Luma Preserving Mapper (LPM)

What is LPM?

Luma Preserving Mapper (LPM hereafter) is a tone mapping and gamut mapping solution for HDR and wide gamut content. LPM tone maps the luma of the RGB pixel instead of the colors itself, but it makes sure that tonemap(luma(RGB)) would be very similar to luma(tonemap(RGB)), that is, it preserves the luma of the pixel.

Why use LPM?

In short, the benefits of LPM are:

- Can be configured to work with any positive RGB input working color space
- Can be configured to target any RGB output color space (target any display)
- Solution is completely ALU based without any LUT, so has the capacity to run asynchronously with workloads which are bottlenecked by other hardware limits.
- Supports packed FP16 for better performance on Vulkan.

How LPM works?

LPM is split into two parts: a setup call and filter call. The setup call writes pertinent data to a fixed size control block with regards to what the tone and gamut mapping calculations need and the filter call reads from the control block and calculates and outputs a tone and gamut mapped color value or pair of values for the FP16 version.

We will first go over the setup.

```
A_STATIC void LpmSetup(
// Path control.
AP1 shoulder, // Use optional extra shoulderContrast tuning (set to false if shoulderContrast is 1.0).
// Prefab start, "LPM_CONFIG_".
AP1 con, // Use first RGB conversion matrix, if 'soft' then 'con' must be true also.
AP1 soft, // Use soft gamut mapping.
AP1 con2, // Use last RGB conversion matrix.
AP1 clip, // Use clipping in last conversion matrix.
AP1 scaleOnly, // Scale only for last conversion matrix (used for 709 HDR to scRGB).
// Gamut control, "LPM_COLORS_".
inAF2 xyRedW,inAF2 xyGreenW,inAF2 xyBlueW,inAF2 xyWhiteW, // Chroma coordinates for working color space.
inAF2 xyRedO,inAF2 xyGreenO,inAF2 xyBlueO,inAF2 xyWhiteO, // For the output color space.
inAF2 xyRedC,inAF2 xyGreenC,inAF2 xyBlueC,inAF2 xyWhiteC,AF1 scaleC, // For the output container color
space (if con2).
// Prefab end.
AF1 softGap, // Range of 0 to a little over zero, controls how much feather region in out-of-gamut
mapping, 0=clip.
// Tonemapping control.
AF1 hdrMax, // Maximum input value.
AF1 exposure, // Number of stops between 'hdrMax' and 18% mid-level on input.
AF1 contrast, // Input range {0.0 (no extra contrast) to 1.0 (maximum contrast)}.
AF1 shoulderContrast, // Shoulder shaping, 1.0 = no change (fast path).
inAF3 saturation, // A per channel adjustment, use <0 decrease, 0=no change, >0 increase.
inAF3 crosstalk) // One channel must be 1.0, the rest can be <= 1.0 but not zero.
```

Let’s go over every parameter:

- shoulder: This boolean option is to control the contrast in the shoulder section of the tone mapper. Set to false if shoulderContrast is 1.0f.
- con: Option to turn gamut mapping on when content or input gamut is wider than output gamut
- soft: Option to use perceptual soft-falloff gamut mapper. If this is true, con must be true
- con2: This option is used when the display mode requires the final output to be in certain color space for example HDR10_ST2084 mode requires final output to be in Rec2020 or FS2_DisplayNative requires final output to be in display specific P3 primaries
- clip: This option is used to disable gamut mapping for the last con2 dependent display mode conversion matrix.
- scaleOnly: This option is used when final output needs to be scaled for certain display modes. For example, for FS2_scRGB output needs to be scaled by MaxDisplayNits / 80.
- xyRedW, xyGreenW, xyBlueW, xyWhiteW: Working/content/input gamut primaries.
- xyRedO, xyGreenO, xyBlueO, xyWhiteO: Output gamut primaries
- xyRedC, xyGreenC, xyBlueC, xyWhiteC: Container gamut primaries for display mode specific conversion when con2 is on.
- scaleC: scale factor when final output needs to be scaled based on display mode. scaleOnly needs to be turned on.
- softGap: Tiny fractional value to decide how much space you want to dedicate for out of gamut values in your output gamut. Soft needs to be true.
- hdrMax: The maximum brightness value of your game's scene in linear space. This is not any units like nits. Simply the max RGB value of the scene.
- exposure: The number of stops in powers of 2 from 18% of hdrMax to hdrMax.
- contrast: This value gets applied to luma of the pixel during tone mapping as follows: luma = luma ^ contrast.
- shoulderContrast: Extra multiplying factor plugged into contrast to decide how much contrast will be in the shoulder section of the tone mapped output. 1.0f means no change ie. as good as shoulder set to false.
- saturation: This value gets applied to pixel ratio as follows: colourRatioN = colourRatioN ^ saturation where N is {R, G, B}.
- crosstalk: This three-float value decides how the input colors when overexposed or go beyond SDR range, should shape into the color gamut. When all values are 1.0, RGB channels on overexposure will lose their colors and go to white. Adjust the ratio based on how you want to preserve the primaries on over exposure. One channel must be 1.0, rest need to be 0.0 < x <= 1.0 but not 0.0. This is important, more on this later.

The LPM header file has preset prefabs following the convention "LPM_CONFIG_" for setting booleans for con, soft, con2, clip and scaleOnly based on what input content gamut is being used, what output gamut you want to target and which HDR mode you want to target. There are also preset prefabs for common color spaces like Rec709, P3 and Rec2020 following convention "LPM_COLORS_" for replacing working, output and container gamut primary variables.

Now all we need to do is call LpmFilter in the shader. The call is as follows:

```
void LpmFilter(
// Input and output color.
inout AF1 colorR,inout AF1 colorG,inout AF1 colorB,
// Path control should all be compile-time immediates.
AP1 shoulder, // Using shoulder tuning.
// Prefab "LPM_CONFIG_" start, use the same as used for LpmSetup().
AP1 con, // Use first RGB conversion matrix, if 'soft' then 'con' must be true also.
AP1 soft, // Use soft gamut mapping.
AP1 con2, // Use last RGB conversion matrix.
AP1 clip, // Use clipping in last conversion matrix.
AP1 scaleOnly) // Scale only for last conversion matrix (used for 709 HDR to scRGB).
```

Let's go over every parameter:

- colorR, colorG, colorB: The first three inputs are simply the pixel color value. The output pixel value of mapper will also be stored in them.
- Shoulder: Boolean option to use shoulder contrast set in setup.
- The next five parameters con, soft, con2, clip and scaleOnly can be set using the same prefabs passed into the setup call.

The final output will not only be tone and gamut mapped, but it will also be modified appropriately based on the display mode the content is targeting.

Peek into the "Black Box"

Here is what gets calculated in LPMSetup and how it gets used in LPMFilter:

LPMSetup calculates the "toneScaleBias" which is used in LPMFilter to tone map luma. Along with that, LPMSetup calculates "lumaW" which is luma coefficient for the working color space and "lumaT" which is luma coefficient of the output color space or target luma in short. Finally, it calculates and stores gamut mapping related data like "softGap2" which is the feather region allocated for out of gamut values, and the required conversion matrices to go from working gamut space to output and container gamut spaces are stored in "conRGB" and "con2RGB" respectively.

LPMFilter reads from the control block LPMSetup writes to and invokes the function which does the mapping called LPMMap. In LPMMap, the first step is to calculate the pixel channel ratio of the input color. This is done by dividing each channel of the pixel with the max value amongst themselves.

```
AF1 rcpMax=ARcpF1(AMax3F1(colorR,colorG,colorB));
AF1 ratioR=colorR*rcpMax;AF1 ratioG=colorG*rcpMax;AF1 ratioB=colorB*rcpMax;
```

If any saturation is set from setup, it gets factored into the ratio.

```
ratioR=pow(ratioR,AF1_(saturation.r));
ratioG=pow(ratioG,AF1_(saturation.g));
ratioB=pow(ratioB,AF1_(saturation.b));
```

