Forensically Sound Retrieval and Recovery of Images from GPU Memory

Yulong Zhang(✉), Baijian Yang, Marcus Rogers, and Raymond A. Hansen

Department of Computer and Information Technology, Purdue University,
West Lafayette 47907, USA
{zhan1621,byang,rogersmk,hansenr}@purdue.edu

Abstract. This paper adopts a method to retrieve graphic data stored in the global memory of an NVIDIA GPU. Experimentation shows that a 24-bit TIFF formatted graphic can be retrieved from the GPU in a forensically sound manner. However, like other types of Random Access Memory, acquired data cannot be verified due to the volatile nature of the GPU memory. In this work a Color Pattern Map Test is proposed to reveal the relationship between a graphic and its GPU memory organization. The mapping arrays derived from such testing can be used to visually restore graphics stored in the GPU memory. Described 'photo tests' and 'redo tests' demonstrate that it is possible to visually restore a graphic from the data stored in GPU memory. While initial results are promising, more work is still needed to determine if such methods of data acquisition within GPU memory can be considered forensically sound.

Keywords: GPU forensics · Graphic recovery · Volatile memory acquisition

1 Introduction

With the advances of Graphics Processing Units (GPUs) technology and GPU-accelerated computing, many software applications begin to outsource matrix related computations to GPUs to facilitate operations such as graphic and image rendering and data analytics [16]. For example, WinZip began support of GPU computing beginning in version 16.5 and experienced significant performance gains in version 18.0 by leveraging OpenCL technology [4, 9, 10].

On the other hand, the increased utilization of the GPU has introduced some serious security vulnerabilities. As of March 2015, the memory size of a GPU can reach as large as 16 GB, potentially opening the door to information hiding [14]. Some malware and worms have also utilized the GPUs as their secret hideout to obscure themselves from anti-virus and anti-malware programs [22]. Even worse, malicious software can also be designed to steal sensitive information from other processes by accessing unauthorized data stored in GPU memory [12].

While GPUs have become toys for cyber criminals, not much work has been done in the field of GPU forensics. It is therefore the purpose of this study to (1) propose a method to retrieve graphic data stored in GPUs main memory (usually 2 GB or more) and visually reconstruct the graphic from the retrieved data; and (2) prove or disprove

J.I. James and F. Breitinger (Eds.): ICDF2C 2015, LNICST 157, pp. 53–66, 2015.
DOI: 10.1007/978-3-319-25512-5_5

the method is forensically sound. Though the experiments are limited to the Windows Photo Viewer application and NVIDIA GeForce GPUs, the methodology proposed here could be extended to general GPU forensics.

The paper is organized as follows: In Sect. 2, related work is introduced followed by our proposed research method in Sect. 3. Experiments and collected data are discussed in Sect. 4. And finally, conclusions are presented in Sect. 5.

2 Literature Review

This section reviews volatile forensics, GPU fundamentals, and related work on retrieving data from GPUs.

2.1 Volatile Forensics Status and Trends

Volatile data, such as system memory, "provides a great deal of information about the system's runtime status at the time of, or just after, an incident" [20]. These include network connections and configuration, running processes, open files, login sessions, and operating system time [13]. *Volatile forensics* is, therefore, important because it has the potential to reveal critical information about criminal activities, such as passwords used for encryption, indications of anti-forensic techniques, and residual memory of a malware application that would go unnoticed otherwise. In addition, volatile forensics is often recommended over hard drive forensics to save time and cost [2].

Conventional practices fail to protect volatile data as potential digital evidence, as investigators are advised to shut down and unplug the compromised computer when the evidence located in hard drive are retrieved [8, 19]. However, sufficient progress has been made in the field with volatile forensics so that it has been accepted in court [3]. Ring and Cole [18] first wrote a paper and developed software to address the issue of capturing data located in system RAM for forensic purposes. Additional tools were developed and evaluated in [7, 20, 21] on forensics of system RAM.

One of the key challenges in volatile forensics is that not all forensically sound principles defined in [1] can be satisfied without effort. There is the potential for these principles to be violated because data is volatile and operations may not be repeatable within the investigation [11]. As a result, no independent third party can retrieve the same results after the incident. Therefore, this principle should be changed as follows: An audit trail or other record of all processes applied to computer-based electronic evidence should be created and preserved. An independent third party should be able to check and review those documents and data preserved and agree that the evidence is not contaminated, and remains forensically sound [1]. This suggests that there will be an additional burden of proof on maintaining the chain of custody [5], and additional caution should be taken to prevent the evidence being tainted by the acquisition process(es), the audit process, or any anti-forensics tools [23].

