SDK White Paper

Video Filtering on the GPU

Eric Young
sdkfeedback@nvidia.com

NVIDIA Corporation
2701 San Tomas Expressway
Santa Clara, CA 95050

October 2004

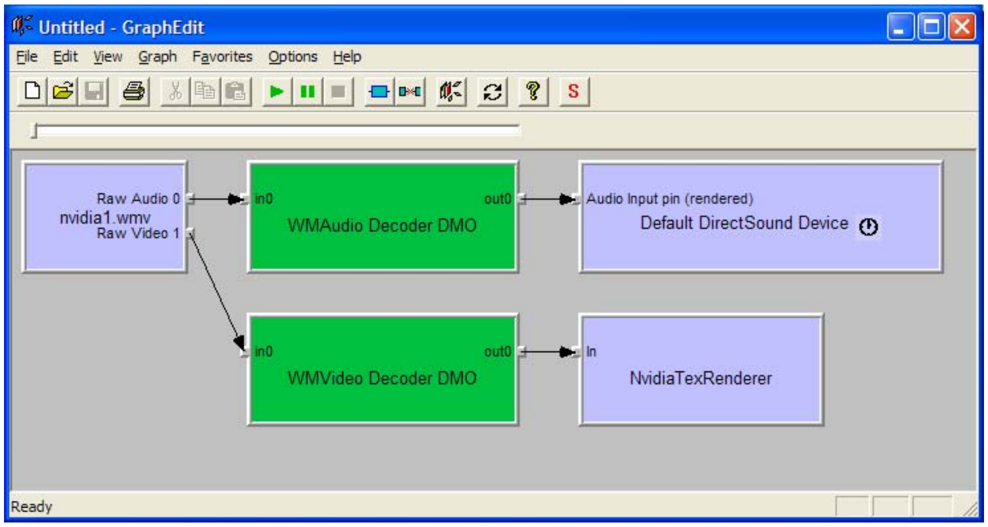
Video Filter Example

Video filtering is performed using image processing techniques. This example demonstrates how to apply compositing and filtering to video using shaders. This technique can be used for a wide range of interesting effects, such as blur, sharpen, luminance edge detection, sepia, radial blur, blending, and wipe.

This document assumes basic knowledge of graphics, Microsoft DirectShow, and shader programming.

This example is implemented using DirectX9.0c and will run on all PS/VS2.0 class hardware.

This example constructs a DirectShow filter graph shown through Microsoft's GraphEdit application. A NvidiaTexRenderer DirectShow filter class is created by the Video Filter example, so video data can be accepted from the decoder. Decoded video frames are copied from the video decoder to a dynamic texture, where video filtering and effects are applied to every video frame. Using pixel shaders, a number of different effects can be performed on the GPU.



Video Filter

Software requirements

This Video Filter example supports many different video formats. Support for DivX®, MPEG-2, and Windows Media™-based content has been tested. Other formats may also be supported if the appropriate DirectShow audio and video decoders are installed. Two WMV9 video files are included with this example. Before running this example, please make sure the WMV9 codec are installed on your system. This is included with Windows Media Player 10 and can be found:

http://www.microsoft.com/windows/windowsmedia/mp10/

The DivX® codec can be found from this website

http://www.divx.com

For MPEG-2 Video files any software decoder will work with this example. For better performance, Nvidia has hardware accelerated DVD decoder filters that can be purchased online.

http://www.nvidia.com/object/dvd_decoder.html

Video Shaders

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.168736 & -0.331264 & 0.5 \\ 0.5 & -0.418688 & -0.081312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{1}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.0 & 0 & 1.402 \\ 1.0 & -0.34413 & -0.714136 \\ 1.0 & 1.772 & 0 \end{bmatrix} \begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} \tag{2}$$

Figure 1. YC_bC_r to RGB color conversion matrix

While color space conversion can be done efficiently on the CPU, this would require video frames to move back and forth between the CPU and GPU. This increases bus traffic, as the GPU needs to wait for each video frame to be uploaded before color space conversion and filtering can be applied. By keeping the data on the graphics board, playback and rendering performance are significantly improved.

Color conversion is applied to every pixel. For RGB to YC_bC_r, Y has a range from [0,+1] and C_b/C_r ranging from [-0.5,+0.5]. To encode RGB to YC_bC_r color space, the matrix (1) needs to be applied. Here we assume that the reference for [black, white] ranges from [0, 1]. For YC_bC_r to RGB conversion, the inverse matrix (2) needs to be applied.

