HTML5 Based Media Player for Real-Time Video Surveillance

Guolei Zhu, Fang Zhang, Wei Zhu, Yayu Zheng
College of Information and Engineering
Zhejiang University of Technology
Hangzhou, China

Abstract—**Media player plays a very important role in real-time video surveillance. But currently many media players, which are based on DirectX, Flash and ActiveX, have many problems such as installation failure, maintenance cumbrance, security defects and so on. With the rapid development of HTML5 technology, the HTML5 video tag is very excellent for static video files, but currently it is not very suitable for streaming live video. To overcome these problems mentioned above, a new media player based on the HTML5 features for real-time video surveillance is designed and implemented in this paper. In the presented method, the HTML5 WebSocket is used to stream real-time H.264 data to the browser and then the HTML5 Canvas is adopted to make the video data render in the HTML5 compatible web browser after the H.264 data is decompressed natively by the JavaScript H.264 decoder. And the usage of HTML5 Web Workers can ensure that the video data is received, decoded and rendered at the same time. At last, the experimental results demonstrate that the new media player in this paper can play the video whose resolution is 640×360 at the rate of 25 frames per second and that it can be applied to real-time video surveillance without browser plug-ins.**

Keywords-media player; surveillance; HTML5; plug-ins

I. INTRODUCTION

With the rapid application and development of network technology and multimedia stream technology, real-time video surveillance system is widely used in many fields. Media player which is a very important part in the video surveillance system is responsible for receiving, decoding and displaying the video data.

Nowadays, media player based on DirectX techniques is widely adopted in real-time video surveillance system. The DirectX technique is developed by Microsoft Corporation and includes security and performance updates, along with many new features across all technologies, which can be accessed by applications using the DirectX APIs. And it can support the decoding and playback of media files with different video and audio compression format and greatly promote the usage of media player. But the media player which is developed by DirectX SDK only can run on Windows OS, not on any other platform such as Linux and Mac OS [1]. Moreover, this type of media player is based on Client/Server(C/S) architecture and increases the cost of software maintenance and management.

The most widely adopted technologies to stream H.264 live video to a web browser remains browser plug-ins such as Adobe Flash plug-in and ActiveX plug-in. Flash player and ActiveX control work well in web browser as an embedded, plug-in runtime environment. And they are very convenient for end users to download and install and actually support high quality digital video effect [2]. But Flash and ActiveX both have a number of security defects and other problems. For example, end users who download and install a vulnerable Flash or ActiveX application can be attacked by an attacker sometimes [3]. Moreover, these technologies for streaming real-time video to a web browser allow the greatest variety of possibilities of configuration on the client side at the expense of having more requirements.

With the rapid development of HTML5 [4] technology, HTML5 video tag can be used to play video in the web browser without plug-ins. The video tag can process the static video files very well in the HTML5 compatible web browser. And it can give a good performance and a high quality video effect. [5] tested streaming a real-time video from a local webcam to an HTML5 video tag within Firefox browser. The browser frequently buffered a long period of time before and during the playback. So the HTML5 video tag is not suitable for streaming real-time video because of the huge lag between the recording and playback currently. In addition, the current HTML5 draft specification does not specify a video codec as standard. For example, H.264 is supported by Chrome and Safari browsers, while Theora is supported by Firefox, Opera and Chrome browsers. It is still not clear whether there will be a video codec supported in all browsers.

HTTP live streaming is great for delivering streaming media to the iOS-based application. HTTP live streaming can be used to send live or pre-recorded audio and video to the Apple products like iPad and iPhone, using an ordinary web server. The video is divided into segments with a length of a specified number of seconds. Then these segments are stored on an ordinary web server. The recommended segment size is 10 seconds and the minimal segment size is 1 second. However, this method is currently only supported by Apple products, not by any other products.

Therefore, to play real-time video across different operation system platforms and across different web browsers without plug-ins, this paper designs and implements a new media player for real-time video surveillance by making use of the HTML5 features such as WebSocket [6], Web Workers [7] and Canvas [8].

This work was supported in part by the Natural Science Foundation of Zhejiang Province under Grants No. Y1110532, Y1110175, Y1100632, LQ12F01008, and was supported by the Research Foundation of Education Bureau of Zhejiang Province under Grant No. Y201018355. The corresponding author is yayu zheng(email: yayuzheng@zjut.edu.cn).

978-1-4673-0964-6/12/$31.00 ©2012 IEEE

II. ARCHITECTURE OF MEDIA PLAYER

The video surveillance system is mainly composed of browser side and server side and is based on Browser/Server (B/S) architecture in this paper. And the media player that plays a very important role in the video surveillance system is mainly composed of four functional modules, including receiving, decoding, rendering and controlling. The basic functional requirements of the media player are shown in figure 1.