We then calculate the luma of the current pixel. The luma coefficient to use depends on whether gamut mapping is turned on. Working space luma coefficient is used if it's on, output space coefficient is used if it's not.

```
AF1 luma;
if(soft)luma=colorG*AF1_(lumaW.g)+(colorR*AF1_(lumaW.r)+(colorB*AF1_(lumaW.b)));
else luma=colorG*AF1_(lumaT.g)+(colorR*AF1_(lumaT.r)+(colorB*AF1_(lumaT.b)));
```

We factor in the contrast and shoulder contrast if applicable and tone map the luma.

```
luma=pow(luma,AF1_(contrast)); // (EXP2, LOG2, MUL).
AF1 lumaShoulder=shoulder?pow(luma,AF1_(shoulderContrast)):luma; // Optional (EXP2, LOG2, MUL).
luma=luma*ARcpF1(lumaShoulder*AF1_(toneScaleBias.x)+AF1_(toneScaleBias.y)); // (MAD, MUL, RCP).
```

Next we perform the gamut mapping as explained below:

```
// Absolute gamut mapping converted to soft falloff (maintains max 1 property).
//  g = gap {0 to g} used for {-inf to 0} input range
//          {g to 1} used for {0 to 1} input range
//  x >= 0 := y = x * (1-g) + g
//  x < 0  := g * 2^(x*h)
//  Where h=(1-g)/(g*log(2)) --- where log() is the natural log
// The {g,h} above is passed in as softGap.
```

Once gamut mapping is done, we calculate target luma ratio and find the ratio scale between luma and target luma.

```
// Compute ratio scaler required to hit target luma (4x MAD, 1 RCP).
AF1 lumaRatio=ratioR*AF1_(lumaT.r)+ratioG*AF1_(lumaT.g)+ratioB*AF1_(lumaT.b);
// This is limited to not clip.
AF1 ratioScale=ASatF1(luma*ARcpF1(lumaRatio));
```

Notice that if gamut mapping is off, input luma will match output or target luma and no further calculations are required. Otherwise, we want to make sure we maintain the tone mapped luma in the gamut mapped color output.

```
 // Assume in gamut, compute output color (3x MAD).
colorR=ASatF1(ratioR*ratioScale);colorG=ASatF1(ratioG*ratioScale);colorB=ASatF1(ratioB*ratioScale);
// Capability per channel to increase value (3x MAD).
// This factors in crosstalk factor to avoid multiplies later.
//  '(1.0-ratio)*crosstalk' optimized to '-crosstalk*ratio+crosstalk'
AF1 capR=AF1_(-crosstalk.r)*colorR+AF1_(crosstalk.r);
AF1 capG=AF1_(-crosstalk.g)*colorG+AF1_(crosstalk.g);
AF1 capB=AF1_(-crosstalk.b)*colorB+AF1_(crosstalk.b);
// Compute amount of luma needed to add to non-clipped channels to make up for clipping (3x MAD).
AF1 lumaAdd=ASatF1((-colorB)*AF1_(lumaT.b)+((-colorR)*AF1_(lumaT.r)+((-colorG)*AF1_(lumaT.g)+luma)));
// Amount to increase keeping over-exposure ratios constant and possibly exceeding clipping point (4x
MAD, 1 RCP).
AF1 t=lumaAdd*ARcpF1(capG*AF1_(lumaT.g)+(capR*AF1_(lumaT.r)+(capB*AF1_(lumaT.b))));
// Add amounts to base color but clip (3x MAD).
colorR=ASatF1(t*capR+colorR);colorG=ASatF1(t*capG+colorG);colorB=ASatF1(t*capB+colorB);
// Compute amount of luma needed to add to non-clipped channel to make up for clipping (3x MAD).
lumaAdd=ASatF1((-colorB)*AF1_(lumaT.b)+((-colorR)*AF1_(lumaT.r)+((-colorG)*AF1_(lumaT.g)+luma)));
```

