Documentation for developing Lightworks effects (Post # 1 contains summary links)

Posted by schrauber - 19 Apr 2017 09:59

Here we collect information about how to create Lightworks .fx effects.

For a **Basic introduction** , see post #143763 of "jwrl" (the eighth post on this page).

A structured summary of many details posted in this thread,

and more code examples can be found on **GitHub** .

==========

Documentation for developing Lightworks effects

Posted by briandrys - 19 Apr 2017 10:09

I'll create a sticky. Users can then post links and other information here for Lightworks FX development documentation.

==========

Documentation for developing Lightworks effects

Posted by schrauber - 19 Apr 2017 10:23

Ok, I've tried to structure the most important information.

==========

Documentation for developing Lightworks effects

Posted by lghtwrks - 19 Apr 2017 10:54

MS - Reference for HLSL

msdn.microsoft.com/en-us/library/windows/desktop/bb509638 (v=vs.85).aspx

MS - Programming Guide for HLSL

msdn.microsoft.com/en-us/library/windows/desktop/bb509635 (v=vs.85).aspx

+ General Search

social.msdn.microsoft.com/search/en-US/windows?query=hlsl programming

==========

Documentation for developing Lightworks effects
Posted by lghtwrks - 19 Apr 2017 11:01

---

jwrl making his own keyer

www.lwks.com/index.php?option=com_kunena&func=view&catid=7&id=104173&Itemid=81

==========

Documentation for developing Lightworks effects
Posted by lghtwrks - 19 Apr 2017 11:04

The NVIDIA Shader Library - HLSL

with downloads

developer.download.nvidia.com/shaderlibrary/webpages/hlsl_shaders.html

==========

Documentation for developing Lightworks effects
Posted by schrauber - 19 Apr 2017 17:15

Edit:

I have tried to show different fx.- program structures.

The drawing with only one pixel shader can be seen in the post above: 143678 .

Others will follow if needed ...

==========

Documentation for developing Lightworks effects
Posted by jwrl - 19 Apr 2017 21:16

If it helps, here's an edited version of an article that I planned some time back, then abandoned.

How do I create an effect?

There are several resources that can be used to help develop user effects. What got me started are two articles written for Redshark News by khaver. For general information on how Lightworks effects work read Understanding Lightworks VFX , and for an introduction to the Lightworks .fx programming check out How to write video effects for Lightworks . And if you click on the link to my chromakeyer, you'll see how many blunders a newbie can make and still not break the Lightworks effects engine! It's pretty robust.

The first thing to be aware of is that these are not really HLSL effects at all, in that they aren't actually written in that language. They are based on Nvidia's Cg language, and as such, the best place to go for a reference is Nvidia's Cg reference manual . That language is based on a mixture of C and C++, with extra shader-specific commands. If you're familiar with any of the C family you should be able to pick it up reasonably quickly. That's why using HLSL code may not always work for you.

The next thing to do is search out examples on line. Look at Cg, HLSL and GLSL versions. They won't necessarily directly translate into Lightworks effects, but they may give you ideas. Additionally because Lightworks' effects compiler is based on two different libraries there can be compatibility issues cross platform. In the Windows world it's D3D based, but in the Mac/Linux world it's Cg. Repeating, the language used cross-platform is Cg, but the compiler isn't.

As a developer it's up to you to sort cross-platform issues out. In brief, if you ensure that your effect compiles safely under the default Windows ps_2_b profile it should be safely cross-platform. The reverse cannot be guaranteed. To cross-check for yourself, a Linux or Mac user should run Lightworks under Wine or Bootcamp or similar and if the effect compiles you will be safe. Failing that, if you have doubts post it here and ask for feedback. Someone is sure to help.

So without further ado, let's look at some examples, as supplied by Editshare, with additional comments by me.

