools to conduct exhaustive experiments and find efficient solutions. The more diverse the evaluating tools are, the more situations can be simulated and the more efficient solutions can be found. Currently, GamingAnywhere is the only cloud gaming testbed that is publicly available for researchers. But it is not enough, since not all games can be streamed on it. So, in this paper, we introduce a new cloud gaming testbed to cover a wider range of games. We hope it will encourage the community to continue developing more testbeds with different architectures to cover even more games and situations.

The introduced testbed is in its infancy and should continuously evolve to build trust among researchers. Therefore, as our future work, we plan to develop more DirectShow filters to add more functionality to the testbed and also provide alternative to the current modules. This allows researchers to compare the performance of different solutions and find the optimum configurations. It should be noted that in addition to open source filters, third-party filters can also be incorporated into the testbed. To do so, the filter must be first registered on the machine and then be loaded through its GUID.

VI. Conclusions

In this paper, we presented an open source cloud gaming testbed based on DirectShow. This open source testbed allows researchers to implement their own ideas, on how to overcome cloud gaming's challenges and increase gamers' experience, using DirectShow filters. They can simply modify or replace the built-in filter graph's nodes to adjust the testbed based on their own research requirements. Using DirectShow filters

[1] http://www.site.uottawa.ca/~shervin/CGTestbed

also makes the testbed extensible and configurable. Thus, it can be adopted in various usage scenarios.

REFERENCES

[1] H. Ahmadi, S. Z. Tootaghaj, M. R. Hashemi, and S. Shirmohammadi, "A game attention model for efficient bit rate allocation in cloud gaming," *Multimedia Systems,* vol. 20, pp. 485–501, 2014.

[2] (2015/08/08). *Distribution and monetization strategies to increase revenues from cloud gaming* [Online]. Available: http://www.cgconfusa.com/report/documents/Content-5minCloudGamingReportHighlights.pdf

[3] C.-Y. Huang, C.-H. Hsu, Y.-C. Chang, and K.-T. Chen, "GamingAnywhere: an open cloud gaming system," in *Proceedings of the 4th ACM Multimedia Systems Conference*, 2013, pp. 36-47.

[4] (2015/08/08). *The Filter Graph and Its Components* [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/dd407188(v=vs.85).aspx

[5] (2015/08/08). *GStreamer: open source multimedia framework* [Online]. Available: http://gstreamer.freedesktop.org/

[6] J.-S. Lee and T. Ebrahimi, "Perceptual video compression: A survey," *Selected Topics in Signal Processing, IEEE Journal of,* vol. 6, pp. 684-697, 2012.

[7] (2015/08/08). *DirectDraw Architecture* [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/hardware/ff553820(v=vs.85).aspx

[8] M. Kartavenkov. (2015/08/08). *DirectShow Virtual Video Capture Source Filter in C#* [Online]. Available: http://www.codeproject.com/Articles/437617/DirectShow-Virtual-Video-Capture-Source-Filter-in

[9] G. Davies. (2015/08/08). *DirectShow RTSP Server filter and RTSP Jukebox* [Online]. Available: http://www.gdcl.co.uk/2013/05/16/RTSP-Jukebox.html

[10] (2015/08/08). *AngularJS — Superheroic JavaScript MVW Framework* [Online]. Available: https://angularjs.org/

[11] (2015/08/08). *ngTouch* [Online]. Available: https://docs.angularjs.org/api/ngTouch

[12] J. MORONY. (2015/08/08). *The Step-by-Step Guide to Publishing a HTML5 Mobile Application on App Stores* [Online]. Available: http://www.joshmorony.com/the-step-by-step-guide-to-publishing-a-html5-mobile-application-on-app-stores/