GPU-BASED BACKGROUND ILLUMINATION CORRECTION FOR BLUE SCREEN MATTING

Nicolas Ley and Christian Weigel

Institute for Media Technology, Technische Universität Ilmenau
Postfach 10 05 65, 98684 Ilmenau, Germany
phone: +(49) 3677-69 2757, fax: +(49) 3677-69 1255, email: christian.weigel@tu-ilmenau.de

ABSTRACT

Separation of foreground objects from an almost constant backing color for video applications is still a common problem ([1],[2]). For non-realtime situations there is a wide variety of different powerful mathematical approaches that can deal with most of the matting problems. For SD/HD studio realtime keyers most solutions are not applicable due to their algorithm complexity or high effort in user interaction. Excellent hardware keyers, such as Ultimatte™ work on most occasions, but even under controlled lighting in a blue-/greenscreen matting problems may occur, or creativity is limited by necessary lighting conditions. As a preprocessing algorithm for traditional chroma keying systems, we present a simple background illumination correction based approach for improving matting problems with uneven or poor lit blue-/greenscreens. Using the computational power of GPU computing (GPGPU [3]) the presented algorithm is realtime capable and offers an improvement for achievable mattes quality.

1. INTRODUCTION

For about sixty-five years the blue screen technique has been the method of choice for background replacement[4]. Using only color information, the blue or green background can be replaced by a different one. In television production this procedure is often referred to as chroma keying which differs in some slight ways from the process used for movie production. The technique is nowadays well-understood and used for a huge number of applications, for instance news or weather presentation, moderation of television journals or virtual studio production. Although excellent realtime keying solutions are available, there are some occasions where they do not provide satisfying results.

Figure 1: uneven lighting

The lighting of the background plays an important role how mattes will turn out satisfactory. Poorly lit blue-/greenscreens degrade the quality of the matte that can be extracted. Uneven lighting is a common problem, because it is very difficult to get the lighting both bright and uniform across a broad surface, figure 1 demonstrates this problem. Another frequent problem is a too bright or too dark lit background combined with uneven lighting (figure 8). Pulling a matte without high cleanup level is nearly impossible.

Figure 2: too dark and uneven lighting

Close up shots are not so vulnerable to lighting problems, but wide screen shots with a high amount of background and floor are problematic. Resulting mattes without cleanup or special techniques (e.g. automatic rotoscoping from postproduction plugins) show a really poor result. Sample results a shown in figure 3. Refining these raw mattes in a live keying system would lead to really hard edged results, due to a high cleanup level. Other limitations occur in the quality of the remaining shadows. The goal of a realtime capable preprocessing algorithm is to improve the basic image which then can be passed to traditional chroma key techniques. As a possible solution a simple approach using illumination estimation based algorithm is presented.

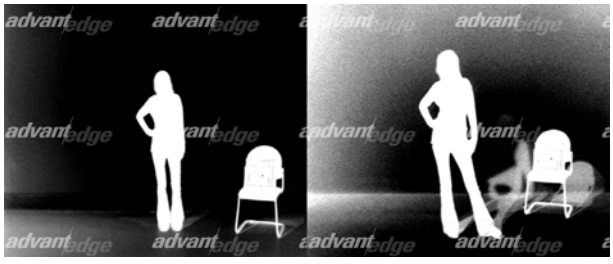


Figure 3: matting artifacts (Ultimatte™ without cleanup)

Using graphics hardware for general purpose computations is getting more and more popular, a vibrant community

of developers has emerged around GPGPU ([3]). For demonstration of the computational power of modern GPU architectures the presented algorithm is implemented as shaderprograms (introduction to GPGPU programming concepts [5]).

2. FUNDAMENTALS

2.1 Illumination estimation

Illumination estimation or shading correction is a very sophisticated subject. Especially edge preserving methods like anisotropic diffusion are common and widely researched. For the presented problem we outline an even simpler method of shading correction, avoiding the solution of complex equation systems, based on single-scale-retinex methods ([6]). The algorithm is based on the assumption that uneven illumination is an additive low frequency signal ([7]). Therefore gaussian low pass filtering could be used to extract the illumination distribution across the image. The Gaussian function $G(x,y)$ is defined by (σ defines the effective spread of the function):

$$G(x,y) = \frac{1}{2\pi\sigma^2}\exp[-(x^2+y^2)/2\sigma^2] \qquad (1)$$

Convolving an image with a gaussian kernel would delimit the spatial frequencies, resulting in loss of edge definition and averaging of intensity values. For large kernels the resulting image is the information of the present lighting. Normally shading correction is used for the whole image, we present a different approach. First the foreground is coarsely keyed, the hole is filled by a simple inpainting technique, now the gaussian convolution is applied.