Single input, Single pass

```
//-----------------------------------------------------------//
// Header
//
```

```
// Lightworks effects have to have a _LwksEffectInfo block
// which defines basic information about the effect (ie. name
// and category). EffectGroup must be &quot;GenericPixelShader&quot;.
//-----------------------------------------------------------//

int _LwksEffectInfo                         // Mandatory header declaration
<
   string EffectGroup = &quot;GenericPixelShader&quot;;  // Mandatory group declaration
   string Description = &quot;Lift, Gamma, Gain&quot;;   // The title displayed
   string Category    = &quot;Colour&quot;;              // Category for the effect
   string SubCategory = &quot;Samples&quot;;             // Optional subcategory
   string Notes       = &quot;Simple example&quot;;      // Optional descriptive text
> = 0;                                      // Mandatory return value.

//-----------------------------------------------------------//
// Inputs
//-----------------------------------------------------------//

// For each 'texture' declared here, Lightworks adds a matching
// input to your effect (so for a four input effect, you'd need
// to declare four textures and samplers).  The order in which
// they're declared is important, because that's the order in
// which they'll appear on the effect.  When the effect is used
// the first texture will be connected to the topmost video layer,
// the second to the next, and so on.

texture Input;

//-----------------------------------------------------------//
// Samplers
//-----------------------------------------------------------//

// For each 'texture' declared above, there must be a matching
// 'sampler' declared.  The sampler is the means by which the
// effect accesses the individual video streams.

sampler FgSampler = sampler_state
{
   Texture = ;
};

//-----------------------------------------------------------//
// Define parameters here.
//
// The Lightworks application will automatically generate
// sliders/controls for all parameters which do not start
// with a leading '_' character
//-----------------------------------------------------------//

float MasterLift
<
   string Group       = &quot;Master&quot;; // Causes this parameter to show in the group 'Master'
```

```
  string Description = &quot;Lift&quot;;   // The text label that will be given on screen
  float MinVal       = -1.00;    // This is a special value, and will display as -100%
  float MaxVal       = 1.00;     // This will display as 100%.
> = 0.0;                        // Default value

float MasterGamma
<
  string Description = &quot;Gamma&quot;;
  string Group       = &quot;Master&quot;;
  float MinVal       = 0.1;      // A value other than 1.0 will display as is, ie., 0.1
  float MaxVal       = 4.00;
> = 1.0;

float MasterGain
<
  string Description = &quot;Gain&quot;;
  string Group       = &quot;Master&quot;;
  float MinVal       = -1.00;
  float MaxVal       = 1.00;
> = 0.0;

float RedLift
<
  string Description = &quot;Lift&quot;;
  string Group       = &quot;Red&quot;;
  float MinVal       = -1.00;
  float MaxVal       = 1.00;
> = 0.0;

float RedGamma
<
  string Description = &quot;Gamma&quot;;
  string Group       = &quot;Red&quot;;
  float MinVal       = 0.1;
  float MaxVal       = 4.00;
> = 1.0;

float RedGain
<
  string Description = &quot;Gain&quot;;
  string Group       = &quot;Red&quot;;
  float MinVal       = -1.00;
  float MaxVal       = 1.00;
> = 0.0;

float GreenLift
<
  string Description = &quot;Lift&quot;;
  string Group       = &quot;Green&quot;;
  float MinVal       = -1.00;
  float MaxVal       = 1.00;
> = 0.0;
```

```
float GreenGamma
<
  string Description = &quot;Gamma&quot;;
  string Group       = &quot;Green&quot;;
  float MinVal       = 0.1;
  float MaxVal       = 4.00;
> = 1.0;

float GreenGain
<
  string Description = &quot;Gain&quot;;
  string Group       = &quot;Green&quot;;
  float MinVal       = -1.00;
  float MaxVal       = 1.00;
> = 0.0;

float BlueLift
<
  string Description = &quot;Lift&quot;;
  string Group       = &quot;Blue&quot;;
  float MinVal       = -1.00;
  float MaxVal       = 1.00;
> = 0.0;

float BlueGamma
<
  string Description = &quot;Gamma&quot;;
  string Group       = &quot;Blue&quot;;
  float MinVal       = 0.1;
  float MaxVal       = 4.00;
> = 1.0;

float BlueGain
<
  string Description = &quot;Gain&quot;;
  string Group       = &quot;Blue&quot;;
  float MinVal       = -1.00;
  float MaxVal       = 1.00;
> = 0.0;

//-----------------------------------------------------------//
// Definitions and declarations
//-----------------------------------------------------------//

// No definitions or declarations are required for this effect - jwrl.

//-----------------------------------------------------------//
// Pixel Shader
//
// This section defines the code which the GPU will execute
// for every pixel in an output image.
//
```

```
// Note that pixels are processed out of order, in parallel.
// When using shader model 2.0, there's a 64 instruction limit.
// Use multple passes if you need more.
//-----------------------------------------------------------//

float4 LiftGammaGain (float2 xy : TEXCOORD1) : COLOR
{
   // Read a pixel from the source image at position 'xy'
   // and place it in the variable 'source' - jwrl.

   float4 source = tex2D (FgSampler, xy);

   // Set up compound RGBA variables from the user-specified
   // channel parameters - jwrl.

   float4 gamma = { RedGamma, GreenGamma, BlueGamma , 1.0 };
   float4 lift  = { RedLift,  GreenLift,  BlueLift,   1.0 };
   float4 gain  = { RedGain,  GreenGain,  BlueGain,   1.0 };

   // Note the use of 1.0.xxx in the code below.  This is a
   // shorthand way of getting the compiler to treat that value
   // as equivalent to float3 (1.0, 1.0, 1.0).  The technique
   // is known as swizzling.  In this case the .xxx suffix is
   // the swizzle operator - jwrl.

   gamma.rgb += MasterGamma - 1.0.xxx;
   lift.rgb  += MasterLift;
   gain.rgb  += MasterGain + 1.0.xxx;

   // Clamp the values.  Here a simple method would be to use the
   // max() function.  Writing gamma.r = max (gamma.r, 0.1) will
   // do exactly what we need, which is to limit the minimum value
   // while leaving the upper limit open ended.  Using either
   // clamp() or saturate() would work, but would limit both ends
   // of the range.

   // Instead of processing each channel separately we could also
   // use gamma = max (gamma, 0.1).  This would work, but would
   // apply the same limits to gamma.a.  That isn't always what
   // you would want.  In this case it would be safe to do since
   // gamma.a = 1.0 - jwrl.

   gamma.r = max (gamma.r, 0.1);
   gamma.g = max (gamma.g, 0.1);
   gamma.g = max (gamma.b, 0.1);

   // Perform the correction.  We swizzle again...

   float4 corrected = pow (source, 1.0 / gamma) * gain;
   corrected += (1.0.xxxx - corrected) * lift;

   // Honour source alpha
```

```
   corrected = lerp (source, corrected, source.a);
   corrected.a = source.a;

   return corrected;
}

//-----------------------------------------------------------//
// Technique
//
// Specifies the order of passes (we only have a single pass
// with this example, so there's not much to do)
//-----------------------------------------------------------//

technique SampleFxTechnique
{
   pass SinglePass
   {
      PixelShader = compile PROFILE LiftGammaGain();
   }
}
```

