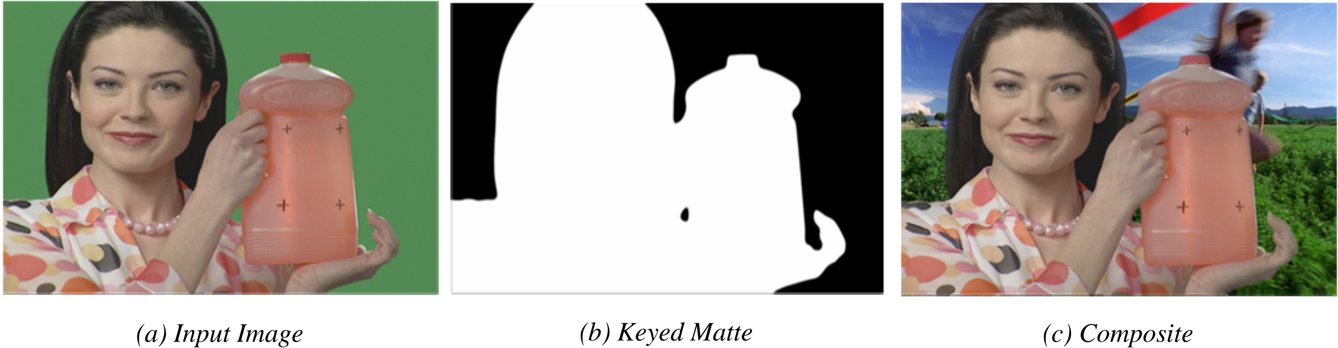
Real-Time Chroma Keying on the GPU

A White Paper by David Yamnitsky
Boris FX, Nov. 2009

Real-Time Chroma Keying on the GPU

David Yamnitsky
Boris FX



(a) Input Image *(b) Keyed Matte* *(c) Composite*

FIGURE 1: Process Summary

Abstract

A method is presented to efficiently compute a Chroma Key effect utilizing the Graphics Processing Unit (GPU) available in most modern computers. At the time of this writing the method is implemented in the BCC Chromakey Studio plug-in for Apple Final Cut and Motion under Apple's FxPlug plug-in format. The process consists of four separate image-processing algorithms, a chroma key matte computation, a spill suppressor, a matte choker, and a light wrap filter. Rather than executing the four filters sequentially, as would be appropriate on the CPU, the method reorganizes the flow of execution to maximize parallel processing and memory throughput, thereby taking full advantage of the parallel nature of GPU hardware. The method optimizes GPU throughput by performing image-processing tasks in parallel and minimizing texture lookup and global memory access.

Keywords: Chroma Key, Image Processing, GPU Computing, Real-time Video Processing.

1 Introduction

A common visual effect in video broadcast and post-production is known as a "Chroma Key," "greenscreen," or "bluescreen." It involves taking video of a subject shot in front of a solid color background and isolating the object from the monochromatic background. A green or blue background is usually selected because of its contrast with the red found in flesh tones. The isolated foreground is then composited over a different scene, simulating the effect of the foreground object being shot as part of that scene. This effect is very popular in areas of video broadcast and post-production such as news casting, weather reporting, and virtual environments, among others. However, to achieve real-time performance of this complex effect, prior implementations have included their own dedicated hardware, tuned specifically for keying. The object of this paper is to present a method to achieve the same performance and render quality on commodity hardware: GPUs.

Most modern computer systems include dedicated hardware for graphics known as GPUs. Driven by the demands of the video game and visualization industries, GPUs have become both extremely powerful, relatively inexpensive, and ubiquitous. GPUs are massively parallel computing hardware that contain hundreds of individual processors, all of which can run in tandem and are specifically tailored to SIMD processing, where the same instruction set is applied across multiple data streams simultaneously. In the past, GPUs have been limited to fixed or "baked in" instruction sets based on staple 3D graphics-specific algorithms such as primitive rendering and Phong lighting, but recently GPUs have become "programmable," allowing them to load and execute custom instruction sets across a large amount of data very efficiently. However, access to this computing power has in large part been limited to graphics APIs such as OpenGL and DirectX, which impose limitations to image processing applications such as abstracted data management, inconsistent data integrity, and fixed parallelization of