2.2 General-Purpose Computation Using Graphics Hardware

GPUs are a compelling solution for applications that require high arithmetic rates and data bandwidths. GPUs achieve this high performance through data parallelism, which requires a programming model distinct from the traditional CPU sequential programming model ([5]). Additionally the gap between the computational power of the CPU compared to the GPU outperforms Moore's law:

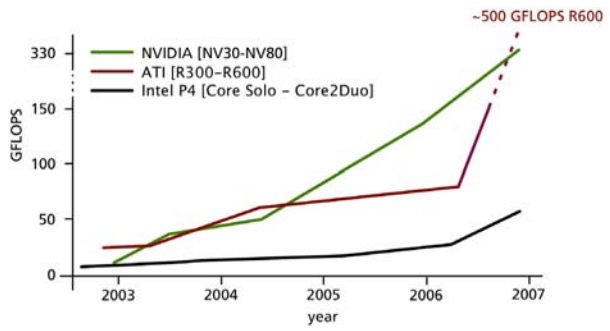


Figure 4: computational power of GPUs

The used paradigm is called stream processing. The stream programming model exposes the parallelism and communication patterns inherent in the application by structuring data into streams and expressing computation as kernels that operate on streams. Kernel functionality is implemented in high-level-language shader programs, CPU arrays are transferred to the GPU's memory and are represented as textures. Bottleneck is still the download/readback of the data. For outperforming the CPU computational power the data transfer times have to be considered. With the latest chipsets and graphic cards (NVIDIA nForce 680i chipset, GeForce 8800GTX) satisfying transfer rates for SD/HD720p images can be achieved (RGBA, 8 bit/channel):

format	download $[ms]$	readback $[ms]$
720x576 SD	1.3	1.0
1024x720 HD720	2.5	2.0
1920x1080 HD1080	6.6	5.3

Table 1: NVIDIA GeForce 8800GTX
max. download/readback rates

Before the date is available for processing on the GPU, data arrays have to be transferred to the graphic memory (download) and afterwards be retransferred to CPU memory (readback). With the common graphic processing APIs (OpenGL, DirectX) similar methods for this transfer for different purposes exist. As a possible bottleneck of algorithm implementations the following bandwidths (see table 2) have to be considered.

Card	FSB	Download	Readback
NV GF 8800 GTX	266	996	1060
NV GF 8800 GTX	333	1105	1285
NV GF 8800 GTX	400	1170	1468
NV GF 6600 GT	400	876	1101
ATI X1900 XTX	266	220	605

Table 2: Download and Readback in (Mb/s) (tested with gpubench[8])

3. ALGORITHM

The presented preprocessing algorithm consists of several buildings blocks, nevertheless the method is clear and efficient.

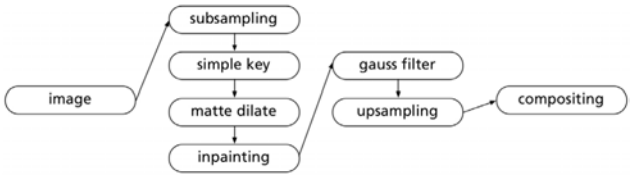


Figure 5: algorithm flowchart

The necessary steps for creating a corrected image with the computation using the GPU is outlined in the following enumeration:

1. download image to the GPU
2. subsample image
3. pull matte with simple chroma key with high cleanup
4. dilate the matte
5. automatic inpainting
6. gaussian convolution
7. upsample image
8. composite illumination corrected image

9. readback image or pull matte with a shaderprogram

After downloading the image to GPU's memory represented as texture, the first step makes use of the assumption that the illumination change is low-frequency. Therefore a subsampling of the image is possible, the low frequencies still remain. Dependent on the subsampling factor (subsampling by a factor of max. 4 is proposed) not only matte pulling and dilatation computation time benefit, furthermore smaller gauss kernels can be used.