The colors are finally clamped to {0, 1} range.

```
colorR=ASatF1(lumaAdd*AF1_(rcpLumaT.r)+colorR);
colorG=ASatF1(lumaAdd*AF1_(rcpLumaT.g)+colorG);
colorB=ASatF1(lumaAdd*AF1_(rcpLumaT.b)+colorB);
```

Lastly, any display mode dependent gamut conversion is done if con2 is set and the display mode dependent scaling required is applied to the final color value before being returned.

```
if(con2){
  ratioR=colorR;ratioG=colorG;ratioB=colorB;
  if(clip){
    colorR=ASatH2(ratioR*AH2_(con2R.r)+(ratioG*AH2_(con2R.g)+(ratioB*AH2_(con2R.b))));
    colorG=ASatH2(ratioG*AH2_(con2G.g)+(ratioR*AH2_(con2G.r)+(ratioB*AH2_(con2G.b))));
    colorB=ASatH2(ratioB*AH2_(con2B.b)+(ratioG*AH2_(con2B.g)+(ratioR*AH2_(con2B.r))));}
  else{
    colorR=ratioR*AH2_(con2R.r)+(ratioG*AH2_(con2R.g)+(ratioB*AH2_(con2R.b)));
    colorG=ratioG*AH2_(con2G.g)+(ratioR*AH2_(con2G.r)+(ratioB*AH2_(con2G.b)));
    colorB=ratioB*AH2_(con2B.b)+(ratioG*AH2_(con2B.g)+(ratioR*AH2_(con2B.r)));}}

if(scaleOnly){colorR*=AH2_(con2R.r);colorG*=AH2_(con2R.r);colorB*=AH2_(con2R.r);}}
```