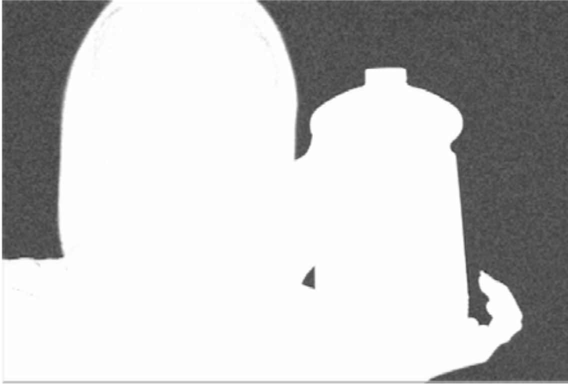


FIGURE 2: Matte Computation

FIGURE 3: Choked Matte

algorithms. Even more recently, the proliferation of GPGPU (General Purpose computation on the GPU) APIs such as OpenCL and NVIDIA CUDA, which expose the computational abilities of graphics hardware to applications beyond simple graphics processing have removed several of the limitations that graphics APIs impose and allow even further optimization of image processing algorithms.

2 Basic Description of the Algorithm

The algorithm consists of four components: a chroma key matte computer, a matte choker, a spill suppressor, and a light wrap filter. While it is possible to implement the proposed method using traditional fragment shaders in the context of a GPU graphics API, the highest performance and quality is achieved using a more modern GPGPU API because of the increased parallelism.

2.1 Chroma Key Matte Computation

The first step of the process is a chroma key matte computation, a per-pixel operation, which provides an initial filtering out of the background color using a proprietary algorithm. While the chroma key process may be sufficient with studio-shot professional images many real-life shoots produce footage that is hard to key. To solve this problem, we use a matte choker, which incorporates box spatial and noise reduction processing.

2.2 Matte Choker

The matte choker is defined by the concatenation of two operations in a user-defined order: a levels operation and a convolution of the matte. The basic levels operation manipulates an input intensity value by resetting the black and white points and linearly interpolating values in between. The levels operation attenuates low alpha values and accentuates high alpha values while still maintaining translucency at edges. To further smooth out the edges, a Gaussian convolution is applied to the matte. While any standard blur kernel may be used, the Gaussian is most appropriate for this task because of its ability to reduce hard edges to smooth gradients. The effect of the matte choker can be seen in **FIGURE 3**. Note the improvement in edge quality and the reduction and background noise.

2.3 Spill Suppression

A common problem in chroma keying, known as spill, is a tinting of the foreground object caused by light "spilling" from the monochromatic background. A proprietary GPU implementation of the spill suppression algorithm is employed.

2.4 Light Wrap

While the previous three filters produce a very high quality key, they do not provide a way to realistically composite the isolated foreground object into a new scene. For an object to appear as if it was shot as part of a scene, it needs to have edge lighting that matches the contents of the scene. To solve this problem, we use a light wrap filter, which composites a portion of a blurred background image around the edges of the foreground. The intent of the light wrap is to apply simulated background lighting at varying intensity along the edges of the image, so we apply a Gaussian convolution to the inverse of the alpha matte and find its union with the original matte. Using this method, the blurred background is applied to the foreground with highest intensity right at the edges and intensity decreasing by a Gaussian distribution away from the edges towards the interior of the foreground object. The result of the light wrap filter composited on black can be seen in **FIGURE 4**.

3 GPU Optimization

The steps detailed above produce a very high quality and

realistic key, but when implemented on the CPU sequentially as detailed in **FIGURE 5**, they do not provide very fast performance. Simply porting this implementation directly to the GPU, however, does not provide appreciably better performance either. While a modern GPU has global memory throughput an order of magnitude faster than that of main system memory, its compute power is *several* orders of magnitude greater, and GPU compute cores lack on-chip cache at close proximity to ALUs, so kernels need to be designed to minimize memory access and maximize computation to be efficient. In addition to maximizing the parallelism of computation, the primary GPU optimization technique that I employ is minimizing memory access, particularly access to image data.

3.1 Parallelization

The first GPU optimization technique employed, resulting in a flow of computation as detailed in **FIGURE 6**, is expanded parallelization beyond "embarrassingly parallel" distribution. By examining the dependencies of each of the comprising steps, it is concluded that three separate kernels can be launched to operate on different aspects of the effect simultaneously. One grid of threads operates on computing the chroma key matte computation and matte choker, while one grid applies the spill suppression algorithm and a third grid blurs the background image, all launching one thread per pixel processed. After the first grid finishes computing the matte choker, it synchronizes with the other two before computing the light wrap.