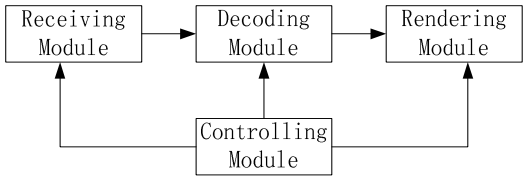

Figure 1. Media player functional modules

A. Receiving Module

HTML5 WebSocket is one of the most important functions in the upcoming HTML5 standard and is a very useful communication tool in Web applications. By making use of the HTML5 WebSocket, client side can maintain bidirectional communication with the server side. So the server can send the video data directly to the web browser without needing to re-establish a connection while the connection remains available.

B. Decoding Module

Because the video data from the server is compressed by the H.264 video coding standard, it can't be rendered directly in the web browser. So the H.264 video should be decoded before rendered in web browser. This paper uses a H.264 video decoder which is written in JavaScript to decompress the video data.

C. Rendering Module

The module is responsible for making the video data render in the browser. In the upcoming HTML5 standard, Canvas is a very useful painting tool. Compared to the Flash and Silverlight, HTM5 Canvas provides an approach to paint pictures with JavaScript in the web browser without plug-ins. After receiving and decoding the video data, the HTML5 compatible web browser can make the video data render in the canvas tag.

D. Controlling Module

The module is responsible for making the three modules mentioned above work well at the same time and responding to end user's requests. To make the media player work well, HTML5 Web Workers is adopted in the media player. Web Workers allows the task of receiving video data to be run in the background without being interrupted.

III. DESIGN AND IMPLEMENTATION

By making use of the HTML5 features including WebSocket, Web Workers and Canvas, this paper designs and implements a new media player which can play real-time video in the HTML5 compatible browser without plug-ins. The architecture of the media player is shown in figure 2.

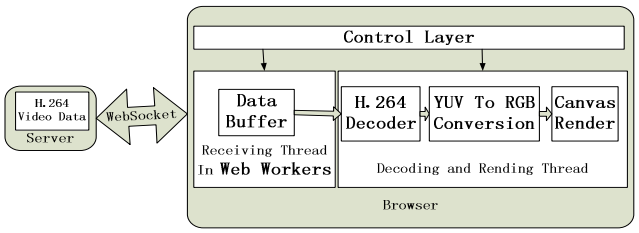

Figure 2. Media player architecture

To send data from the server to the client side, a traditional technique called "polling" is used. Every time the browser asks the web server for the video data, the browser must establish a connection with the server. And then the server responds by sending the data to the browser and closing the connection. So the conventional method for transporting video data not only increases the burden of traffic on the network, but also increases the delay time greatly.

To solve the problem, the media player uses the WebSocket protocol for bidirectional communication with a server. After the connection is established, the HTTP protocol is upgraded to the WebSocket protocol. And then the server can send data directly to the browser without needing to re-establish the connection every time. Moreover, the ports adopted in the WebSocket are 80 and 443. So the video data can pass through the firewall well. Figure 3 shows the transmission of the video data from the server to the browser based on WebSocket protocol.

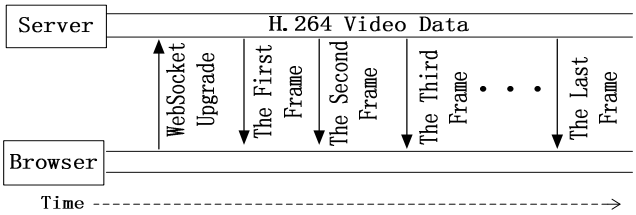

Figure 3. Video data transmission

HTML5 applications are obviously developed by using JavaScript language. But compared to many other development environments, JavaScript historically suffers from an important limitation that all its execution process remains inside a single thread. This JavaScript limitation implies that the main window will be frozen during a long-running process. And the user can't interact with the main application any more.

The HTML5 Web Workers APIs define a way to run script in the background. Some tasks can be executed in threads living outside the main page and thus non-impacting the drawing performance. By making use of postMesseage() API, data stored in the Typed Array like ArrayBuffer can be send to a Web Worker effectively. The method to pass data is called the structured clone. Though the structured clone is absolutely fine for data passing, it is still a copy operation. The time of passing a 32MB message to a Worker can be hundreds of milliseconds. Another approach for message passing between two contexts is webkitPostMessage(). It is a transferable operation. Because of zero-copy, ArrayBuffer can be transferred from one context to another context. When ArrayBuffer is transferred to a Worker, the original data is

cleared and becomes unavailable. And the time of transferring a 32MB message to a Worker can be less than 10 milliseconds. So the webkitPostMessage() API can greatly improve the performance of passing message to a Worker. However, only the Google Chrome 17+ browsers can support the webkitPostMessage() currently.