(a) simple chroma key (b) cleanup & matte dilatation

Figure 6: matte pulling

As the channel for background estimation either luminance or the blue/green channel is used. A simple chroma key matte can be computed by channel per pixel min-max-Operations of the input image, resulting in a binary matte. A simple dilate operation is implemented as a filter kernel, the size depends on the subsampling level:

$$k(x,y) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad (2)$$

After this operations foreground elements are determined. Foreground mask and image are now used for a simple inpainting algorithm (overview at [9]). The simplest method for inpainting would be a linear interpolation between the edge pixels of the matte. A more sophisticated fast digital inpainting can be done by repeatedly convolving the region to be filled with a diffusion kernel (e.g. [10]). Convolving an image with a Gaussian kernel is therefore equivalent to isotropic diffusion (linear heat equation). Possible diffusion kernels are (variables (a,b,c) [10]):

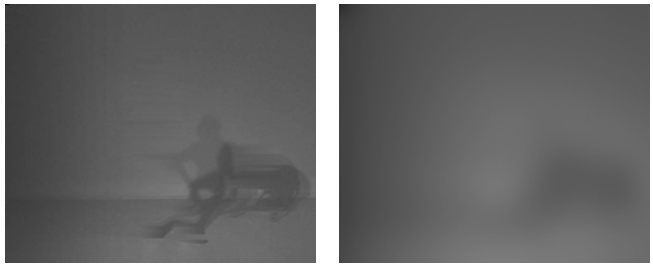
$$k(x,y) = \begin{bmatrix} a & b & a \\ b & 0 & b \\ a & b & a \end{bmatrix}, k(x,y) = \begin{bmatrix} c & c & c \\ c & 0 & c \\ c & c & c \end{bmatrix} \quad (3)$$

Figure 7(a) shows the result of the inpainting algorithm. As presented in subsection 2.1 the inpainted image can be used for illumination estimation by convolving with a large gaussian kernel. The optimal size of the gaussian filter is dependent upon the scale of the objects in the image and the overall size. Experiments have shown that as a starting value $1/20$ of the image width is recommended.

Figure 7(b) shows the result of the illumination estimation which can be used for compositing a corrected image.

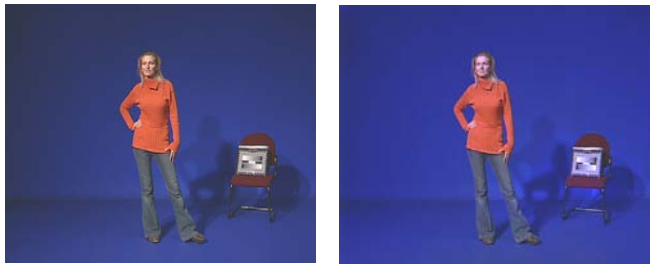
The following equation in pseudo code notation present a possible compositing method for blue screen background, where r,g,b are the image color channels, *illum* is the illumination estimation and k_1 is a factor:

$$\begin{bmatrix} r \\ g \\ b \end{bmatrix} = \begin{bmatrix} max(k_1 * r * illum, r) \\ g \\ b * illum \end{bmatrix} \quad (4)$$



(a) simple chroma key (b) cleanup & matte dilatation

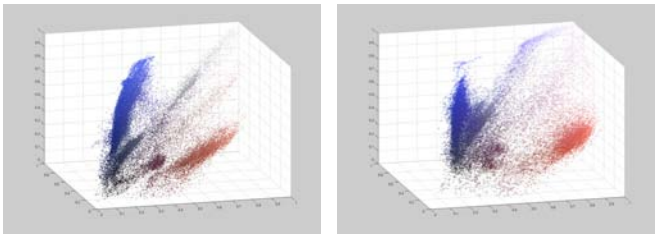
Figure 7: matte pulling



(a) original image (b) illumination correction

Figure 8: image comparison

Noticeable is the change in the color distribution in rgb-space. Foreground and background clusters separate as expected even better after the illumination correction, the deviation of the background pixel distribution is lower.



(a) original image (b) corrected image

Figure 9: color distribution in rgb-space

With a further pass on the GPU easily brightness/contrast or gamma modifications could be done in realtime. The corrected image can now be used for further keying processes while the original image is used for the final composite.