3.2 Kernel And Memory Consolidation

Even more consequential in increasing performance than enhanced parallelization, is kernel and memory consolidation. The first thread grid launched computes the per-pixel chroma key matte computation, performing one texture lookup in the input image per pixel processed. To minimize redundant access to this memory across multiple kernels, the kernel also computes both phases of the matte choker. While it is common practice to copy the matte into a separate buffer for manipulation and then applying the manipulated matte back to the input, on the GPU this is very inefficient as the effect will have to execute multiple kernels, synchronizing between all of them, and have the overhead of additional buffer storage. To remedy this an alpha blur is used, as opposed to a single channel convolution on a separate matte. While manipulating data in place, alpha blurring brings up the issue of memory coalescing, as most common GPUs read memory in word format.

4 Discussion and Conclusion

By manipulating all data in place across multiple kernels in parallel, the effect exhibits significantly improved performance on GPUs, realizing the ability to preview a key on a clip of HD in real-time on modern hardware. Future research to improve the performance of the effect would be best focused in two areas, first in convolution optimization and second in GPU memory bandwidth. The most time consuming part of computation is in Gaussian blur computation, which incurs multiple texture reads. In addition, despite memory read optimization the effect is still memory bound, so increased memory bandwidth from texture memory would improve the effect's performance.