It should be noted that existing techniques and tools on volatile forensics typically do not apply to GPU forensics. This is due to there being no defined file system within the GPU's RAM. Which means, where and how the graphic data is stored and organized

in the GPU memory (not the System RAM) are often elusive to the OS and other applications. Existing volatile forensics approaches and tools on CPUs, coprocessors, and system RAM are not able to reveal any information stored inside the memory space of GPUs. In addition, metadata, such as the header information of a graphic file may not exist or is difficult to locate inside the GPU memory, making the investigation even more challenging.

2.2 GPU Structure and Memory Management

A typical GPU is made up of a control component, a cache component, an Arithmetic Logic Unit (ALU) and a memory component. An ALU in the GPU is often called a Stream Multiprocessor (SM), which contains multiple Stream Processors (SP). The memory component of a GPU, like that of a CPU, has different parts operating at different speeds. For example, an NIVIDIA GPU usually contains five types of memory. From the slowest and largest to the fastest and smallest, they are: global memory, constant memory, texture memory, shared memory, and registers memory. Global memory is the only type of memory that is accessible to both the GPU and the CPU and it is also the type of memory examined in this research.

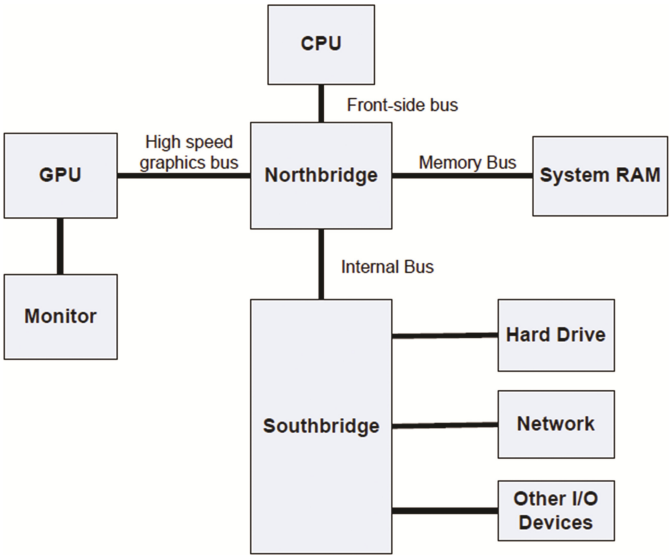

Fig. 1. Illustration of hardware connections used to transfer data from a hard disk to System RAM to Global GPU Memory

Figure 1 describes how components inside a PC are connected. The GPU is connected to and controlled by the CPU via Northbridge. In the case of NVIDIA, the GPU can be manipulated by CUDA commands. The GPU is also directly connected to the monitor via the external bus. When an application needs to utilize the GPU for processing:

1. Image data is located on the hard drive and copied to system RAM
2. Data is then copied to the GPU memory (often in different format)
3. GPU performs the computation and places the results into global memory
4. Data is displayed on monitor or placed in system RAM

For example, OpenGL will always send the data to the monitor without transmitting it back to system RAM. This indicates that forensics of system RAM alone will *not* be sufficient to capture all the live evidence because not all the data will be stored in system RAM.

2.3 Retrieving Data from GPU

Breß et al. [6] and Lee et al. [14] have both attempted to retrieve data from an NVIDIA GPU memory. They determined that it is not possible to access a certain memory space via the physical memory address where it is located. The methods they proposed were similar and described as follows:

1. Get GPU memory space information using `cudaMemGetInfo()`
2. Get access to the data of closed programs and the location it was freed from by allocating the entire memory space using `cudaMalloc()`
3. Copy all the data in GPU global memory using the `cudaMemcpy()`
4. Free the memory space allocated by using the `cudaFree()`

According to CUDA documentation, the four API methods listed above do not modify the content stored in GPU. This process is therefore believed to be able to copy data from GPU memory without contaminating it [6].

Since there is the potential that the process of retrieving GPU data could contaminate evidence stored in system RAM, it is suggested that forensic analysis of the GPU should be done only after forensic analysis of system RAM.

