Implementing Real-Time Video Abstraction

Sven Olsen Holger Winnemöller Bruce Gooch

NORTHWESTERN UNIVERSITY

The Project

A real-time stylization filter that produces cartoon-like videos.

- Combines several image-processing algorithms.
- Image-based algorithm that produces temporally coherent results.
- Designed for efficient parallel implementation.

Source

Filtered

This poster focuses on design concerns and implementation details, and does not provide a complete description of the algorithm. For that, please see our paper, *Real-Time Video Abstraction* [3].

A Fast Adaptive Smoothing Filter

At the core of the algorithm is an adaptive smoothing operation that selectively simplifies image content, leading to a cartoon-like look.

Iterated bilateral filtering and anisotropic diffusion are both adaptive smoothers. Adaptive smoothing operations are similar to a mean-shift filter, in that they move pixels values towards local modes. Iterated bilateral filtering in particular may be seen as a restricted mean-shift procedure [1].

However, unlike mean-shift, adaptive smoothing requires only local information, and is thus better suited to implementation on graphics hardware.

We create an adaptive smoothing effect by iteratively applying the separable approximation to a bilateral filter first introduced by Pham and Vliet [2]. The result is very similar to that of a true iterated bilateral filter, but the separability allows for a faster implementation.

True Iterated Bilateral

Fast Approximate Smoother

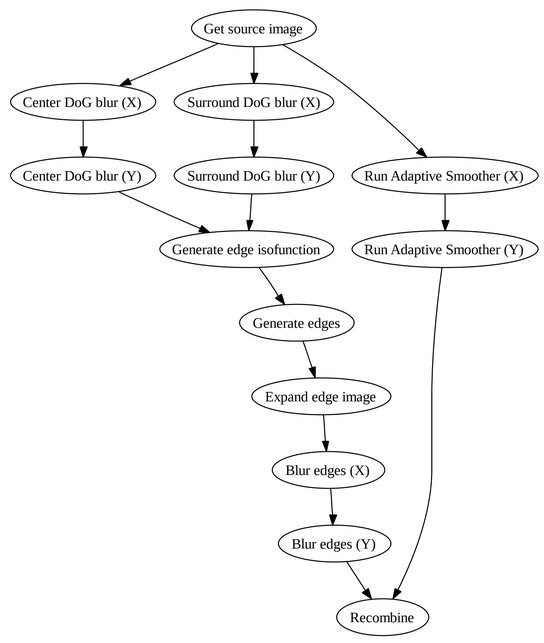
The spacial bias in the approximation leads to minor artifacts around diagonal edges, but otherwise the result is very similar to a true iterated bilateral filter.

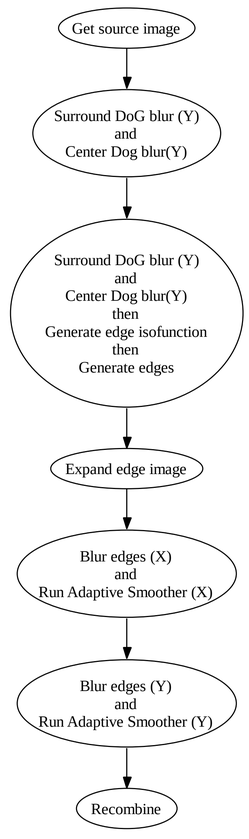
Reducing the Number of Shader Passes

We generate an optimized sequence of GPU fragment programs from a simple description of the filter's component operations.

Our stylization algorithm is composed of more than thirty distinct image processing operations. Each component could be implemented as a single fragment shader pass, but this is relatively inefficient, as several operations can often be combined into a single pass.

Multiple operations may be performed in sequence inside a single shader pass, or two or more operations may be combined by storing the results of each in different color channels. We have built tools that find and perform these optimizations automatically.





The first graph shows several component operations and their interdependencies, before any merging has been applied. The second graph shows the merged operations. Each node denotes a single shader pass. Some merges are the result of stacking memory, in which case, we write 'A and B', while others are implemented sequentially inside the same shader pass, 'A then B'.

Modified Cg Code

Each component operation is written as if it were a single shader program, using a slightly modified version of the Cg language.

```
//restricted memory reads calls replace texRECT et al.
float sharpness = kern(bumps,0,0);
sharpness = clamp(sharpness,0,1)*12+2;

//image names are used in place of texture ids.
float intpart;
float leftover = modf(kern(source)*7,intpart);

//modified return syntax
intpart += .5*tanh(sharpness*(leftover-.5) );
[dest] = (intpart/bins) * 100;
```

The information in the Cg-like code is used to create a directed acyclic graph storing operation dependencies, along with other information used to inform the program merging algorithms.

References

[1] Danny Barash and Dorin Comaniciu. A common framework for nonlinear diffusion, adaptive smoothing, bilateral filtering and mean shift. *Image and Video Computing*, 22(1):73–81, 2004.

[2] Tuan Q. Pham and Lucas J. Van Vliet. Separable bilateral filtering for fast video preprocessing. In *IEEE International Conference on Multimedia & Expo*, pages CD1–4, Amsterdam, July 2005.

[3] Holger Winnemöller, Sven C. Olsen, and Bruce Gooch. Real-time video abstraction. *To Appear in Proceedings of SIGGRAPH 2006*.

{sven|holger|bgooch}@cs.northwestern.edu http://videoabstraction.net/