Single input, Multi pass

```
//-----------------------------------------------------------//
// Header
/-----------------------------------------------------------//

int _LwksEffectInfo
<
   string EffectGroup = &amp;quot;GenericPixelShader&amp;quot;;
   string Description = &amp;quot;Shear&amp;quot;;
   string Category    = &amp;quot;Stylize&amp;quot;;
   string SubCategory = &amp;quot;Samples&amp;quot;;
   string Notes       = &amp;quot;Two pass example&amp;quot;;
> = 0;

//-----------------------------------------------------------//
// Inputs
//-----------------------------------------------------------//

texture Input;

// This is a TWO pass effect, so we need an intermediate texture
// which stores the results from pass 1, and serves as the input
// to pass 2.  This is done by defining a target texture in this
// way.  Because it's a target texture only, it will not appear
// as a user input to the effect - jwrl.
```

```
texture OutputFromPassOne : RenderColorTarget;

//-----------------------------------------------------------//
// Samplers
//-----------------------------------------------------------//

sampler FgSampler = sampler_state
{
  Texture = ;
};

sampler P1OutSampler = sampler_state
{
  Texture = ;
  AddressU  = ClampToEdge;        // These have been included to show the settings
  AddressV  = ClampToEdge;        // available when declaring a sampler.  ClampToEdge
  MinFilter = Linear;             // addressing is shown instead of Clamp because of
  MagFilter = Linear;             // a cross-platform difference in Clamp addressing.
  MipFilter = Linear;             // These could be omitted in this effect - jwrl.
};

// Additional note:  There are differences in cross-platform default behaviour.  While
// debugging effects it was found that the sampler default state could not always be
// relied on.  For that reason it is safer to fully declare samplers, or at the very
// least, fully declare the address modes - jwrl.

//-----------------------------------------------------------//
// Parameters
//-----------------------------------------------------------//

float Horizontal
<
  string Description = &quot;Horizontal&quot;;
  float MinVal       = -1.00;
  float MaxVal       = 1.00;
> = 0.0;

float Vertical
<
  string Description = &quot;Vertical&quot;;
  float MinVal       = -1.00;
  float MaxVal       = 1.00;
> = 0.0;

//-----------------------------------------------------------//
// Shaders
//-----------------------------------------------------------//

float4 ShearHorizontal (float2 xy : TEXCOORD1) : COLOR
{
  // Amend the x position based on the user-specified shear
  // amount
```

```
  float2 pos = xy;
  pos.x += (pos.y - 0.5) * Horizontal;

  return tex2D (FgSampler, pos);
}

float4 ShearVertical (float2 xy : TEXCOORD1) : COLOR
{
  // Amend the y position based on the user-specified shear
  // amount

  float2 pos = xy;
  pos.y += (pos.x - 0.5) * Vertical;

  return tex2D (P1OutSampler, pos);
}

//-----------------------------------------------------------//
// Techniques
//
// Now that we have more than one pass, this specifies the
// order in which the passes are executed.
//-----------------------------------------------------------//

technique SampleFxTechnique
{
  pass ShearX
  <
    string Script = &quot;RenderColorTarget0 = OutputFromPassOne;&quot;;  // Sets the target the
  >                                                  // result is placed in
  {
    PixelShader = compile PROFILE ShearHorizontal ();
  }

  pass ShearY
  {
    PixelShader = compile PROFILE ShearVertical ();
  }
}
```