3 Methodology

The research question of this investigation is to explore whether it is possible to use forensically sound methods to recover the graphic data produced by the Windows Photo Viewer and captured from an NVIDIA GPU memory space.

This study breaks the question into three phases. Phase I demonstrates a forensically sound method exists to capture GPU data. Phase 2 illustrates how a graphic can be restored from the GPU data. And Phase 3 tests the validity of method.

The study used the Windows 8 Windows Photo Viewer to generate the data in the GPU global memory. An NVIDIA GeForce 650 was chosen as the testing hardware simply because it was the GPU available in the laboratory environment. A GPU memory-dump application was also developed to test the functionality and validity of the recovery process.

3.1 A Forensically Sound Method to Collect Evidence from GPU

This research adopted the method from [6, 14] to retrieve data that is buffered text data or rendered graphics data from a GPU. However, changes were made to prove that known evidence could be captured in a forensically sound manner.

The first step was to prove that graphics data from the GPU memory could be captured directly. To do so, a graphic was opened from Windows Photo Viewer and then closed with no modifications or alterations. Because Windows Photo Viewer uses GPU acceleration, the graphic data was temporarily stored in the global memory of the GPU. Then, the four CUDA APIs described in Sect. 2.3 were called to copy the GPU memory content to the system RAM, and then saved to the external storage as the evidence.

When a GPU-accelerated application is closed, the data stored in the GPU global memory is not reset. Rather, the corresponding memory space is labeled as free and will be allocated to other running processes. As a result, a simple memory dump of everything in the GPU is not effective nor efficient because it contains a mixture of data from applications that are currently executing, as well as recently closed, while the focus of this work is to prove the evidence of a single application can be captured reliably, in a forensically sound manner. To avoid unnecessary work of identifying which data (and memory locations) belongs to which process, the global memory space is first 'cleaned' before the Windows Photo Viewer is launched, as shown in Fig. 2. The `cudaMemSet()` API was used to reset the contents of GPU global memory to 0. Note that the 'cleaning' stage will change GPU data and it is only needed in the lab environment to simplify the analyses. This step should *not* be performed in the actual live capturing process.

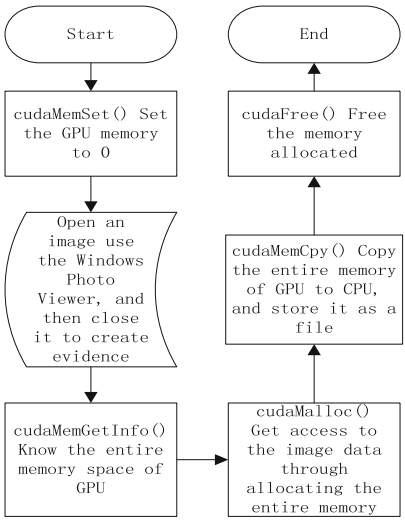

Fig. 2. Process model used to test the collection of graphic data from GPU memory

The basic process that guides the investigators on how to collect the evidence from the GPU's global memory has been introduced by both the [6, 14]. In this research, two changes were made in order to provide forensic reliability to the results. These differences are: (1) create known images before the experiment and use them as evidence; (2) clean memory and collect the evidence immediately after cleaning process to identify the relationship between the original graphics and the evidence.

To prove that data produced in Windows Photo Viewer are captured in GPU memory, a simple square test was designed. The square test draws three squares in 24-bit TIFF format with a different number of pixels. If the number of pixels captured matched with the number of the pixels in the original graphics, it meant the evidence had been captured correctly. If it did not match, further analysis was needed.

To prove the integrity and the reliability of the method, a test was designed to capture two graphics in three different format: jpg, bmp, and TIFF. The number of pixels, and the MD5 hash value of the captured data was studied to test if consistent results can be achieved repeatedly. The testing flow is described in Fig. 3.