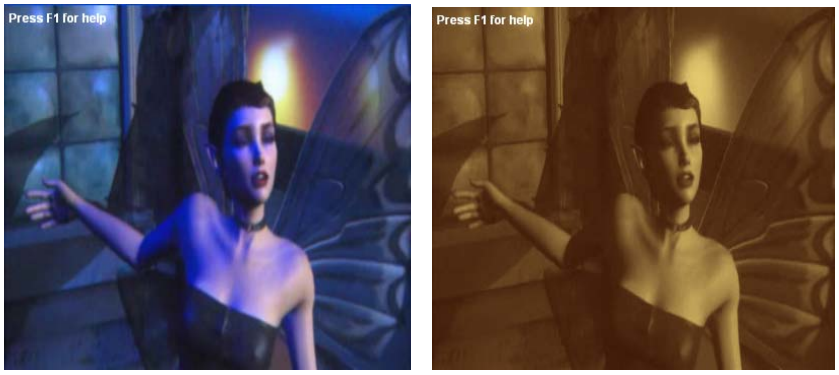


Figure 2. Sepia Filter

In the Figure above, the diagram on the left shows the original video stream. The diagram on the right shows the video stream after the sepia filter is applied.

To apply a Sepia Filter, there are a few steps that are required.

1. We first compute the constant G, which is the dot-product of the transfer matrix and the pixel color.

$$G = \begin{bmatrix} 0.3 & 0.59 & 0.11 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} = (0.3R + 0.59G + 0.11B)$$

2. Next compute the values of colors C_M (muted) and C_S (sepia) with the equation below. C_{ST} (Stain Tone) and C_{PT} (Paper Tone) are both constant colors. D (*desaturation*) is a constant with values that range between [0,1].

$$C_M = \begin{bmatrix} R \\ G \\ B \end{bmatrix} (1 - D) + \begin{bmatrix} G \\ G \\ G \end{bmatrix} (D)$$

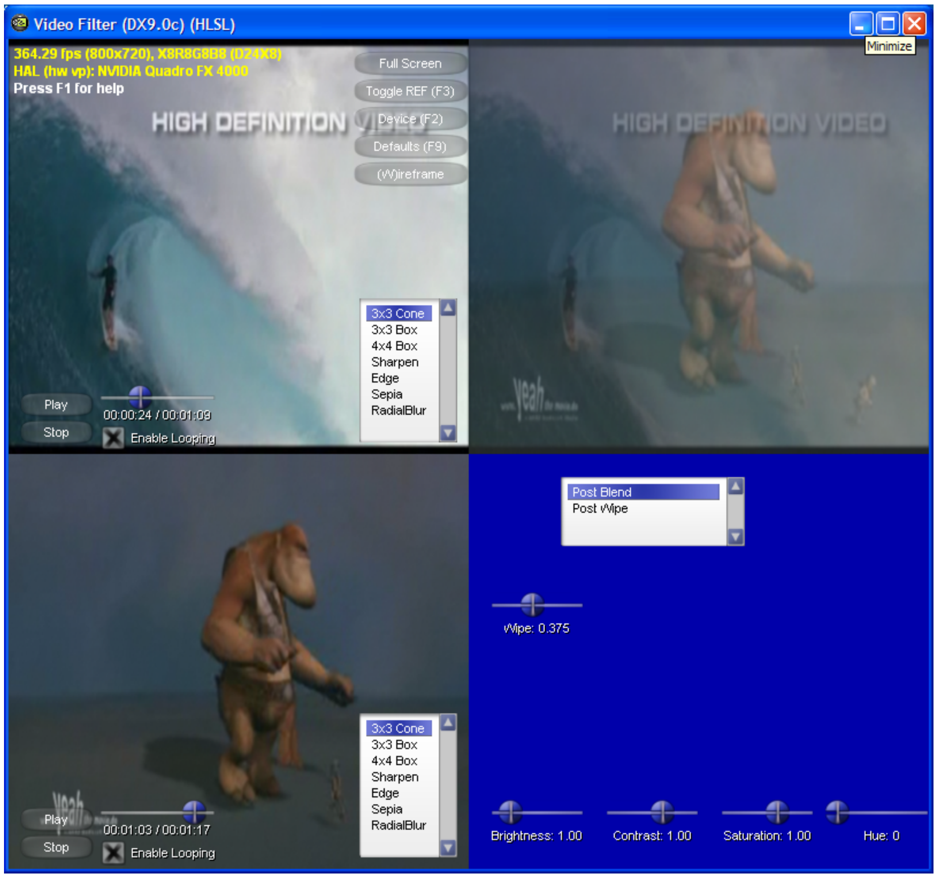
$$C_S = C_{ST} \bullet (1 - G) + C_{PT} \bullet (G)$$

3. To compute the final color C_{FINAL}, we linearly interpolate by using: T (*toning*) which is a constant with values that range between [0, 1].

$$C_{FINAL} = C_S \bullet (1 - T) + C_M \bullet (T)$$

Video Compositing

Final video image may be composed of one or more video streams. There are *Processor Amplifer* slider controls (*Brightness*, *Contrast*, *Saturation*, *Hue*) on the bottom of this example. These settings can be modified interactively for the final rendered video.