Multi input, Single pass

```
//-----------------------------------------------------------//
// Header
//-----------------------------------------------------------//

int _LwksEffectInfo
<
```

```
   string EffectGroup = &quot;GenericPixelShader&quot;;
   string Description = &quot;Channel Combiner&quot;;
   string Category    = &quot;Stylize&quot;;
   string SubCategory = &quot;Samples&quot;;
   string Notes       = &quot;Multi input, single pass example&quot;;
> = 0;

//-----------------------------------------------------------//
// Inputs
//-----------------------------------------------------------//

texture R;
texture G;
texture B;
texture A;

//-----------------------------------------------------------//
// Samplers
//-----------------------------------------------------------//

sampler RSampler = sampler_state { Texture = ; };
sampler GSampler = sampler_state { Texture = ; };
sampler BSampler = sampler_state { Texture = ; };
sampler ASampler = sampler_state { Texture = ; };

//-----------------------------------------------------------//
// Parameters
//-----------------------------------------------------------//

float Opacity
<
   string Description = &quot;Opacity&quot;;
   float MinVal       = 0.0;
   float MaxVal       = 1.0;
> = 1.0;

//-----------------------------------------------------------//
// Shaders
//-----------------------------------------------------------//

float4 CombineRGBA (float2 xy : TEXCOORD1) : COLOR
{
   return float4 (tex2D (RSampler, xy).r,
                  tex2D (GSampler, xy).g,
                  tex2D (BSampler, xy).b,
                  tex2D (ASampler, xy).a * Opacity );
}

//-----------------------------------------------------------//
// Technique
//-----------------------------------------------------------//
```

```
technique SampleFxTechnique
{
  pass SinglePass
  {
    PixelShader = compile PROFILE CombineRGBA ();
  }
}
```