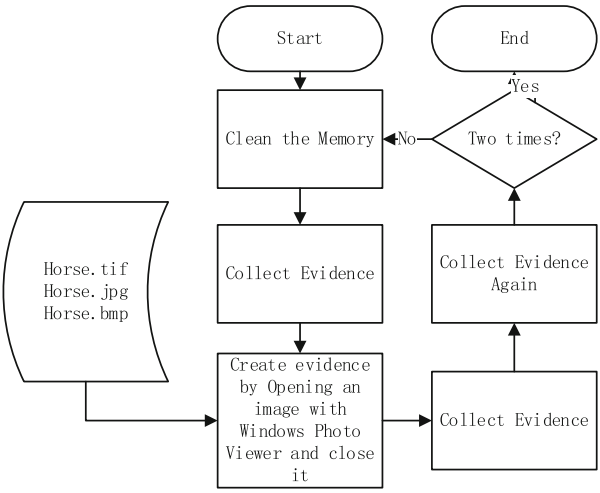

Fig. 3. Process model used to test the consistency of GPU memory capture

3.2 Recover TIFF Graphic from GPU Data

Both Windows Photo Viewer and CUDA APIs are proprietary applications. No public information is found on how graphic data are stored in the global memory of NVIDIA GPUs. To determine the memory allocation processes, Color Test, Line Test and the Color Map Pattern Test were designed. The lossless TIFF format was chosen because it is one of the popular formats that support raster graphics image and lossless compression.

In Color Tests, RGB values of 000000, 0000FF, 00FF00, 00FFFF, FF0000, FF00FF, FFFF00, and FFFFFF are used in the original graphics to compare with the captured

GPU data. In Line Tests, lines of different lengths, widths, and colors were used to relate how pixels are organized in GPU memory.

To better discover the patterns, a more sophisticated Color Map Pattern test were designed. In this test, a TIFF-format square with varying RGB color values was created (e.g. a 200 × 200 pixels square with RGB value incremented from 0 to 39999). It was opened in Windows Photo Viewer and the graphic data was retrieved from GPU global memory. If the graphic itself and the captured data can both be described in matrix arrays, then the transforming pattern can also be described in a matrix array. The Color Map Pattern procedures are illustrated in Fig. 4. Each mapping matrix is calculated as follows:

Let `Evi[]` denote the sequential RGB value of original graphic, `Ch[]` denote the sequential RGB value retrieved from GPU global memory and `Patn[]` denote a mapping array. For each pixel, i, ($0 \le i \le 39999$), there is `Ch[patn[i]] = Evi[i]`. In the designed experiment, both `Ch[]` and `Evi[]` are known, so each pixel of a mapping pattern can be calculated from the Eq. (1):

$$\text{patn}[i] = \text{Ch-1}[\text{Evi}[i]]. \tag{1}$$

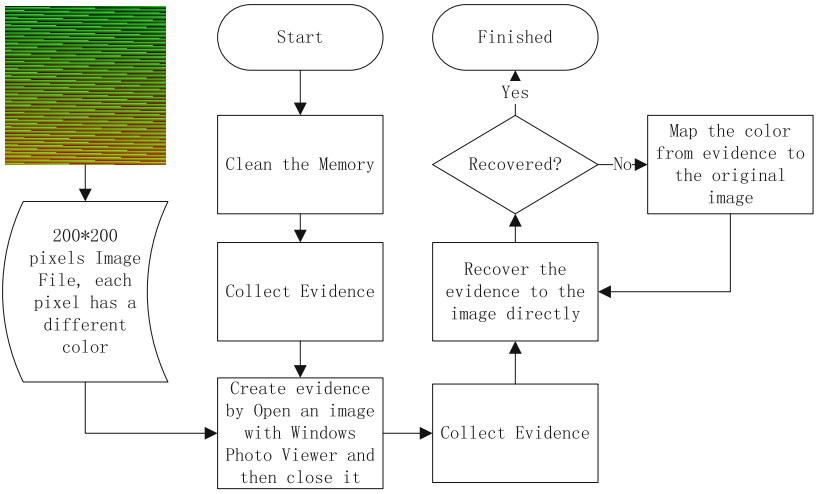

Fig. 4. Procedures used to implement proposed Color Map Pattern Test to discover mapping matrices

3.3 Validity of the Method

A series of experiments were designed to validate the graphic recovery process. The first experiment was a Photo Test where the data of images retrieved from the GPU were displayed on screen and compared against the original images. If they were visually identical, then the recovery approach was deemed correct. Otherwise it is not correct. The second experiment was the Redo Test where the operations were conducted on a different computer and then verified to determine if the same results were obtained.

4 Results and Discussions

This section describes the results of the experiments. The hardware platform is Intel Core i7- 3630QM CPU @ 2.4 GHz, 12.0 G DDR3 System RAM, NVIDIA GeForce 650 GPU. The OS running the application is 64-bit Windows 8.1 Enterprise.

