Real-Time Video Watermarking on Programmable Graphics Hardware

Alan Brunton and Jiying Zhao
School of Information Technology and Engineering (SITE), University of Ottawa
800 King Edward Ave., Ottawa, Ontario, Canada, K1N 6N5
e-mail: {abrunton,jyzhao}@site.uottawa.ca

Abstract

In this paper, we propose a real-time video watermarking system on programmable graphics hardware. Real-time video watermarking is important to the use of digital video in legal proceedings, security surveillance, new reportage and commercial video transactions. The watermarking scheme implemented here is based on Wong's scheme for image watermarking, and is designed to detect and localize any change in the pixels of any frame of the incoming video stream.

We implement this scheme for real-time operation on programmable graphics hardware. The Graphics Processing Units (GPUs) found on many modern commodity-level graphics cards have the ability to execute application-defined sequences of instructions on not only geometric primitives, defined by vertices, but also on image or texture fragments mapped to rasterized geometric primitives. These fragment programs, also known as fragment or pixel shaders, execute in hardware and in parallel on the GPU for each fragment, or pixel, that is rendered, making the GPU well suited for image and video processing.

We illustrate real-time performance, low perceptibility, and good bit-error rates and localization by way of a general testing framework that allows straightforward testing of any video watermarking system implemented on programmable graphics hardware.

Keywords — *real-time watermarking; image and video watermarking; programmable graphics hardware.*

1 Introduction

Content authentication is important in many real-time video applications, such as the use of video in legal proceedings or news reports, or as raw information for law enforcement or intelligence analysts. In each of these cases, it is important that users have a high degree of confidence that the video they are examining has not been altered in any material way since it was captured. As is the case for many media, digital video is more easily modified than analog video. Embedding a fragile watermark in digital video allows one to detect any changes made to the video by the absence of the watermark, which is destroyed when the video is altered, hence the term fragile watermark. By contrast, a robust watermark is persistent against alterations to the digital content, in this case video. Instead of content authentication, robust watermarking is used for copy tracing, owner identification and to allow advertisers to verify that their commercials are being run by television stations in the volume for which they have paid [1].

We present a fragile watermarking system for real-time content authentication and tamper localization in video using a modified version of Wong's scheme for image watermarking [2]. We implement our system using programmable graphics hardware found in many modern PCs. The Graphics Processing Units (GPUs) on modern video cards are effectively high-speed, highly parallel, single instruction multiple data (SIMD) streaming processors that operate in parallel to the CPU. Furthermore, modern video cards have their own memory space, in many cases in the range of 128 or 256 MB. These properties make graphics hardware a platform well-suited for video processing applications including, as we show here, video watermarking.

2 Background and Related Work

In this section we discuss content authentication watermarking schemes for images and video relating to our system in Section 2.1. Because many video watermarking schemes are based on image watermarking schemes, image watermarking is also relevant to our system. We then discuss previous work done on general purpose computing on the GPU in Section 2.2.

2.1 Image/Video Watermarking

In [2] Wong presents an image authentication watermarking scheme, which inserts a binary watermark into the least significant bit (LSB) of each pixel in the image. The original image is partitioned into blocks and each block is modified by setting the LSB of each pixel in the block to zero. Then a cryptographic function is computed from the modified block and the width and height of the image. The watermark for the block is then computed by taking the bit-wise exclusive or (XOR) of the binary output of the hash function with a binary image tiled to the same dimensions as the original image. The resulting binary watermark is then encrypted and inserted into the LSB of the modified block to form the watermarked image block.

In [1] Busch et al present a real-time watermarking system based on a modified version of the Zhao-Koch image watermarking system [3]. In this system, a single bit is encoded in an image block in the Discrete Cosine Transform (DCT) domain by permuting a selected sub-band of the DCT coefficients of that block so that the relationship amongst the coefficients comprising that sub-band encodes the bit. While Zhao and Koch describe their system as a robust watermarking method for images under JPEG compression, Busch et al apply their modified version as a semi-fragile watermarking system designed to survive MPEG-2 compression, but not modification of the video content.