Single input, Multi technique

```
//-----------------------------------------------------------//
// Header
//-----------------------------------------------------------//

int _LwksEffectInfo
<
  string EffectGroup = &amp;quot;GenericPixelShader&amp;quot;;
  string Description = &amp;quot;Channel Extractor&amp;quot;;
  string Category    = &amp;quot;Stylize&amp;quot;;
  string SubCategory = &amp;quot;Samples&amp;quot;;
  string Notes       = &amp;quot;Single input, multi technique example&amp;quot;;
> = 0;

//-----------------------------------------------------------//
// Inputs
//-----------------------------------------------------------//

texture Input;

//-----------------------------------------------------------//
// Samplers
//-----------------------------------------------------------//

sampler FgSampler = sampler_state
{
  Texture = ;
};

//-----------------------------------------------------------//
// Parameters
//-----------------------------------------------------------//

// If Lightworks encounters a parameter called 'SetTechnique',
// it will be shown on the FX panel like any other parameter,
// but it will also implicitly choose an execution technique
// to match the user-selection (eg. if the user chooses 'Green'
// from the drop-down list, Lightworks will arrange for the
// 'Green' technique (at the bottom of this file) to execute
```

```
// when the effect is rendered).

int SetTechnique
<
   string Description = &quot;Channel&quot;;
   string Enum = &quot;Red,Green,Blue,Alpha&quot;; // The Enum parameter allows you to use literal
> = 0;                              // comma-delimited text in your selection menu

//-----------------------------------------------------------//
// Shaders
//-----------------------------------------------------------//

float4 ExtractRed (float2 xy : TEXCOORD1) : COLOR
{
   float4 pixel = tex2D (FgSampler, xy);

   // This next is the original Editshare technique used in
   // their example to assign the red channel to all of RGB.
   // There are other ways possible - see ExtractGreen() and
   // ExtractBlue() below this - jwrl.

   return float4 (pixel.r, pixel.r, pixel.r, 1.0);
}

float4 ExtractGreen (float2 xy : TEXCOORD1) : COLOR
{
   float4 pixel = tex2D (FgSampler, xy);

   // This changes the original Editshare technique so that
   // we directly assign the green channel to all of RGB in
   // one hit.  ExtractBlue() below this shows another - jwrl.

   return float4 (pixel.ggg, 1.0);
}

float4 ExtractBlue (float2 xy : TEXCOORD1) : COLOR
{
   // Even more efficiently, it's possible to do everything
   // at once - jwrl.

   return float4 (tex2D (FgSampler, xy).bbb, 1.0);
}

float4 ExtractAlpha (float2 xy : TEXCOORD1) : COLOR
{
   float4 pixel = tex2D (FgSampler, xy);

   return float4 (pixel.a, pixel.a, pixel.a, 1.0);
}

//-----------------------------------------------------------//
// Techniques
```

```
//-----------------------------------------------------------//

// The technique names used here are not important: their order
// is.  If you were to move the &quot;Green&quot; technique to the head
// of this list it would execute when the user selected &quot;Red&quot;,
// and vice versa.

technique Red
{
  pass SinglePass
  {
    PixelShader = compile PROFILE ExtractRed ();
  }
}

technique Green
{
  pass SinglePass
  {
    PixelShader = compile PROFILE ExtractGreen ();
  }
}

technique Blue
{
  pass SinglePass
  {
    PixelShader = compile PROFILE ExtractBlue ();
  }
}

technique Alpha
{
  pass SinglePass
  {
    PixelShader = compile PROFILE ExtractAlpha ();
  }
}
```

If you want to examine these shader examples in detail, highlight them in the code block and copy them into a plain text editor. I repeat: it **must** be a plain text editor. Word or other Office style editors will not do. Additionally, the file must be saved in the form MyEffect **.fx**.
MyEffect.fx
.txt
or MyEffect.fx
.doc
won't work. Alternatively you can download them from
SampleFX.zip
.

==========

Documentation for developing Lightworks effects

Posted by jwrl - 20 Apr 2017 01:14

_____

I've now tracked down a series of answers to user questions posted by developers Hammerhead and Great White. Where necessary to make the answer clearer I have added a question before it. I've also occasionally added additional information in italics after the answer from Hammerhead.

_____

I cleaned up my effects folder and all my effects went black. Why?

Hammerhead wrote:

Once you've created a new effect tempate from the FX file, *the FX file is still in use* (and will be compiled each time Lightworks starts) so if the file is removed or modified (such that it no longer compiles) then any applied effects will no longer work (you'll get black).

_____

Sometimes my exported effect doesn't match what I set up while I was editing. Why is this?

Hammerhead wrote:

Don't forget that **all** images are rendered at the project output resolution and *then* sized for preview or export.