In the reliability test, the experiments were conducted two times. Each time, the number of pixels of the graphic equaled the number of pixels in the evidence captured from GPU, as listed in Table 1.

Table 1. Results of Square Tests show #pixels in the evidance are idential to #pixels in the original graphic

Time	FileName	Size	#Pixels_Orig	#Pixels_Evi	Yes/ No
1	Red_tif_24_100w100h.tif	100 * 100	10000	10000	Yes
1	Red_tif_24_200w200h.tif	200 * 200	40000	40000	Yes
1	Red_tif_24_300w300h.tif	300 * 300	90000	90000	Yes
2	Red_tif_24_100w100h.tif	100 * 100	10000	10000	Yes
2	Red_tif_24_200w200h.tif	200 * 200	40000	40000	Yes
2	Red_tif_24_300w300h.tif	300 * 300	90000	90000	Yes

From Table 1, column Time represents the sequence of the experiments. FileName shows basic image attributes. For example, the first file named Red_tif_24_100w100h.tif is a 24-bit TIFF image that is red and has a size of 100 pixels width and 100 pixels height. Size shows the size of the images, #Pixels_Orig shows the number of pixels in the original image, and #Pixels_Evi shows the number of pixels in the evidence. Yes/No column shows whether or not the original pixels number equals the evidence pixels number.

From Table 1, it is observed that in each experiment, the number of pixels in the graphic equals the number of pixels in the evidence, as it is clear that in each time of the experiment, the #Pixels_Orig value equals the value of #Pixels_Evi.

From the data collected from the GPU, it is also observed that each pixel in the evidence will store its graphic data in a totally different sequence when it is compared to how the data is stored in the original image. This allocation process was also observed in Color Test as well.

In the integrity and reliability experiments, two different experiments were conducted. In each experiment, graphic data was retrieved three times. The MD5 hash value of captured GPU memory data were calculated and recorded in Table 2.

Table 2. Results of Integrity and Reliability Test show MD5 hashvalue of the evidence are identical within the same experiments and are different between each experiments.

Time	FileName	Format	OrgPixel	HashValue (MD5)
1	Horse	jpg	1024 * 706	2D6A2CBC0F319AEFCA612BBC0893C68F
1	horse_bmp_24	bmp	1024 * 706	2DB6AEECC32D9F3180C03FDA438C6029
1	horse_tif_24	tiff	1024 * 706	D8D437C7F2FFD79CA03618C4033D4059
1	Horse	jpg	1024 * 706	2D6A2CBC0F319AEFCA612BBC0893C68F
1	horse_bmp_24	bmp	1024 * 706	2DB6AEECC32D9F3180C03FDA438C6029
1	horse_tif_24	tiff	1024 * 706	D8D437C7F2FFD79CA03618C4033D4059
2	Horse	jpg	1024 * 706	B90596F6DFD288C3E0A949412F985DB1
2	horse_bmp_24	bmp	1024 * 706	95377AB8EC6ECE4CBD4A0F18A36AD490
2	horse_tif_24	tiff	1024 * 706	2ACD1A88AD8F21C3956084AE5A67D529
2	Horse	jpg	1024 * 706	B90596F6DFD288C3E0A949412F985DB1
2	horse_bmp_24	bmp	1024 * 706	95377AB8EC6ECE4CBD4A0F18A36AD490
2	horse_tif_24	tiff	1024 * 706	2ACD1A88AD8F21C3956084AE5A67D529

Column Time in Table 2 shows the sequence of the experiments. FileName demonstrates name of the file. Format shows the graphics format, OrgPixel demonstrates the sizes of the graphics, and HashValue (MD5) column shows the MD5 hash value of the evidence.

The results show that the hash values of the same digital GPU evidence were always consistent in all three captures. This means the integrity of the GPU data is preserved. However, the results are *not* repeatable: the same pictures will be mapped to different locations of GPU memory space with different patterns in a different test, and therefore produce totally different hash values. This is significant to understand that the same image data may be loaded into different memory locations on different systems. Additionally, it is likely that the image data may be loaded into different memory locations on the same system if the tests were run later. In both instances, new MD5 values should be calculated for those instances.