Hartung and Girod [4] present a method for robust watermarking of uncompressed video and a compatible extension for MPEG-2 compressed video based on spread spectrum techniques.

2.2 General Purpose Computing on GPU

Graphics hardware in modern PCs operates on two types of data primitives: vertices and fragments. A vertex defines a 4-tuple typically in projective 3-space. Vertices are most commonly used to define polygonal geometry of a scene to be rendered. A fragment is a small section of a texture or render target, typically corresponding to a pixel of a texture image or render buffer (a block of memory to which the GPU can write). Fragments are also generally represented as 4-tuples, typically RGBA color data. Programmable graphics hardware allows developers to specify custom processing of both vertices and fragments.

Consumer-level graphics hardware has become more programmable in recent years as demand for better graphics in computer games has increased. GPUs have become sufficiently versatile that they are no longer suited only to rendering applications, but to any task which can be formulated in a data-parallel way. GPU-based algorithms require data-parallelism because they gain processing acceleration by operating on multiple vertices and pixels in parallel. In parallel programming terms they can perform a gather, but not a scatter. Gathering is possible because modern GPUs allow arbitrary access to up to sixteen input textures within a single fragment shader.

The increasing computational power of GPUs has led to the development of numerous algorithms which are not strictly graphics related, including many scientific computing algorithms. Iterative, grid-based methods are particularly well-suited to GPU implementations including computational fluid dynamics solvers for fluid flow [5] and cloud simulation and rendering [6]. Graphics hardware implementations of level set methods have also been developed for volumetric data [7] and for images [8]. Other image processing algorithms have also been implemented using graphics hardware, including the generalized Hough transform [9] and the Fast Fourier Transform (FFT) [10]. Hillesland et al implement an image-based modelling algorithm on the GPU [11].

3 Watermarking Scheme

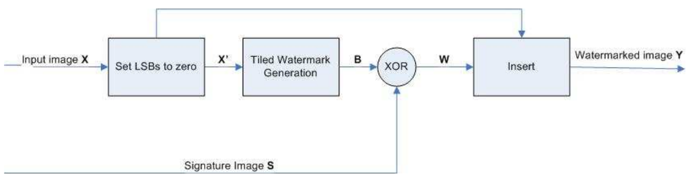

Figure 1: Watermarking Insertion Sequence.

The watermarking scheme used in this system is a simplified version of Wong's scheme for image watermarking [2]. The simplification of that scheme mostly has to do with removing the cryptographic components of the original system. Figures 1 and 2 illustrate the insertion and extraction processes of the system respectively.

An original image or video frame $\mathbf{X}$ with dimensions M by N is partitioned into blocks $\mathbf{X}_r$ of dimensions I by J for $r = 1, 2, ...MN/IJ$. For each block $\mathbf{X}_r$ a corresponding block $\mathbf{X}'_r$ is created by setting the LSBs of each pixel in $\mathbf{X}_r$ to zero. Then a binary image $\mathbf{A}$ is created (in this work we use a binarized sub-sampled version of $\mathbf{X}'$) and tiled to the same dimensions as $\mathbf{X}$ to create the image $\mathbf{B}$. A binary signature block S_r is generated for each block X_r from some message bit vector $\mathbf{M}$, and this signature is embedded by setting the LSBs of $\mathbf{X}_r$ to $\mathbf{S}_r \oplus \mathbf{B}_r$ thus creating the watermarked image $\mathbf{Y}$, where $\mathbf{B}_r$ is the corresponding block of $\mathbf{B}$ and $\oplus$ denotes a bitwise exclusive or operation.

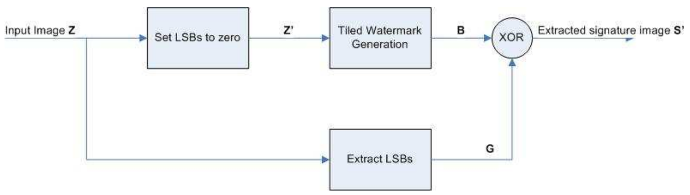

Figure 2: Watermarking Extraction Sequence.

