A NOVEL ALGORITHM FOR REAL-TIME FULL SCREEN CAPTURE SYSTEM

Te-Yi Liu, Yi-Chin Huang and Wen-Chin Chen
Communication and Multimedia Laboratory
Dept. of Computer Science and Information Engineering
National Taiwan University, Taipei, Taiwan.
{danny, yichin, wcchen}@cmlab.csie.ntu.edu.tw

Abstract – In this paper, we propose a novel system that can real-time capture and compress the full screen of PC into a video clip. It is real-time in that it can capture to 30 frames per second under the resolution of 1600×1200 with *True Color*. One application of this system is to produce a digital presentation clip for instruction or tutorial. Moreover, as the video clips can be streamed over Internet or Intranet, they can be used for remote education or training. We believe this approach is clearer and more efficient than conventional text manual or handbook. As our system only captures the differences of successive snapshots instead of every single screen, it is more efficient and produced more compact clips than other existing systems. In addition, the compression algorithm adopted in our system is also described in this paper.

INTRODUCTION

There has been an explosive growth in software industries in the past several years. More and more corporations have developed their own software packages and introduced them to people around the world. The proliferation of software packages is creating a pressing need for tutorial manuals and handbooks which not only introduce softwares' functionalities as well as methods of operation to users but also play a significant role in training course of company for newcomers; however, these materials are still paper-made now.

Nowadays, the use of visuals in user manuals for the computer industry seems to be a must. Often this is done by including various screen capture images throughout the manual. They can show, for example, a required start-screen or the correct result of an action [1]. However, although manuals with still screen capture images are widely used, users still cannot learn software efficiently due to the station of screen capture images. It is significantly advantageous to develop a screen capture system that can record every animated action on screen and generate a digital video clip [2]. In this paper we present a novel algorithm for real-time capturing and compressing the full screen into a video clip that can be stored in disk or streamed over Internet under the resolution of *1600×1200 True Color*.

PC Display Architecture

Typical display architecture is described as follows, the main CPU and its large memory space are on one side of the system expansion bus, and on the other side are graphics coprocessor and its somewhat smaller, but still significant, memory space.

0-7803-7025-2/01/$10.00 ©2001 IEEE.

In regular situations, operating system provides a block of display memory with specific size according to the current resolution and color depth of screen; in addition, RGB data that represent the current whole screen exactly lies in the display memory. When operating system repaints the screen, the new RGB data will be transferred from system memory to display memory through a performance-limiting expansion bus, which is called *Direction A*. However, the screen capture system performs a reverse directional action in contrast with normal behavior. When screen capture system wishes to capture current full screen, it will copy the RGB data from display memory to system memory through that expansion bus, which is called *Direction B*. Unfortunately, speed of *Direction B* is much slower than speed of *Direction A* since there are some hardware accelerations only for *Direction A*.

REAL-TIME CAPTURE MECHANISM

The screen capture system consisted of a real-time capture mechanism which was capable of real-time capturing information from screen efficiently and a well-designed compression algorithm which was able to generate a compact file for storing or streaming over Internet. In this chapter we would introduce two strategies of real-time capture mechanisms; one was conventional *Full Screen Capture* and the other was *Message Driven Polling* that we have designed.

Full Screen Capture

Full Screen Capture, which is the most straightforward real-time capture mechanism, was to capture full screen whenever needed. In such a case, the screen capture system would periodically retrieve full screen RGB data from display memory to system memory and formed a video clip with a sequence of RGB data.

However, this mechanism was extremely inefficient due to its heavy costs in spaces and time. For example, assume that current resolution of screen is 1024×768 *True Color* (32 bits), hence the size of a screen snapshot is $1,024 \times 768 \times 4 = 3,145,728$ Bytes, that is 3 MB. Unfortunately, we have experimented that transferring such 3 MB data from display memory to system memory would cost around 300ms*. Under the real-time playback constraint, 30 frames per second capture ability should be satisfied; however, according to our experimental statistics, transferring these data would cost 9,000 ms, that is 9 second, *which contradicted with the real-time assumption.*

According to our experiments, when adopting this approach, the application definitely cannot perform real-time due to the performance-limitation of expansion bus between display memory and system memory. However, the existing system almost adopted Full Screen Capture, and we have proved that it cannot satisfy real-time performance. In next section, we will propose a *Message Driven Polling* algorithm that can improve these shortcomings.

Message Driven Polling

In this section we proposed a novel approach in order to improve the perform-

ance of conventional Full Screen Capture. Since the vital drawback of Full Screen Capture was the overhead of excess data form display memory to system memory, our proposed approach would make more efforts to reduce the data that should be transferred from display memory.