With the development of H.264 developed by ISO/IEC and ITU-T [9], the H.264 video coding standard is widely applied in the real-time video surveillance system. Because the video data from the server is compressed by H.264, it can't be rendered in the browser directly. So a H.264 video decoder which is written in C is used. But the decoder written in C can't be run on the web directly. To run C on the web, Emscripten is used in this paper. As a compiler from LLVM bitcode to JavaScript, Emscripten is written in JavaScript. C/C++ can be compiled into the LLVM bitcode through a frontend called Clang and then the LLVM bitcode can be compiled into JavaScript through a backend called Emscripten.

A following simple example is given to show how a C program can be compiled into JavaScript using Emscripten. The simple C program includes the follows:

```
#include <stdio.h>
int main()
{
   int total = 0;
   for (int i = 1; i < 50; i++)
   total += i;
   printf("1+...+50=%d\n", total);
   return 0;
}
```

This program calculates the sum of the integers from 1 to 50. After compiled by Clang and Emscripten, the above C program can be compiled into the following JavaScript code:

```
function _main() {
   var __stackBase__ = STACKTOP;
   STACKTOP += 12;
   var __label__ = -1;
   while(1) switch(__label__) {
   case -1:
   var $1 = __stackBase__;
   var $total = __stackBase__+4;
   var $i = __stackBase__+8;
   HEAP[$1] = 0;
   HEAP[$total] = 0;
   HEAP[$i] = 0;
   __label__ = 1;
   break;
   case 1:
   var $3 = HEAP[$i];
   var $4 = $3 < 50;
   if ($4) { __label__ = 2; break; }
   else { __label__ = 4; break; }
   case 2:
   var $6 = HEAP[$i];
   var $7 = HEAP[$total];
   var $8 = $7 + $6;
   HEAP[$total] = $8;
   __label__ = 3; break;
   case 3:
   var $10 = HEAP[$i];
   var $11 = $10 + 1;
   HEAP[$i] = $11;
   __label__ = 1; break;
   case 4:
   var $13 = HEAP[$total];
   var $14 = _printf(__str, $13);
   STACKTOP = __stackBase__;
   return 0;
   }
}
```

Therefore, many other languages like C and C++ can be run in the web browser without any plug-ins potentially by using Emscripten [10].

As an important part of the media player, HTML5 Canvas is responsible for displaying the video data in the web browser. Allowing developers to access the pixels is one of the most useful features in the Canvas API. When the getImageData method is called, a 2D rendering context ImageData object will be gotten. This ImageData object has an important content called CanvasPixelArray which contains information about all the pixels in the areas which is accessed. Each pixel can be presented by four integer values that range from 0 to 255 that refer to the Red(R), Green(G), Blue(B) and Alpha(A) values. Through rewriting each pixel with different color and alpha values, a different picture can be shown in the browser. Figure 4 shows each pixel accessed in the Canvas.

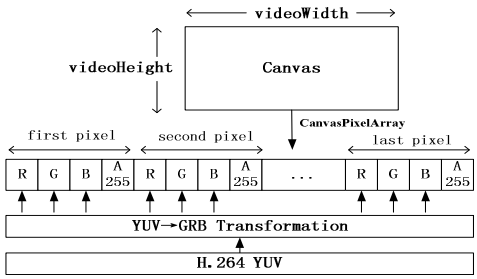

Figure 4. Pixel accessed in the Canvas

The pseudo code about accessing pixels in the Canvas as follows:

```
imageData = context.getImageData(0, 0, videoWidth, videoHeight);
canvasPixelArray = imageData.data;
for ( i = 0; i < canvasPixelArray.length; i + = 4 )
{
canvasPixelArray[i] = Red value from H.264 video data;
canvasPixelArray[i+1] = Green value from H.264 video data;
canvasPixelArray[i+2] = Blue value from H.264 video data;
canvasPixelArray[i+3] = 255;
}
context.putImageData(imageData,0,0);
```

If all the pixels in the 2D rendering context are updated rapidly at a specific rate, the result offers a stream-like video experience.

IV. EXPERIMENTAL RESULTS

To verify the feasibility of the media player, a web server based on Node.js is used to send the H.264 video whose resolution is 640×360 and the Internet Explorer, Google Chrome and Mozilla Firefox browsers are used to receive and play the video in the computer in which the CPU is Intel(R) Core(TM) i5 M450 @ 2.40GHz and the RAM is 2.0 GB. Table 1 shows the experimental results.

TABLE 1. EXPERIMENTAL RESULTS

Browser	Version	Support HTML5	Maximum Frame Rate (FPS)	Maximum Bit Rate (KbPS)
Internet Explorer	8	×	N/A	N/A
	9	√	N/A	N/A
	10	√	32	546
Google Chrome	15	√	49	836
	16	√	48	819
	17	√	50	853
Mozilla Firefox	9	√	36	614
	10	√	35	597
	11	√	37	631

Figure 5 shows three screenshots in the Internet Explorer 10, Google Chrome 15 and Mozilla Firefox11.