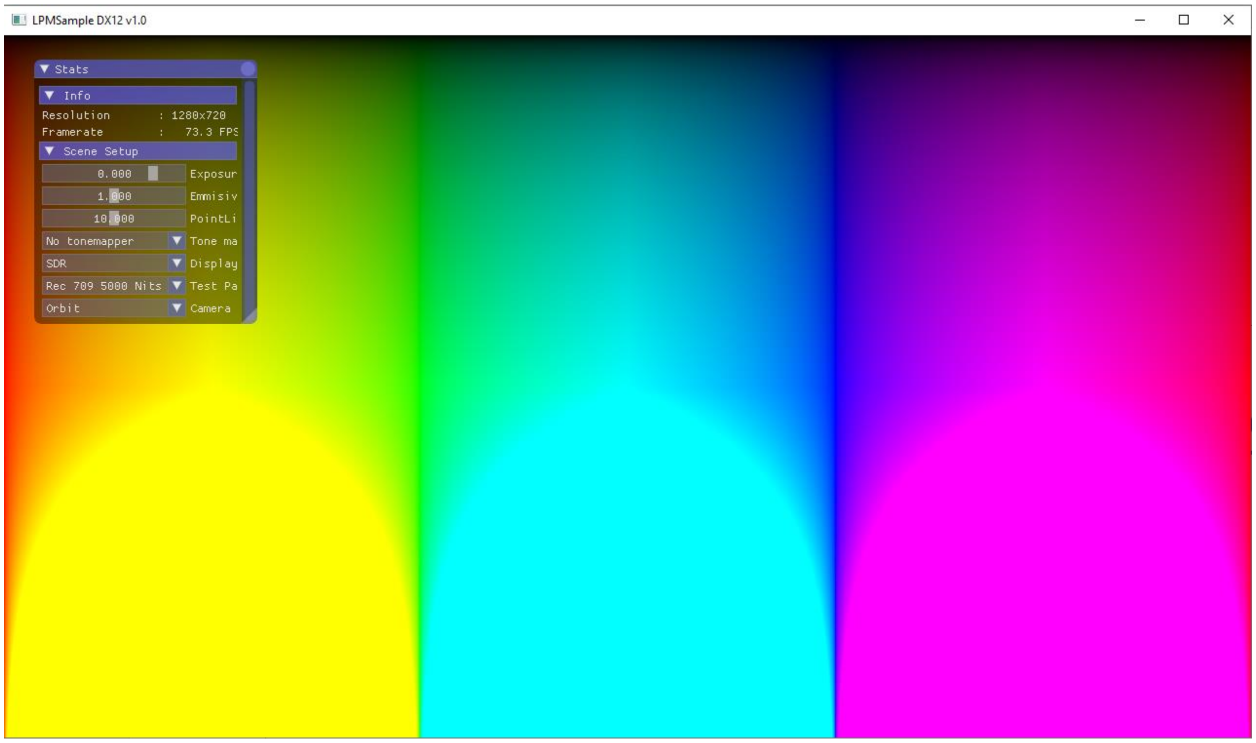
Tuning Crosstalk

One of the important parameters that LPM can adjust is crosstalk, and this decides how the colors look on overexposure. Between the extreme case scenarios of absolute clamped saturation and complete desaturation, lies the ideal setup which depends on the content.

Crosstalk is a three channel RGB value which tints color saturation on overexposure.

Let us go over tuning crosstalk.

LPM Off(clamped saturation)



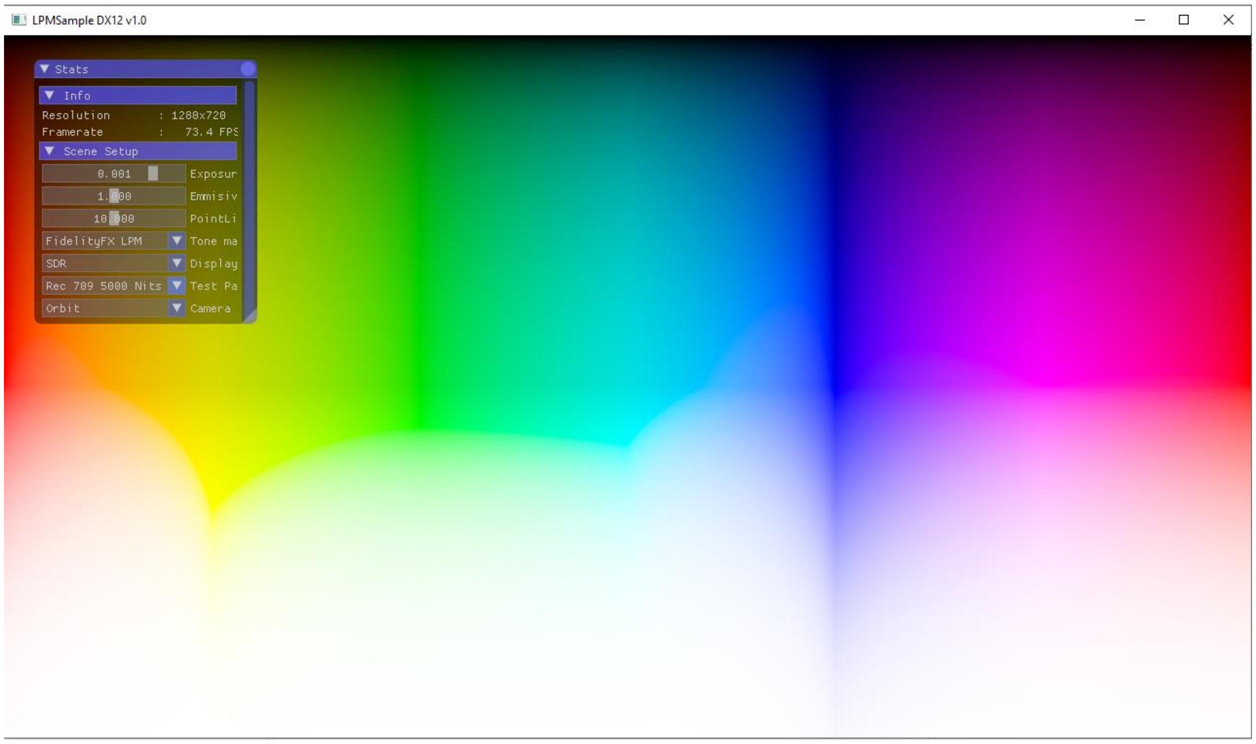
```
Resolution     : 1280x720
Framerate      :   73.3 FPS
```

0.000	Exposur
1.000	Emmisiv
10.000	PointLi
No tonemapper	Tone ma
SDR	Display
Rec 709 5000 Nits	Test Pa
Orbit	Camera

As you can see in this image, we have LPM turned off, and with no tone mapping the saturation is completely clamped to a fixed value. Not ideal as a huge chunk of values look all the same.