_____

Are the Lightworks specialisations (header, runtime variables, etc) documented anywhere?

Hammerhead wrote:

I haven't written a document yet, but the sample .FX files in 'LightworksEffect Templates' are commented fairly extensively

...Lightworks supplies the following runtime variables :

_Progress

= (currentFrame - segmentStartFrame) / segmentLength

_OutputWidth

The width of the current output format in pixels

_OutputHeight

The height of the current output format in pixels

_OutputAspectRatio

The aspect-ratio of the current output format

Additional information: *there is a problem with the way that Lightworks returns _OutputHeight that means on interlaced projects it returns half the expected value when playback is stopped. To avoid this I use the following definition.*

#define Output_Height (_OutputWidth/_OutputAspectRatio)

I then use Output_Height instead of _OutputHeight in my code. If you don't do something like that you can get unexpected results.

==========

Documentation for developing Lightworks effects

Posted by jwrl - 20 Apr 2017 01:50

_____

OK, here are more of Hammerhead's questions and answers.

_____

How can I make a file picker like the Image Matte effects?

Hammerhead wrote:

Unfortunately there's no way of creating/binding a file-picker from a shader (the image matte effect is not implemented using a pixel-shader).

...Bear in mind that some effects have hand-coded panels (rather than shader-generated ones) so it may not be possible to replicate some of the UI elements that are present in some of the native effects.

_____

How do I get the mouse movable position tool on the Viewer like the Image Matte effects?

Hammerhead wrote:

You can make any position parameter editable on the viewer by adding a couple of extra annotations to your x/y params, namely "SpecifiesPointX" and "SpecifiesPointY"; Here's how I do it in the radial blur effect :

```
float CentreX
<
  string Description = &amp;quot;Origin&amp;quot;;
  string Flags = &amp;quot;SpecifiesPointX&amp;quot;;
  float MinVal = 0.00;
  float MaxVal = 1.00;
> = 0.5;

float CentreY
<
  string Description = &amp;quot;Origin&amp;quot;;
  string Flags = &amp;quot;SpecifiesPointY&amp;quot;;
  float MinVal = 0.00;
  float MaxVal = 1.00;
> = 0.5;
```

Additional information: *when using "SpecifiesPointY" it may be necessary to invert the sense of the value returned. In the example above* float Centre_Y = 1.0 - CentreY *will do that. You then use Centre_Y in your code instead of CentreY.*

There's also a third parameter in this group, "SpecifiesPointZ". To see that being used look at the position parameters in the 3D DVE effect. _____

_

How do I get the colour picker and not just a slider?

Hammerhead wrote:

I've attached a copy of the pixel shader that we use for generating colour gradients. It demonstrates how to declare/use colour-based parameters

```
float4 TLColour
<
  string Description = &quot;Top Left&quot;;
  bool SupportsAlpha = true;
> = { 1.0, 0.0, 0.0, 1.0 };
```

Additional information: *You can use either standard brackets in that parameter to give you (1.0, 0.0, 0.0, 1.0) or curly brackets as shown above. If you use the standard brackets the colour shown will default to white, regardless of any values that you may have set in your code.*

I need to make a colour wheel.

Hammerhead wrote:

You can make colour params be represented using a colour wheel by adding the annotation "SpecifiesColourOffset"; For example :

```
float4 MidTintColour
<
  string Description = &quot;Midtones&quot;;
  string Group       = &quot;Tonal Ranges&quot;;
  string Flags       = &quot;SpecifiesColourOffset&quot;;
> = { 1.0, 1.0, 1.0, 1.0 };
```

Additional information: *the colour wheel doesn't return simple RGB values when you use that switch. The brightness setting ranges from 50% to 159%, but doesn't seem to permit ever reaching white.*

How do I turn keyframes on automatically?

Hammerhead wrote:

You can make float-based parameters implicitly keyframed by adding two more annotations - KF0 and KF1 - like this :

```
float Amount
<
```

```
   string Description = &amp;quot;Amount&amp;quot;;
   float MinVal = 0.0;
   float MaxVal = 1.0;
   float KF0    = 0.0;
   float KF1    = 1.0;
> = 0.5;
```

==========

Documentation for developing Lightworks effects

Posted by jwrl - 20 Apr 2017 20:08

_____

Now we have some answers from Great White, to do with HLSL/GLSL issues.