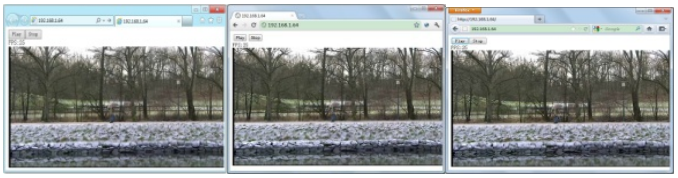

Figure 5. Three screenshots in three different browsers

The media player uses Typed Array to deal with the video data. But Typed Array isn't supported by Internet Explorer 9. So the media player can't be run in Internet Explorer 9 even though it supports the HTML5. In addition, because Internet Explorer 8 doesn't support the HTML5, it can't run the media player either.

According to the experimental results, the media player can be run in these browsers that support HTML5 except for IE8 and IE9. Moreover, the video which is played in the media player is clear, stable and continuous. And the maximum frame rate can reach more than 30 frames per second. So the media player can play the video at the rate of 25 frames per second if necessary. In addition, due to the difference of JavaScript engine performance in different browsers, the maximum frame rate reached in Google Chrome browser is much higher than the maximum frame rate reached in Internet Explorer and Mozilla Firefox browsers.

When the media player plays video, it takes up a lot of system resources in the end user's computer. Table 2 shows the resources occupancy information when playing the video whose resolution is 640×360 at the rate of 25 frames per second.

TABLE 2. RESOURCES OCCUPANCY WHEN PLAYING VIDEO

CPU Type	CPU Occupancy	RAM Occupancy
Intel(R) Core(TM)2 Duo CPU T5800 2.00 GHz	54% ~ 65%	178 MB
Intel(R) Core(TM)2 Duo CPU T9400 2.53 GHz	46% ~ 58%	176 MB
Intel(R) Core(TM) i5 CPU M450 2.40 GHz	25% ~ 34%	180 MB

The results in the table 2 were gotten when the video played in the three computers in which the CPU is Intel(R) Core(TM)2 Duo T5800 @ 2.00 GHz and the RAM is 1.0 GB, the CPU is Intel(R) Core(TM)2 Duo T9400 @ 2.53 GHz and the RAM is 3.0 GB, the CPU is Intel(R) Core(TM) i5 M450 @ 2.40GHz and the RAM is 2.0 GB. According to the results in the table 2, the CPU occupancy rate is much high and has a very big fluctuation while the video is playing. But with the enhancement of computer performance, the CPU occupancy becomes lower. The RAM occupancy is approximately 180MB and it doesn't change obviously with the enhancement of computer performance.

V. CONCLUSION

This paper designs and implements a new media player based on the HTML5 features for real-time video surveillance. Compared to those media players developed by DirectX, Flash and ActiveX, the media player in this paper is much easier to install and manipulate and can be run directly in the HTML5 compatible browser without plug-ins. But the media player is over-reliance on the CPU. So how to use WebGL and GPU to reduce the burden on the CPU is the future research.

REFERENCES

[1] Faxin Fu, Yuan shi, Xiaoli Lang. Design of General Media Player. Wireless, Mobile and Multimedia Networks, 2006 IET International Conference on. 6-9 Nov. 2006: 1-4.

[2] Jacqueline Emigh. New Flash player rises in the Web-video market. Computer. Feb. 2006: 14-16.

[3] Takanobu Watanabe, Zixue Cheng, Mizuo Kansen, et al. A New Security Testing Method for Detecting Flash Vulnerabilities by Generating Test Patterns. Network-Based Information Systems (NBiS), 2010 13th International Conference on. 14-16 Sept. 2010: 469-474.

[4] Ian Hickson. HTML5. http://dev.w3.org/html5/spec/Overview.html

[5] Mathias Kaspar, Nigel M. Parsad, Jonathan C. Silverstein. CoWebViz: interactive collaborative sharing of 3D stereoscopic visualization among browsers with no added software. Proceedings of the 1st ACM International Health Informatics Symposium. 2010: 809-816.

[6] Ian Hickson. The WebSocket API. http://dev.w3.org/html5/websockets/

[7] Ian Hickson. Web Workers. http://dev.w3.org/html5/workers/

[8] Andrew Wessels, Mike Purvis, Jahrain Jackson, et al. Remote Data Visualization through WebSockets. Information Technology: New Generations (ITNG), 2011 Eighth International Conference on, 2011: 1050-1051.

[9] Hari Kalva. The H.264 Video Coding Standard. Multimedia, IEEE. Oct.-Dec. 2006: 86-90.

[10] Alon Zakai. Emscripten: An LLVM-to-JavaScript Compiler. Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion. Oct. 2011: 301-312.