The extraction process is very similar to the insertion process. A potentially watermarked image $\mathbf{Z}$ has its LSBs extracted and stored in a binary image $\mathbf{G}$, and then its LSBs are set to zero to form $\mathbf{Z}'$. Then the watermark tile $\mathbf{A}$ is generated from $\mathbf{Z}'$, and is tiled to form $\mathbf{B}$. The extracted signature image $\mathbf{S}'$ is computed as $\mathbf{B} \oplus \mathbf{G}$.

4 Real-Time Implementation on GPU

The watermarking scheme described in Section 3 was implemented for operation on the GPU. Because the geometry (a single viewport-aligned quadrilateral) does not change and it does not need to be perspectively projected, a single, simple vertex shader is used for each step of both insertion and extraction. This vertex shader simply sets the correct texture coordinates for the four corners of the viewport.

A fragment shader was written to set the LSBs of an incoming video stream to zero and to generate a tiled watermark image from an input video stream. A fragment shader was written to perform an exclusive or of an input signature image and a tiled binary watermark image and insert the result into a video stream. Similarly, a fragment shader was written to perform an exclusive or of the LSBs of an input video stream and a tiled watermark image and write the result to an output signature image.

The input signature image is generated from an input message (a 96-bit string identifying the video source and frame) in software and then sent to the graphics card as a texture for use by the GPU. The output signature image is written to by the GPU and read back into main memory where the message bits are extracted in software.

Current GPUs do not allow bit-wise operations. All texture data loaded in fragment shaders are converted to floating-point 4-vectors. Unsigned fixed-point texture data is scaled to the range [0, 1]. As a result, certain operations in this scheme must be emulated. The exclusive or operation is computed as follows: $A \oplus B = (1 - AB)max(A, B)$ where A and B are either 0 or 1. Reading and writing

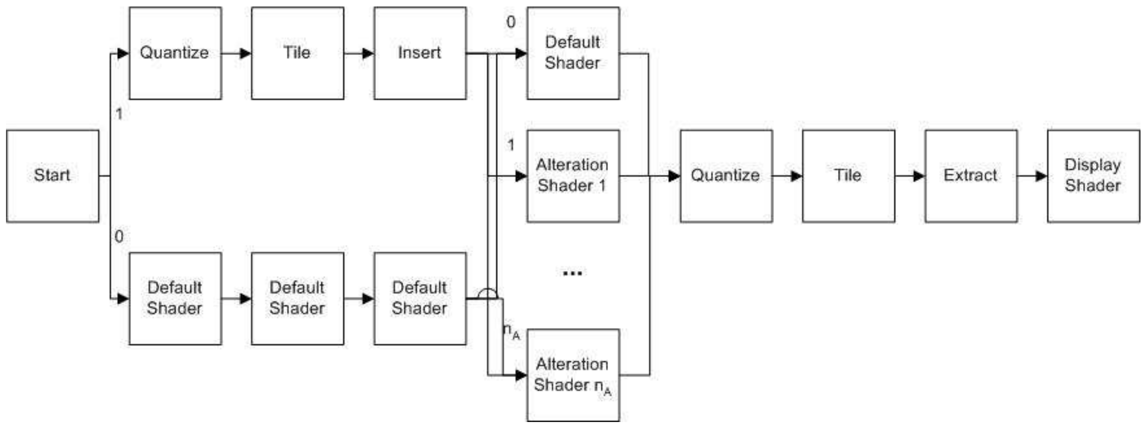

Figure 3: Watermarking testing graph.

the LSBs of an image were emulated by scaling by powers of two. If the original input image is stores values with k bits, then setting LSBs in the fragment shader programming environment is effectively quantizing the input image by $2^k - 1$.