Minimum Data Retrieval

Accordingly, although the *Graphics Device Manager* of operating system should be prepared to update the entire client area whenever it receives a *Repaint* message, it often needs to update only a smaller area, most often a rectangular area within the client area. This is most obvious when a dialog box overlies part of the client area. Repainting is required only for the rectangular area uncovered when the dialog box is removed. That area is known as an *"invalid region."* The presence of an invalid region in a client area is what prompts operating system to place a *Repaint* message in the application's message queue. An application receives this message only if part of its client area is invalid.

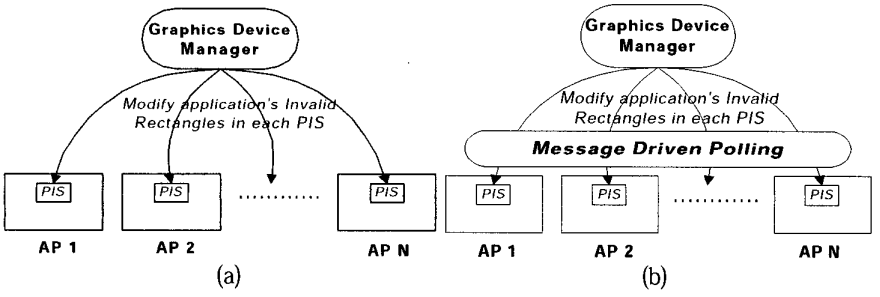

Figure 1: (a) Relationship between Graphics Device Manager and applications.
(b) Add Message Driven Polling component into original architecture.

Figure 1(a) shows that the operating system internally maintains a *"Paint Information Structure (PIS)"* for each application. This structure contains, among other information, the coordinates of the smallest rectangle that encompasses the invalid region. This is known as the *"invalid rectangle."* If another region of the client area becomes invalid before the application processes a pending *Repaint* message, the operating system calculates a new invalid region (and a new invalid rectangle) that encompasses both areas and stores this updated information in the *PIS*. Once an application would like to repaint its own client area, the *Graphics Device Manager* would repaint the current invalid rectangle of the application's client area by examining its *PIS*.

Implementation

Since *Graphics Device Manager* only repaints the invalid rectangles of applications, invalid rectangles must be not only the differences between successive screen snapshots but also the minimum data that should be transferred from display memory to system memory. Moreover, we could be aware of each occurred invalid rectangle by means of keeping track of every modification of running applications' *PIS*.

Thus, in Figure 1(b) the add-in *Message Driven Polling* component, which was to intercept all messages sent to each application and find out information about invalid rectangles, was the most crucial to our system. After catching the information about invalid rectangles, this mechanism could know the exact position of rectangle that is being repainted immediately and then copy the RGB data of *only* this rectangle from display memory to system memory. The flowchart of *Message Driven Polling* is shown in Figure 2.

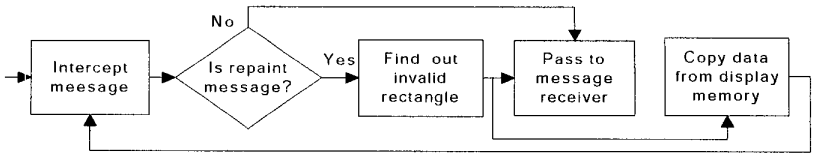

Figure 2: Message Driven Polling behavior.

In short, this novel mechanism has improved the performance of real-time capturing significantly since differences between successive screen snapshots are usually very tiny; furthermore, there are no data transferred from display memory to system memory if the full screen did not change.

COMPRESSION ALGORITHMS

This chapter proceeds to present an algorithm to compress screen snapshots and compose them into a video clip after capturing them from display memory by *Message Driven Polling*.

The Proposed Data Compressor

A screen snapshot has invariant primary color since colors in a screen snapshot are usually quite simple and repeated, such as the background of the desktop is blue, the background of the window is white; in addition, there are still some black fonts and gray toolbars, and this property would be highly critical to our compressor. As described above, the compressor should be designed to eliminate redundant repeated colors. In addition, the colors in screen snapshot are usually primary colors, such as blue, white, gray, etc; therefore, putting some primary colors in an extra table could save much space. In our system, a preprocessor had been executed in a busy computer for 24 hours to determine 256 (2^6) colors which have highest frequency of occurrence, and the capacity of table would also be 256.

Furthermore, Figure 3 illustrates several types of compression. In (a), if a color are also in table and occurs continuously, the compressor will fill the SIZE field with its continuous occurrences and set bit T to 1. If the occurrences exceed 2^6, the bit Z will be set to 1 and then follows another byte to represent remaining occurrences, this situation would like (b). On the contrary, if this color were not in table, the real RGB would be stored, like (c).

