Implementing Real-Time Video Abstraction

Sven C. Olsen Holger Winnemöller Bruce Gooch

Northwestern University*

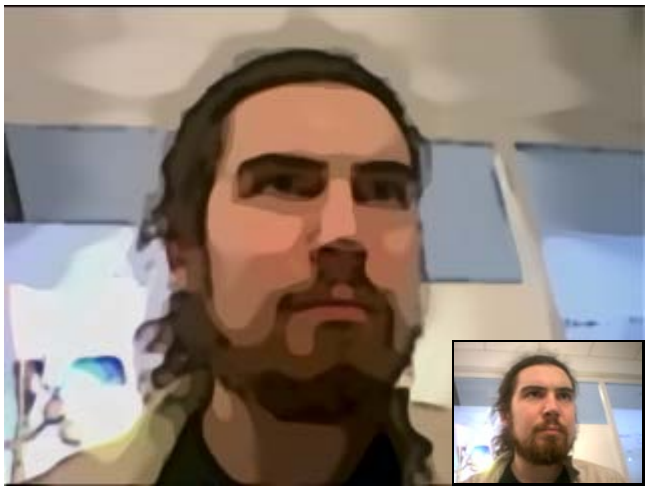

Figure 1: The GPU implementation of our system can be connected to a video capture source (such as a webcam), creating a real-time stylization effect. Demos and project information are available at: http://videoabstraction.net/

1 Introduction

This sketch is a companion to our paper, *Real-Time Video Abstraction*. It includes implementation details and design considerations that are only briefly treated in the core writeup. The main topics to be covered are the performance/quality tradeoffs of the anisotropic diffusion-like effect used in the early phases of the system, and the fragment process optimization techniques used in the real-time GPU implementation.

2 Topics

2.1 Approximating Anisotropic Diffusion

Our system creates an anisotropic diffusion-like effect by iteratively applying a separable approximation to a bilateral filter. This separable approximation was first introduced by Pham and Vliet[2005], but its application in the context of iterative adaptive image smoothing has not been previously studied. There are systematic errors in the bilateral approximation, but we have found that these errors do not cause objectionable artifacts in our final results.

Approximate anisotropic diffusion filters have a long history. The original connection machine program proposed in Perona and Malik [1991] was not a formally correct solution to their own equations, though in practice the errors introduced by their approximations did not introduce negative visual artifacts. We show that a similar truth holds for an iterated approximation to the bilateral filter. We compare performance/quality tradeoffs for true anisotropic diffusion, Perona and Malik's fast, parallel approximation to anisotropic diffusion, true iterated bilateral filtering, and iterated approximations to the bilateral filter.

2.2 Fragment Process Optimization

Our GPU implementation uses NVIDIA's Cg language and compiler. We relate the particular performance studies which motivated our own implementation strategies. First, we observed that when two image processing effects are combined into a single Cg program, a naive concatenation of the two sources often compiles to the same assembly code as a hand-optimized combined file. However, we also found that the fastest possible implementations of Gaussian blurs and approximate bilateral filters require hardcoding large numbers of constants, and performing loop unrolling as a pre-process, before passing the partially optimized code off to Cg.

2.3 Flexible GPU Image Processing Code

Our GPU implementation exploits the ease with which several image processing operations can be combined into a more efficient single pass merged program. All image processing operations are represented internally as annotated snipits of Cg code, and merging distinct operations into a single shader pass is postponed until after the entire image processing effect has been specified. This allows any component operation of the framework to be edited or replaced independent of any other component, even if both operations will be performed inside of the same shader pass. Thus we were able to develop a fragment shader implementation for a complex image processing effect which was both reasonably well optimized, yet also flexible enough to allow for experimentation.

References

PERONA, P., AND MALIK, J. 1991. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence 12*, 7.

PHAM, T. Q., AND VLIET, L. J. V. 2005. Separable bilateral filtering for fast video preprocessing. In *IEEE International Conference on Multimedia & Expo*, CD1–4.

WINNEMÖLLER, H., OLSEN, S. C., AND GOOCH, B. 2006. Real-time video abstraction. In *To Appear in Proceedings of SIGGRAPH 2006*.

*{sven|holger|bgooch}@cs.northwestern.edu