5 Testing

A testing framework was developed to allow fast and flexible testing of any watermarking scheme implemented for GPU operation. The testing framework takes the form of a directed graph. At each node in the graph, a fragment shader executes on specified input textures. A start node branches either to a series of nodes corresponding to the insertion fragment shaders, or to a node that does nothing. The nodes recombine into one of $n + 1$ nodes consisting of a node that does nothing to the possibly watermarked video frame, and n nodes that alter the frame in different ways. These nodes converge and proceed through a series of nodes corresponding to the extraction shaders and a display shader. Choosing different paths through this graph tests the system in different ways. As noted in Section 4, before the first insertion shader the signature texture is generated, and after the final extraction shader the message is read from the signature texture. Also during this process, we generate test output for insertion and extraction times and bit error rates. The graph for this is shown in Figure 3. The graph could be modified to test other watermarking schemes.

6 Results

Testing was conducted on two systems denoted as *System 0* and *System 1*. *System 0* refers to a system consisting of an AMD Athlon XP 1900+, an ATI Radeon 9500 Pro, and 512 MB of RAM running Windows 2000, receiving a 320 by 240 video stream from a Webcam. *System 1* refers to a system using an Intel P4 3.4 GHz, an NVidia GeForce 6800, and 1 GB of RAM running Windows XP, receiving a 640 by 480 video stream from a Point Grey Research Dragonfly camera at 30 Hz.

Table I shows timing data for the two systems. In the table, N_I and N_E denote the number of insertions and extractions performed respectively, and t_I and t_E denote the average insertion and extraction times in milliseconds respectively. System denotes the system the test was conducted on. Table II shows the number of frames watermarked (Column 1) and the bit error rate (Column 2) for the entire sequence when attempting to extract the message from the video frame. Figures 4, 5, and 6 show the watermark display results for cases where no watermark has been inserted in the video frame, where a watermark has been inserted, and where a watermark has been inserted and the video subsequently altered. In each of these figures the top-left is the original video frame, the top-right is the potentially watermarked frame, the bottom left in the input signature image, and the bottom right is the extracted signature image.

System	N_I	t_I	N_E	t_E
0	246	1.45	425	0.04
1	595	2.59	812	0.02

TABLE I
Insertion and Extraction Times.

No. checks	BER
246	0.0
595	0.0
673	0.0

TABLE II
Bit Error Rates for different test sequences.

7 Conclusions

A real-time video watermarking system based on a simplified version of Wong's scheme for image watermarking implemented on programmable graphics hardware was presented. The system was shown to have good bit error rates and low insertion and extraction times, as well as good lo-

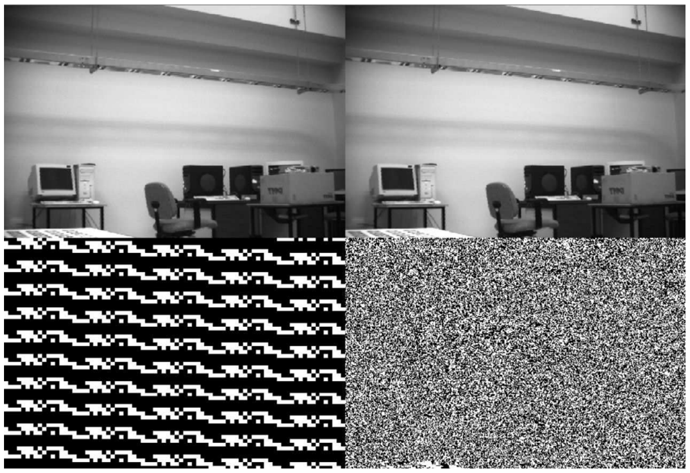
Figure 4. The watermark display results with no watermark inserted into the video stream.

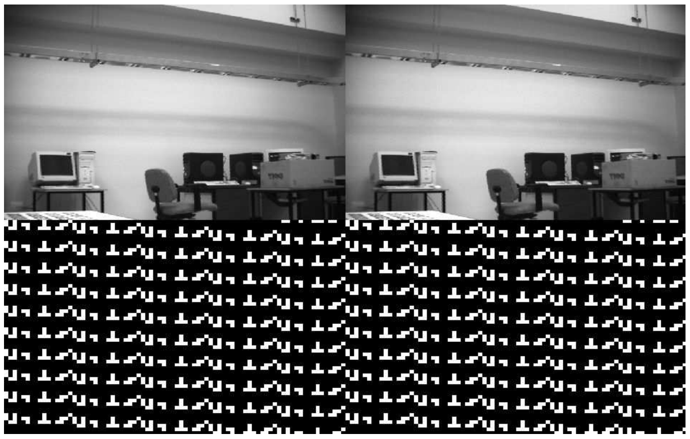
Figure 5. The watermark display results with a watermark inserted into the video stream.