The summarized results of Color Test are shown in Table 3. The RGB values were stored differently in GPU memory. The findings suggest the color values are stored in the format of BGR(FF), as opposed to RGB. The individual color values maintain endianness.

In the Table 3, the Num represents the sequence of the experiments; Color column shows how the tested original 24-bit TIFF file stored the color information; Clean(Y/N) column shows whether or not the environment was cleaned before the evidence is produced and put into the GPU memory; and Description column demonstrates how the graphics pixel information was stored in the GPU main memory.

Table 3. Results of Color Test show RGB values were stored differently in GPU memory

Num	Color	Clean (Y/N)	Description
1	000000	Y	000000FF
2	0000FF	Y	FF0000FF
3	00FF00	Y	00FF00FF
4	00FFFF	Y	FFFF00FF
5	FF0000	Y	0000FFFF
6	FF00FF	Y	FF00FFFF
7	FFFF00	Y	00FFFFFF
8	FFFFFF	Y	FFFFFFFF

Based on the findings of Color Test, every pixel occupies 32 bits of space in the global memory of an NIVIDIA GPU. The first 24 bits represent the RGB values in the order of Blue, Green, and Red, and the last eight bits were padded with 1 s. However, each pixel of a graphic is not stored sequentially in the GPU memory. Figure 5 illustrates a typical scenario where recovering a graphic by sequentially reading the GPU memory data rendered a totally different graphic.

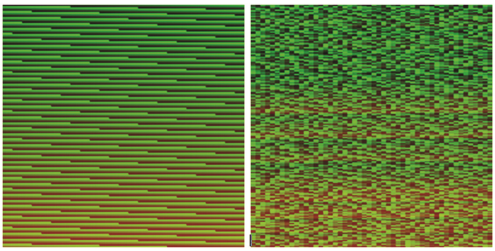

Fig. 5. Comparison between the original graphic (left) and the recovered graphic (right) by sequentially reading GPU memory data directly.

Many experiments were designed and tested to determine how the graphic data were organized in GPU memory. However, a well-defined simple rule may not exist, or at least was not easily found, to map the pixels of a graphic to their location in the global memory of a GPU. To overcome the barriers set by the unknown implementation details of both Windows Photo Viewer and CUDA APIs, the Color Map Patterns Tests were implemented to enumerate all the possible conversions. Specifically, a graphic with the same size as the evidence graphic was constructed. Each pixel of the testing graphic was given a unique RGB value so that it could be identified in the GPU memory dump. The

test was conducted repeatedly and the conversion matrices were computed. If a new conversion matrix was found, it was recorded as a candidate for the conversion. Due to time constraints, the test stopped when ten possible conversions were found.

Then a TIFF-formatted Photo was opened from Windows Photo Viewer and collected from the GPU. The test was conducted four times. Each time, the memory dump was manipulated by one of those ten possible conversions. The results showed that in tests 2, 3, and 4, one of the discovered conversions were able to completely restore the graphic. However, test 1 was not successful, indicating a different conversion matrix was used and was not yet found in the Color Map Pattern Test. Figure 6 shows the conversion results of the second photo test. Pattern A5 was able to completely restore the photo. Comparing with the original photo, the recovered photo was visually identical.

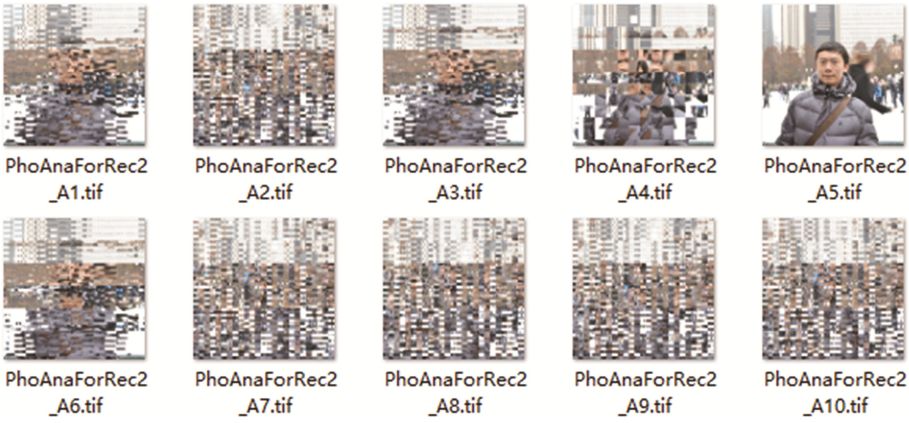

Fig. 6. Results of restored graphics after applying all 10 potential matching matrices (Test 2)