FIGURE 4: Light Wrap

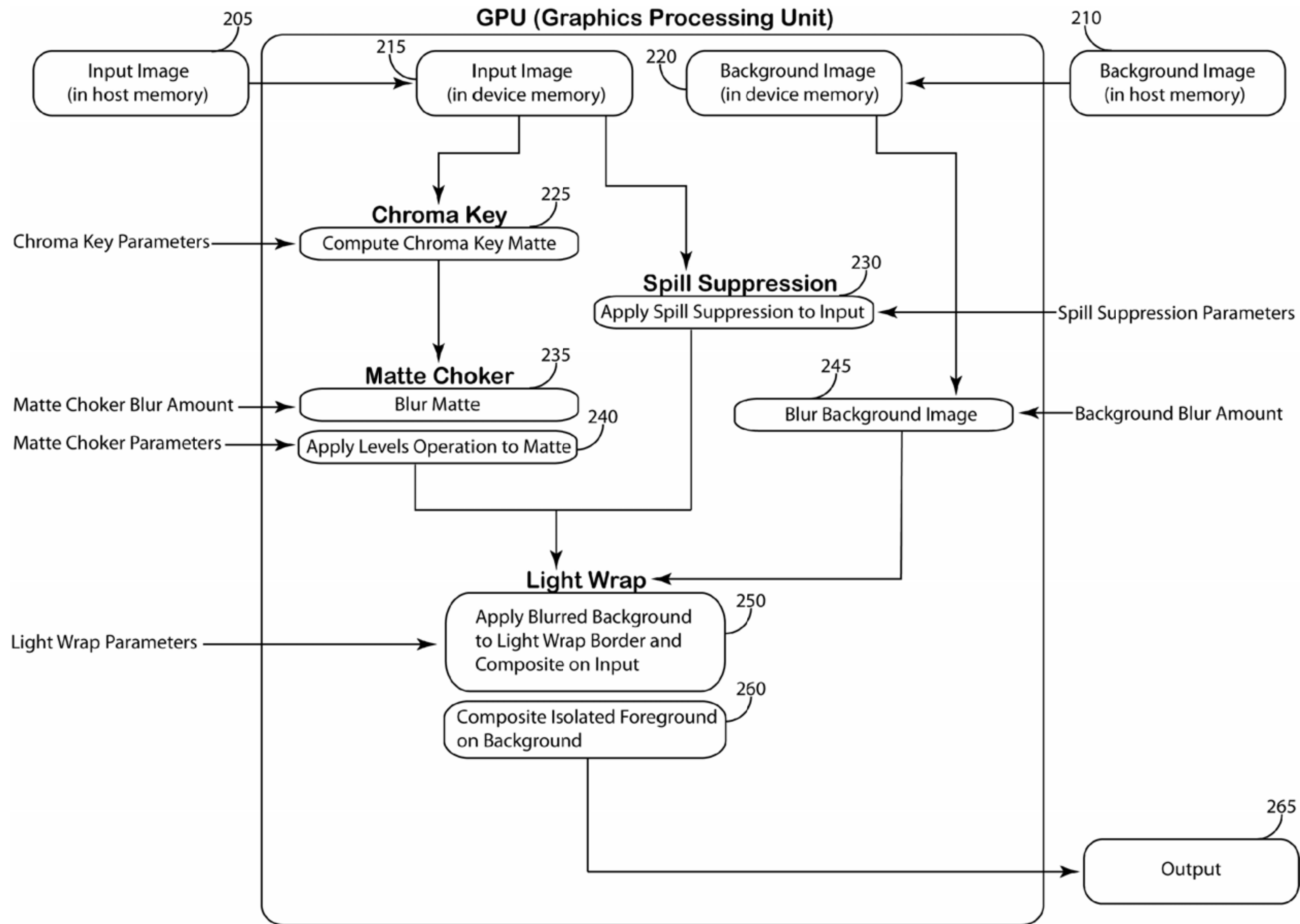


FIGURE 6: GPU Optimized Implementation

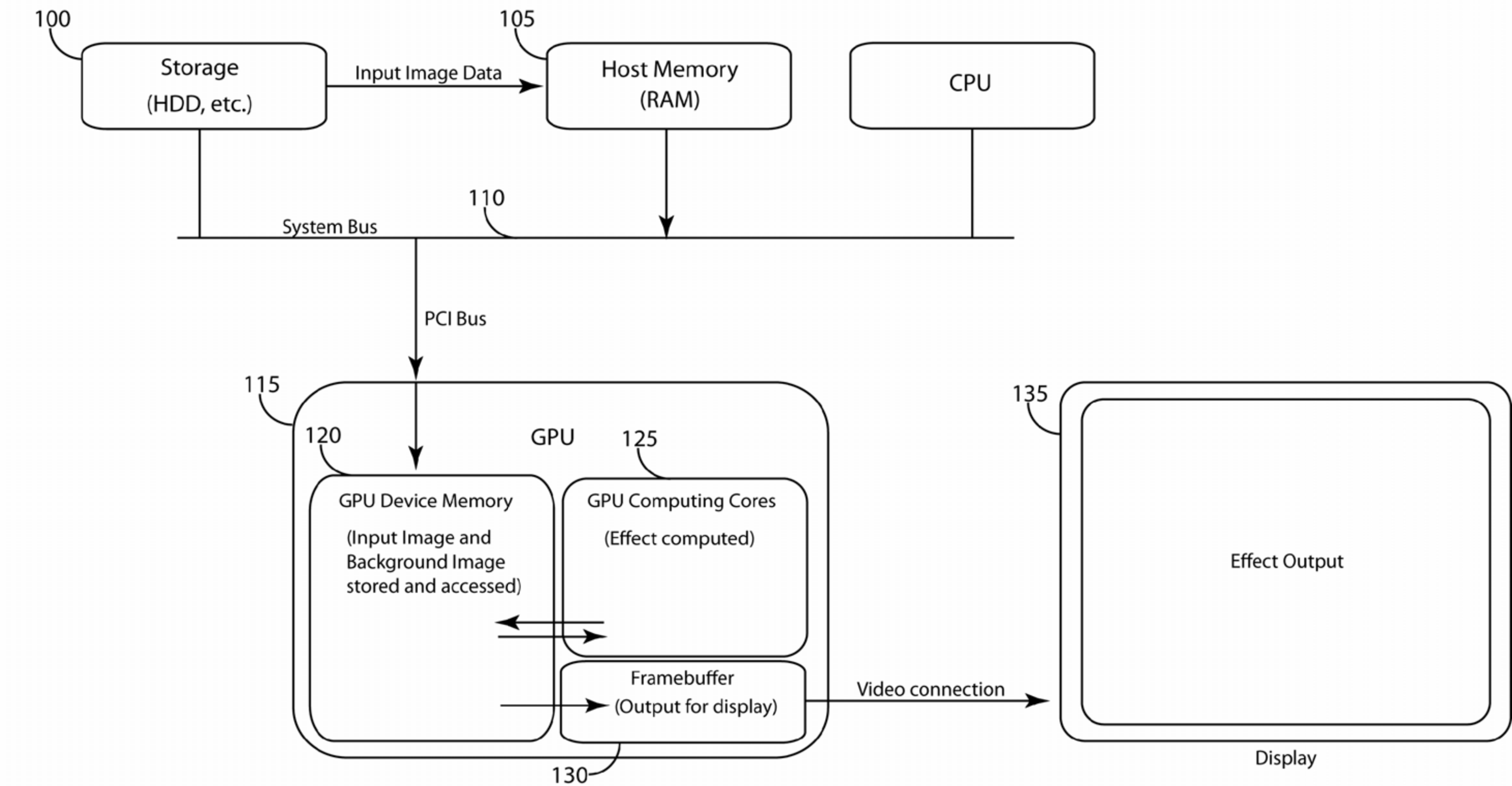


FIGURE 7: Hardware Summary