_____

How are Lightworks shaders implemented in the Linux or Mac versions since they are written in HLSL which is for DirectX? Does Lightworks translate the .fx files to an OpenGL shader before they are compiled?

Great White wrote:

We use nVidia's cg library (which is cross-platform and cross-vendor) - it allows us to compile the .fx files 'as is' so that we don't need to have different OpenGL versions for Linux/Mac. There are some slight incompatibility issues between the cg compiler and the dx compiler (eg. declaring 'const' arrays) but nothing that you can't easily work around.

_____

Is there anything that I need to know when working with the Linux/Mac version of the compiler?

Great White wrote:

For reference, the shader compiler on Linux :

- Does not like variables that are declared as 'const'
- Prefers 'fmod' instead of '%'

Additional information: *the reference to Linux in Great White's answer can also be taken to include OS-X.*

==========

Documentation for developing Lightworks effects

Posted by jwrl - 21 Apr 2017 00:07

_____

Hammerhead wrote:

You can make colour params be represented using a colour wheel by adding the annotation "SpecifiesColourOffset"

If you use the "SpecifiesColourOffset" flag the result that you will see in your settings menu will look like this.

How it's possible to sensibly make use of it I've yet to work out. As you drag to the centre it tends to go black, or at the very best, mid grey. I think that for user effects the standard colour wheel without that flag is better.

There is a second switch, "SpecifiesColourRange", that you can use. This gives a drawbar style layout and looks like this.

Because this is a more specialised tool intended mainly for use in keying I will leave it up to the interested user to follow up. You will find an example of its use in the Editshare chromakey sample code.

==========

Documentation for developing Lightworks effects

Posted by brotenet - 23 Apr 2017 05:41

Wow!!

I seriously did not expect such great response!!

Thanks jwrl !!!

==========

Documentation for developing Lightworks effects

Posted by jwrl - 23 Apr 2017 18:40

Glad to be able to help. And I have more.

If you go to the "Shaders" folder in your Lightworks program folder you will find a lot of Editshare-supplied examples. They all use "compile **PROFILE** MyShader ()" as the compiler directive. In Windows this currently defaults to the
ps_2_b
profile. In OS-X and Linux it uses whatever profile the operating system will support.

If you need a specific shader profile to get your effect working on Windows you will need to explicitly declare that profile. In LW 14.0 and earlier the safest way to do this was to add a definition to your code, then supply two versions, one for Windows, the other for Linus/OS-X. This is an example of what I mean, using ps 3.0.

```
#define PROFILE ps_3_0   // Allows the effect to compile in Windows

// #define PROFILE ps_3_0   Allows the effect to compile in Linux/OS-X
```

If you put that declaration at the very head of your code and simply comment it out for the non-Windows versions it will compile, but you **must** supply both versions in zip file form. My preferred method was to put each in separate folders inside the one zip. That way the effect file could have the same name cross platform, thus reducing the risk of user confusion when the effects are referred to.

In Lightworks version 14.5 and higher there is a simpler technique possible which means that you don't need to maintain two versions. Simply add the following three lines of code.

```
#ifdef WINDOWS  // This flag is only available in LW versions 14.5 and up
#define PROFILE ps_3_0
#endif
```

That will ensure the Lightworks will compile the effect on all three platforms that it supports, without any further action on your part.

Also please give some thought to the titles that you give effects. Something descriptive is good, but not if the name becomes something like "Classic analog TV with intermittent loss of vertical lock and ghosting". "Old TV set" would have to be better. The first would probably not get much further than "Classic analog TV with intermi" when displayed anyway, although I don't know the character limit with effect names. I've never been motivated to test it.

When naming the effect file keep it short - that would be better in OldTV.fx than in my My_custom_effect_to_simulate_early_TV.fx. **Always** attach the effect files to your post zipped, and in your forum post include a description of what it actually does along with a screen grab showing before and after versions. If you feel that it is necessary you can include a grab of the user interface too. I sometimes do, sometimes don't.