A5 pattern is one of the common patterns that was identified through the Color Map Pattern Test. What we have to mention is that this recovery process can only be done through visual selection at this time. More practical methods need to be discovered in the further research. However, to the author's opinion, this process can be used in NVIDIA GPU no matter how the series change. Windows Photo Viewer was selected only because we observed that the Windows Photo Viewer can produce evidence in the GPU memory. The author believes that investigators can use a similar method to recover any graphic data that is produced by a different application.

The Color Map Pattern Tests and the Photo Tests were conducted on a different computer with a similar NVIDIA GPU. The results showed a similar trend where three out of four Photo Tests could recover an image from one of those ten discovered conversion matrices.

The findings from all the experiments are listed as follows:

1. In Square Test and Color Test, the number of pixels captured from GPU memory space equals the number of pixels of the original TIFF graphic. And the pictures were identical between the original graphic and captured graphic. The only caveat is that the RGB value of a pixel will be reordered to BGR and padded with 8 bits of

1 s. This suggests that a 24-bit TIFF formatted evidence can be retrieved from GPU memory without any loss. It also suggests that a graphic takes a different form when stored in hard drive than stored in GPU memory. GPU forensic should not aim to prove two forms of data are binary identical, but instead, to prove the data captured is not contaminated and is visually identical. Additional tests are needed to verify if the same properties hold for pictures with different formats or of extremely large sizes.

2. Results obtained from the Integrity and Reliability Test showed that the GPU memory dump approach always generates identical MD5 hash values in the same experiment (repeatedly read GPU containing the same content). But the MD5 hash values are always different among experiments (repeatedly read GPU containing the same graphic by re-open it from Windows Photo Viewer). On the positive side, the results showed the proposed method is forensically sound and reliable in the same test. The results also showed the non-repeatability feature of volatile forensics.
3. The Color Map Pattern Test proved to be an effective technique at recovering a graphic from GPU memory dump. It was especially valuable when the mapping between a graphic and its GPU memory organization is unknown and difficult to discover. With the help of this test, potential transformation matrices were enumerated. The more transformation matrices, the more likely a graphic can be restored. If we further assume that the graphic evidence is a visually meaningful picture and no two potential conversions produce visually meaningful pictures, and then the proposed work can recover a graphic from GPU memory in a forensically sound manner.

5 Conclusions

In this work, we first explained the need of GPU forensics. Since it is possible for data stored in GPU to bypass System RAM and flow directly to the display devices, GPU forensics cannot be replaced by System RAM forensics. Existing work [6, 14] illustrated a process to retrieve data from GPU global memory but it was not thoroughly examined if it is forensically sound. A series of experiments were constructed in our work and proved that evidence opened by Windows Photo Viewer can be retrieved in a forensically sound manner in the same experiment. The method however is not repeatable by re-opening the same graphic as the hash values vary based on memory allocation determined at the time the application loads the image. The reliability within each experiment, however, holds true. That is, the hash values matched within each experiment, but not between experiments. The experiments also showed the graphic data stored in GPU memory is different from the format it is stored on the hard drive in terms of both color representations and order of the pixels.

Our analyses and experiments demonstrated that no simple and clear rules can be easily found between a graphic and how it is mapped to the global memory of a GPU. In an effort to visually restore the graphic data retrieved from GPU memory, a novel Color Depth Map Test was designed in our work. The test produced a number of conversion matrices by constructing a graphic that all the pixels can be uniquely identified. The tests

also showed that it is possible that a graphic can be visually recovered from one of the conversion matrices discovered from the Color Depth Map Test. The restoring method, as of right now, is not reliable and cannot be labeled as forensically sound yet due to the visual matching only. However, by trying more conversion patterns and conducting additional carefully designed experiments to prove its reliability, we are confident that the proposed method points the researchers in the right direction for discovering a forensically sound method to visually restore a graphic hiding in the GPU memory.

It should be noted that there may be limitations of these experiments due to the image files were only 200px × 200px 24-bit TIFF files. It is intended that future work will investigate this further, both in terms of image sizes as well as image formats. Since a TIFF is a raster format image, it is expected that other lossless raster image formats will have similar findings to these results. However, again, further work is needed to verify this supposition. Additionally, work should be done with lossy formats as well as vector-based graphic formats to identify the forensically sound recovery methods.