Now let’s see how it looks when LPM is turned on with default crosstalk setup of RGB(1, 1, 1).

LPM On (Crosstalk(1, 1, 1) complete desaturation)

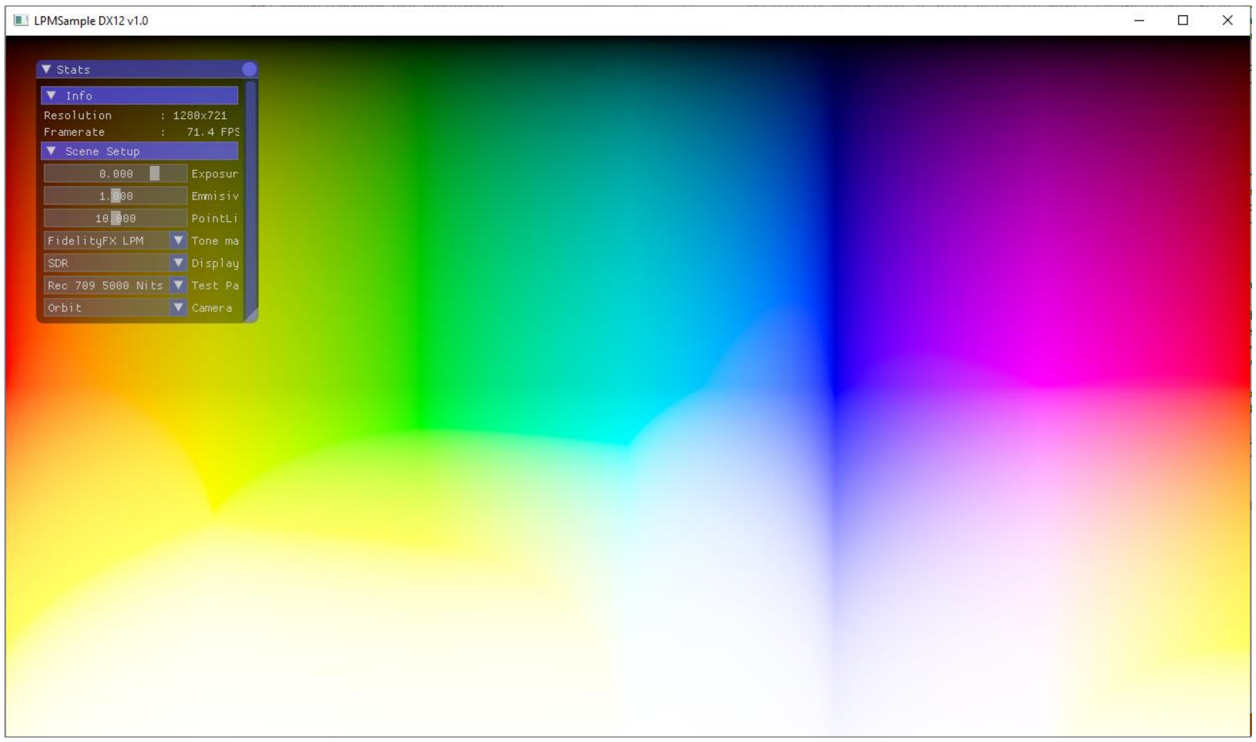


```
Resolution      : 1280x720
Framerate       :   73.4 FPS
0.001               Exposur
1.000               Emmisiv
10.000              PointLi
FidelityFX LPM      Tone ma
SDR                 Display
Rec 709 5000 Nits   Test Pa
Orbit               Camera
```

In the above image, we can see a complete desaturation of colors when values are overexposed. This is also not ideal, and colors quickly clamp to white and we don't see any interesting detail.