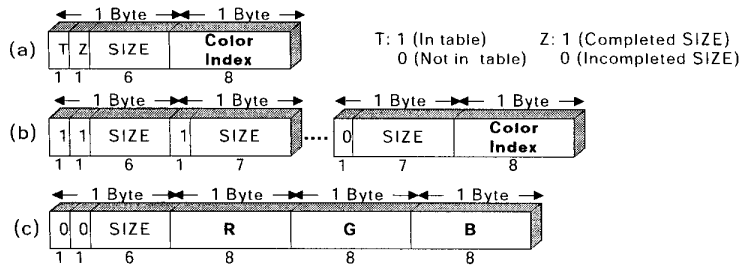

Figure 3: Format of compression file.

RESULT

We evaluated our proposed approach on several data sets. Our goal is to test and validate the approach for real-time capturing. Moreover, some comparisons between our proposed approach and other existing systems were proceeded.

Data Measurement

Figure 4 shows the data measurement of *Full Screen Capture* as well as *Message Driven Polling*. It illustrates the amounts of data that are transferred from display memory to system memory under the resolution of 1024×768 True Color. First, Curve 2 demonstrates the capacity of expansion bus that we have mentioned in previous chapter. Apparently, we can see Curve 1 that describes *Full Screen Capture* needs more capacity to achieve real-time since the slope of Curve 1 is much greater than the slope of Curve 2. Nevertheless, the slopes of Curve 3 and Curve 4 which represent our proposed *Message Driven Polling* are both less than the slope of Curve 2; in addition, Curve 3 illustrates the worst case of *Message Driven Polling*. The above evidence confirms that our proposed approach can achieve real-time in every situation.

Performance comparison

Most of the existing screen capture applications cannot achieve real-time capturing, and we choose FlickerFree Winstructor and Microsoft Media Encoder 7.0 to compare with our proposed system*. Table 1 shows the results of comparison; these results would prove that our approach performs much better works on FPS as well as on compression ratio than others. Furthermore, even in very high resolution, such as 1600×1200 and above, our approach still can achieve real-time since the performance of our method of real-time capture does not significantly affected by resolution adjusting.

---

* *The times in this paper were all measured by Microsoft Windows 2000 OS and Intel Pentium III 500 processor with GeForce 32MB VRAM graphics card and 256MB RAM under the resolution of 1024×768 True Color (32 bits).*

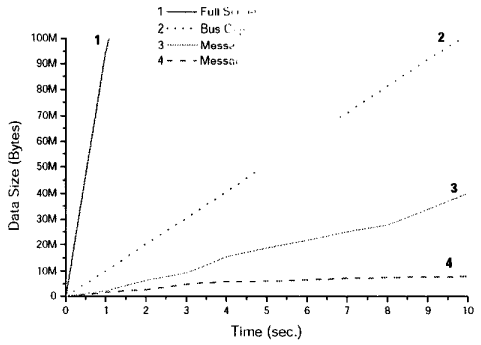

Figure 4: Data measurement of Full Screen Capture and Message Driven Polling.

		Winstructor	Microsoft Media Encoder	***Our System***
640×480	Avg. FPS	9.1	9.6	***30.3***
	Comp. Ratio	1 : 76	1 : 114	***1 : 671***
800×600	Avg. FPS	5.7	6.2	***30.4***
	Comp. Ratio	1 : 109	1 : 99	***1 : 694***
1024×768	Avg. FPS	N/A	5.6	***30.1***
	Comp. Ratio	N/A	1 : 117	***1 : 701***
1600×1200	Avg. FPS	N/A	N/A	***30.4***
	Comp. Ratio	N/A	N/A	***1 : 754***

Table 1: Performance comparison of our approach and other systems.

CONCLUSION

In this paper, we proposed a real-time high performance full screen capture system which is focus on an ability of real-time capturing full screen sequences and generating a video clip with 30 frames per second under very high resolution like *1600× 1200*. The experimental results have showed that our system framework achieved better performance than other existing systems. More work is needed to add speech component and develop a more efficient data compressor.

References

[1] M. Gellevij, H. V. D. Meij, T. D. Jong, and J. Pieters, "The Effects of Screen Captures in Manuals: A Textual and Two Visual Manuals Compared," *IEEE Transactions on Professional Communication*, Vol. 42, No. 2, Jun 1999, pp.77 - 91.

[2] A. M. Memon, M. E. Pollack, and M. L. Soffa, "Hierarchical GUI Test Case Generation Using Automated Planning," *IEEE Transactions on Software Engineering*. Vol. 27, No.2, Feb 2001, pp.144 - 155.