References

1. ACPO E-Crime Working Group: Good practice guide for computer-based electronic evidence. In: 7safe Information Security Website (2011)
2. Adelstein, F.: Live forensics: diagnosing your system without killing it first. Commun. ACM **49**(2), 63–66 (2006)
3. Aljaedi, A., Lindskog, D., Zavarsky, P., Ruhl, R., Almari, F.: Comparative analysis of volatile memory forensics: live response vs. memory imaging. In: Privacy, Security, Risk and Trust (Passat) and 2011 IEEE Third International Conference on Social Computing (Socialcom), pp. 1253–1258. IEEE Press, New York (2011)
4. AMD. http://web.amd.com/assets/customerreferenceprogrampackage2012/CRP%20Oct%202013%20WinZip%20Case%20Study.pdf
5. Bilby, D.: Low down and dirty: anti-forensic rootkits. In: Proceedings of Ruxcon (2006)
6. Breß, S., Kiltz, S., Schaler, M.: Forensics on GPU co-processing in databases research challenges, first experiments, and countermeasures. In: BTW Workshops (2013)
7. Campbell, W.: Volatile memory acquisition tools-a comparison across taint and correctness (2013). http://ro.ecu.edu.au/adf/115/
8. Center, C.C.: Steps for Recovering from a Unix or NT system compromise. Technical report, Software Engineer Institute (2001)
9. Claricesimmons. http://community.amd.com/community/amd-blogs/amd/blog/2013/10/30/the-new-winzip-18-with-accelerated-performance-for-amd-apus-and-gpus
10. Geeks3D. http://www.geeks3d.com/20111217/winzip-16-5-will-support-opencl-for-ultra-fast-compression-and-decompression/
11. Hay, B., Bishop, M., Nance, K.: Live analysis: progress and challenges. Secur. Priv. **7**(2), 30–37 (2009)
12. Jang, K., Han, S., Han, S., Moon, S.B., Park, K.: Sslshader: cheap SSL acceleration with commodity processors. In: Nsdi (2011)
13. Kent, K., Chevalier, S., Grance, T., Dang, H.: Guide to Integrating Forensic Techniques into Incident Response. NIST Special Publication, 800-86 (2006)
14. Lee, S., Kim, Y., Kim, J., Kim, J.: Stealing Webpages rendered on your browser by exploiting GPU vulnerabilities. In: 2014 IEEE Symposium on Security and Privacy, pp. 19–33. IEEE Press, New York (2014)

15. McKemmish, R.: When is digital evidence forensically sound? In: Ray, I., Shenoi, S., (eds.) Advances in Digital Forensics IV. Springer (2008)
16. NVIDIA. http://www.nvidia.com/object/what-is-gpu-computing.html#sthash.fYjRi2ZR.dpuf
17. Palmer, G.: A road map for digital forensic research. In: First Digital Forensic Research Workshop, pp. 27–30, Utica, New York (2001)
18. Ring, S., Cole, E.: Volatile memory computer forensics to detect kernel level compromise. In: López, J., Qing, S., Okamoto, E. (eds.) ICICS 2004. LNCS, vol. 3269, pp. 158–170. Springer, Heidelberg (2004)
19. Service U.S. S.: Best practices for seizing electronic evidence (2007). http://www.treas.gov/usss/electronic_evidence.shtml
20. Sutherland, I., Evans, J., Tryfonas, T., Blyth, A.: Acquiring volatile operating system data tools and techniques. ACM SIGOPS Operating Syst. Rev. **42**(3), 65–73 (2008)
21. Urrea, J.M.: An analysis of Linux RAM forensics. Unpublished Doctoral Dissertation, Monterey, California, Naval Postgraduate School (2006)
22. Vasiliadis, G., Polychronakis, M., Ioannidis, S.: GPU-Assisted Malware. Int. J. Inf. Secur. **14**(3), 289–297 (2010). http://dl.acm.org/citation.cfm?id=2777077
23. Wang, L., Zhang, R., Zhang, S.: A model of computer live forensics based on physical memory analysis. In: 2009 1st International Conference on Information Science and Engineering, pp. 4647–4649. IEEE Press, Nanjing (2009)