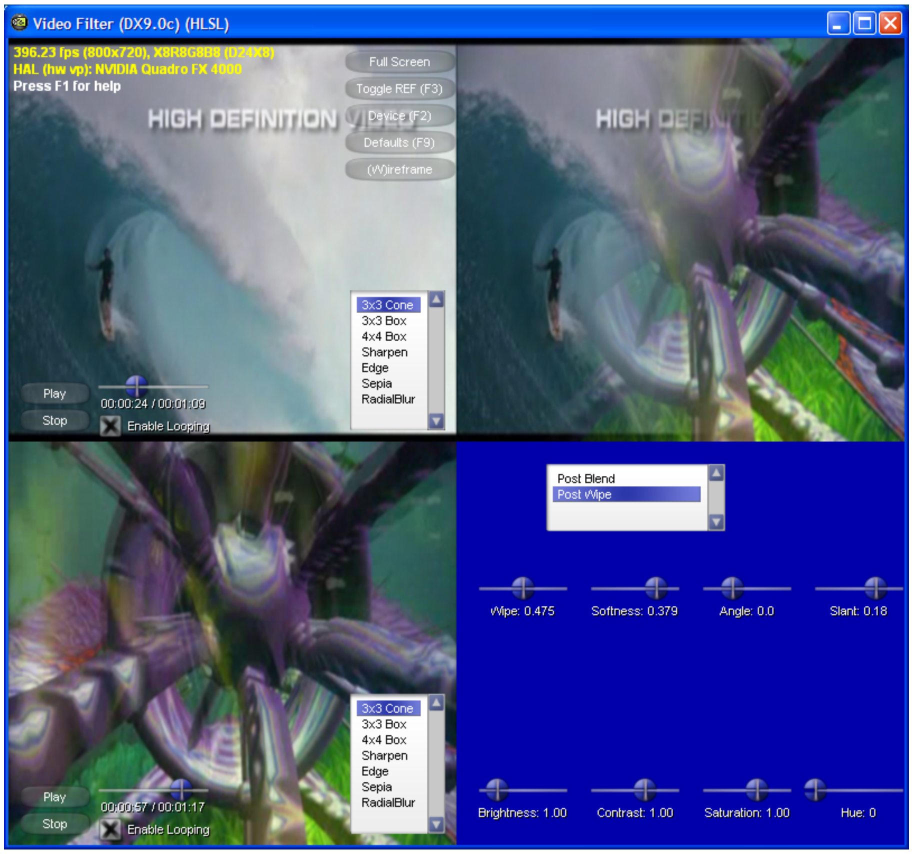


```
364.29 fps (800x720), X8R8G8B8 (D24X8)
HAL (hw vp): NVIDIA Quadro FX 4000
Press F1 for help
00:00:24 / 00:01:09
00:01:03 / 00:01:17
Wipe: 0.375
Brightness: 1.00   Contrast: 1.00   Saturation: 1.00   Hue: 0
```

Figure 3. Post Blend

In the Figure 3, the top and bottom left windows are two video input streams. And the top right window is the result of the blending from the first two streams. Final rendering is simply a linear interpolation between two video frames. Using the *Wipe* slider to control the parameters, the video fades between the first and second video stream.

$$\begin{bmatrix} R_{FINAL} \\ G_{FINAL} \\ B_{FINAL} \end{bmatrix} = \begin{bmatrix} R_{W1} \\ G_{W1} \\ B_{W1} \end{bmatrix} \bullet (1-\alpha) + \begin{bmatrix} R_{W2} \\ G_{W2} \\ B_{W2} \end{bmatrix} \bullet (\alpha)$$



```
396.23 fps (800x720), X8R8G8B8 (D24X8)
HAL (hw vp): NVIDIA Quadro FX 4000
Press F1 for help
00:00:24 / 00:01:09
00:00:57 / 00:01:17
```

Wipe: 0.475 Softness: 0.379 Angle: 0.0 Slant: 0.18

Brightness: 1.00 Contrast: 1.00 Saturation: 1.00 Hue: 0

Figure 4. Post Wipe

In Figure 4, the windows on the top and bottom left are video input streams. The window on the top right shows the final wipe technique. Using the sliders, variables for *Softness* and blending edge *Angle,* and *Wipe* can be adjusted independently to create the appropriate scene transition effect.

NVIDIA Corporation
2701 San Tomas Expressway
Santa Clara, CA 95050
www.nvidia.com