4. RESULTS AND CONCLUSION

The algorithm presented in this abstract implements an realtime capable and easy preprocessing method for improving the keying result for images with uneven or poor lighting in the background. As shown in figure 10 the mattes with

Figure 10: Final Key (Ultimatte^{TM} without cleanup)

a preprocessing illumination correction show interesting results without cleaning up and clipping. The achievable matte can outperform the quality of established realtime solutions and can be additionally used as preprocessing for postproduction.
The sample implementation is done on the GPU with common shaderprograms and needs no readback during computation. Execution times (e.g. NVIDIA GeForce 8800 GTX) are shown in table 3: Obvious is the realtime capability of

algorithm	execution time SD-frame $[ms]$
download image	1.3
subsampling	0.2
chroma key	0.5
dilate	0.5
inpainting	1.0
separable gauss filter	1.5
bilinear upsampling	0.2
compositing	0.2
readback image	1.0

Table 3: algorithm duration of the single steps

this algorithm. Even more advantages are achievable if the corrected image is not readback to the CPU memory. A per-pixel keying algorithm is easy implementable on the GPU with execution times in the range of $1-2ms$. A Further speed improvement can be achieved by combining several building blocks into a single shader-program and thereby increase the arithmetic operations per transferred word (computational/arithmetic intensity).
As a result of an illumination correction the color of foreground objects is changed too and are therefore useless for the final compositing of foreground and background images. For a general keying system a special procedure is proposed (see figure 11. The corrected foreground object is only used for receiving an appropriate matte. For the final composite a preprocessed foreground image with additional color correction is applied. This step yields to more flexibility for adjusting brightness, contrast and color of the individual images. With the illumination correction as preprocessing there is more headroom in the creativity for lighting situations in live productions and therefore the"'poor-lit-bluescreen-problem"' gets a little less problematic.

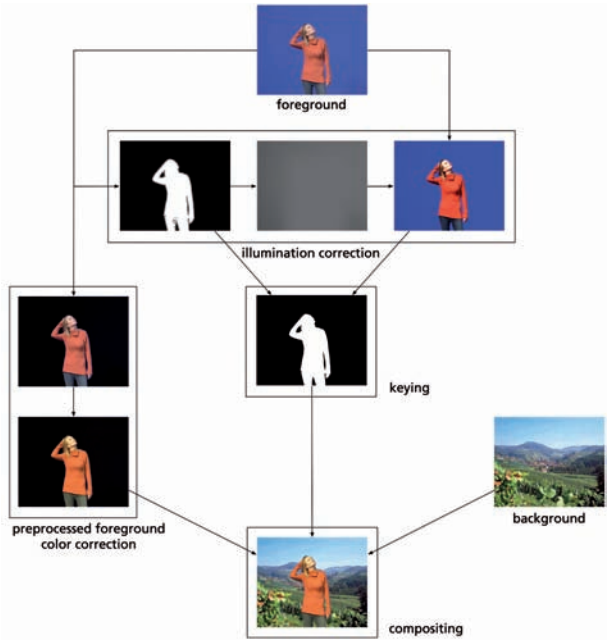


Figure 11: uneven background color

REFERENCES

[1] Steve Wright, *Digital Compositing for Film and Video*, Focal Press, 2002.

[2] Alvy Ray Smith and James F. Blinn, "Blue screen matting," in *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. 1996, pp. 259–268, ACM Press.

[3] GPGPU, "General purpose computation on graphic processing units," http://www.gpgpu.org, 2006.

[4] Lars Hörchens, "Segmentation of video sequences for compositing applications in television production," M.S. thesis, TU Ilmenau, 2005.

[5] John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krger, Aaron E. Lefohn, and Timothy J. Purcell, "A survey of general-purpose computation on graphics hardware," *Computer Graphics Forum*, vol. 26, 2007.

[6] D. J. Jobson and G. A. Woodell, "Properties of a center/surround retinex. Part 2: Surround design," *NASA STI/Recon Technical Report N*, vol. 96, pp. 15986–+, Aug. 1995.

[7] F J W-M Leong, M Brady, and J O'D McGee, "Correction of uneven illumination (vignetting) in digital microscopy images," *J Clin Pathol*, vol. 56, no. 8, pp. 619–621, 2003.

[8] Pat Hanrahan, Jeremy Sugerman, Kayvon Fatahalian, Mike Houston, Tim Floey, Daniel Horn, and Ian Buck, "Gpubench," 2004.

[9] Marcelo Bertalmio, Guillermo Sapiro, Vicent Caselles, and Coloma Ballester, "Image inpainting," in *Siggraph 2000, Computer Graphics Proceedings*, Kurt Akeley, Ed. 2000, Annual Conference Series, pp. 417–424, ACM Press / ACM SIGGRAPH / Addison Wesley Longman.

[10] Manuel M. Oliveira, Brian Bowen, Richard McKenna, and Yu-Sung Chang, "Fast digital image inpainting.," in *VIIP*, M. H. Hamza, Ed. 2001, pp. 261–266, ACTA Press.