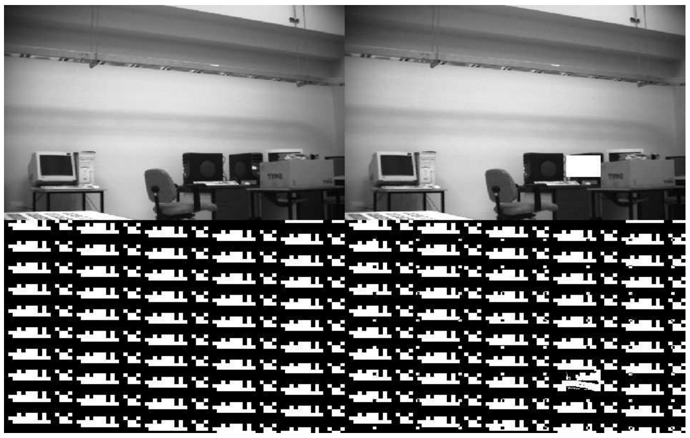
Figure 6. An example of localization of an alteration made to a watermarked image.

calization.

Possibilities for future work include implementing a more advanced watermarking system, preferably one that could persist through compression, and implementation of the cryptographic components of the watermarking system. Any future work should include more comprehensive and realistic testing.

References

[1] C. Busch, W. Funk, S. Wolthusen, *"Digital Watermarking: From Concepts to Real-Time Video Applications," IEEE Computer Graphics and Applications*, vol. 19, no. 1, pp. 25-35, January/February 1999.

[2] P. W. Wong, *"A Public Key Watermark for Image Vertification and Authentication," Proc. International Conference on Image Processing*, 1998.

[3] J. Zhao, E. Koch, *"Embedding Robust Labels into Images for Copyright Protection," Proc. International Congress on Intellectual Property Rights for Specialized Information, Knowledge and New Technologies*, Vienna, Austria, August 1995.

[4] F. Hartung, B. Girod, *"Watermarking of Uncompressed and Compressed Video," Signal Processing*, vol. 66, pp. 283-301, May 1998.

[5] N. Goodnight, C. Woolley, G. Lewin, D. Luebke, G. Humphreys, *"A Multigrid Solver for Boundary Value Problems Using Programmable Graphics Hardware," Proc. Eurographics/SIGGRAPH Workshop Graphics Hardware 2003*, pp. 112-119, July 2003.

[6] M. Harris, *Real-Time Cloud Simulation and Rendering*, University of North Carolina Technical Report TR03-040, 2003.

[7] A. E. Lefohn, J. M. Kniss, C. D. Hansen, R. T. Whitaker, *"A Streaming Narrow-Band Algorithm: Interactive Deformation and Visualization of Level Sets," IEEE Transactions on Visualization Computer Graphics*, 10 (40), pp. 422-433, July/August 2004.

[8] M. Rumpf, R. Strzodka, *"Level Set Segmentation in Graphics Hardware," Proc. IEEE International Conference on Image Processing*, vol. 3, pp. 1103-1106, 2001.

[9] R. Strzodka, I. Ihrke, M. Magnor, *"A Graphics Hardware Implementation of the Generalized Hough Transform for Fast Object Recognition, Scale and 3D Pose Detection, Proc. IEEE International Conference on Image Analysis and Processing (ICIAP)*, pp. 188-193, 2003.

[10] K. Moreland, E. Angel, *"The FFT on a GPU," SIGGRAPH/Eurographics Workshop on Graphics Hardware 2003 Proceedings*, pp. 112-119, July 2003.

[11] K. E. Hillesland, S. Molinov, R. Grzeszczuk, *"Nonlinear Optimization Framework for Image-Based Modeling on Programmable Graphics Hardware," Proc. SIGGRAPH 2003.*