Because the forum software does an abysmal job of scaling images I always scale my screen grabs in Photoshop to 480x270 from 1920x1080 originals and save them as PNG files. In my case they are usually genuine screen grabs, but from time to time I export a frame from Lightworks and use that. Either way works, but for me the screen grab method is faster because I have an action in Photoshop that copies the contents of the clipboard into a new image then scales it to 480x270. I just do a Ctrl-Print Screen to get it to the clipboard and and thus don't have to explicitly open a file. I show the before and after as a split image, usually with a divide between them.

If you supply the grabs in that 480 form I will take care of getting them into the user effects forum.

==========

Documentation for developing Lightworks effects

Posted by jwrl - 23 Apr 2017 23:31

There's one other thing that I do. I have saved a template file for creating effects. It's undergone some modification since I started doing this, but here's the current version.

```
// @Maintainer [Optional]
// @Released [Required]
// @Author [Required - repeat if more than one author]
// @Created [Required]
// @see [PNG or JPEG screen grab - repeat for more images or MP4 examples of use]

/**
 PLACE DESCRIPTION HERE USING AS MANY LINES AS NECESSARY
*/

//-----------------------------------------------------------------------------//
// Lightworks user effect MyEffect.fx
//
// PLACE REVISION HISTORY HERE
//-----------------------------------------------------------------------------//

int _LwksEffectInfo
<
   string EffectGroup = &quot;GenericPixelShader&quot;;
   string Description = &quot;...&quot;;
   string Category    = &quot;...&quot;;
   string SubCategory = &quot;...&quot;;
   string Notes       = &quot;...&quot;;
> = 0;

//-----------------------------------------------------------------------------//
// Inputs
//-----------------------------------------------------------------------------//

...

//-----------------------------------------------------------------------------//
// Samplers
//-----------------------------------------------------------------------------//

...

//-----------------------------------------------------------------------------//
// Parameters
//-----------------------------------------------------------------------------//

...

//-----------------------------------------------------------------------------//
// Definitions and declarations
//-----------------------------------------------------------------------------//

...

//-----------------------------------------------------------------------------//
// Functions
//-----------------------------------------------------------------------------//
```

```
float2 fn_some_func (...)
{

...

}

//-----------------------------------------------------------------------------------------//
// Shaders
//-----------------------------------------------------------------------------------------//

float4 ps_main (...) : COLOR
{

...

}

//-----------------------------------------------------------------------------------------//
// Techniques
//-----------------------------------------------------------------------------------------//

technique MyEffect
{
  pass pass_one
  {
    PixelShader = compile PROFILE ps_main ();
  }
}
```

I'm not suggesting for one moment that this should be slavishly copied by anyone writing FX files. It's just what I do.

I also prefix any functions that I use with fn_, and shaders with ps_ so that it's immediately clear what's going on. If I decide to create a variant of fmod() that always returns a positive result there will be no confusion if my function is called fn_fmod(). That's actually a dumb example, but I suspect that you will get what I mean.

On the subject of functions, it will always be more efficient if you can manage to use as few as possible. The overhead of a function call is actually higher than implementing it as a separate compile pass, or better still, inline code.

Also conditional statements aren't particularly efficient. They will always fully evaluate the condition, then discard the result they don't need. As well, if possible I will usually try and structure conditionals so that I can force an exit as the sole result of the true condition. In normal programming this is definitely

not the best practice but here it can improve efficiency, since it means that there is no else condition. That may not be too important with a simple if/else statement, but can matter a great deal if you have if/else if/else if/else constructs.

Finally loops will always be unrolled at compile time at least in the Windows compiler. I'm unsure about that for the other two. This has implications for just how much you can build into a loop. It also means that you can't define the exit condition as a variable.

Of the two fragments in the code block below the first will work, the second won't necessarily.

```
for (int i = 0; i < 20; i++) {

  // this version works

  }

for (int i = 0; i < Num; i++) {

  // this version won't work in Windows, will in Mac/Linux

  }
```

There is a workaround which is very bad programming practice. Cg will not let you execute a break from within a loop, but you can force a return. A conditional evaluation of the end state within the loop will work, but is a workaround at best, and definitely should be avoided. For interest only it would look like this.

```
for (int i = 0; i < 20; i++) {

  ...

  if (i >= Num) {
  // optionally do something here
  i = 20;
  }

  ...

}
```

==========