What we want to see is colors slowly desaturate, something like going from red to orange to yellow to then finally white. To get that look, we first start by reducing the blue channel of cross talk.

Crosstalk(1,1,1/(16,32,64,128))



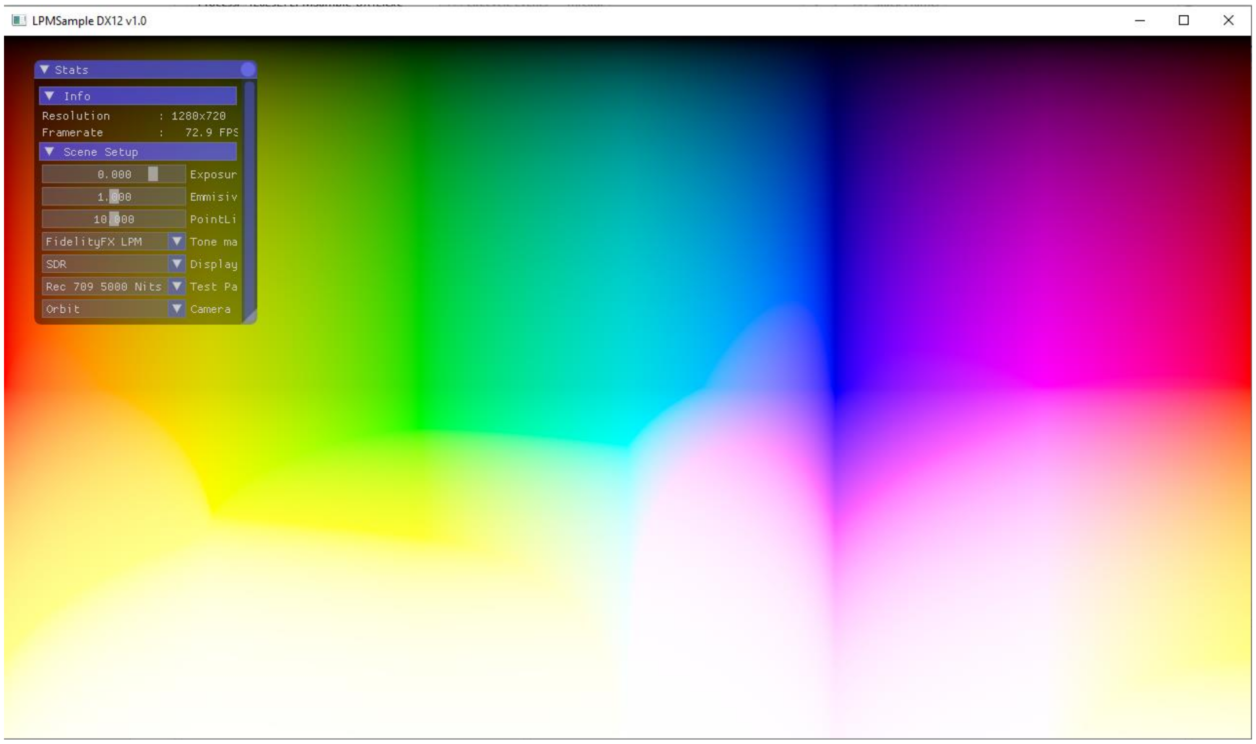
```
LPMSample DX12 v1.0

Stats
Info
Resolution     : 1280x721
Framerate      :   71.4 FPS
Scene Setup
0.000               Exposur
1.000               Emmisiv
10.000              PointLi
FidelityFX LPM      Tone ma
SDR                 Display
Rec 709 5000 Nits   Test Pa
Orbit               Camera
```

Now you can see it transition gradually to white for red and green channel. The transition can be tuned by how small the blue channel is. However, notice how blue still desaturates immediately. To tackle that, we have two choices.

First is to reduce green channel to tint blue to purple or magenta.

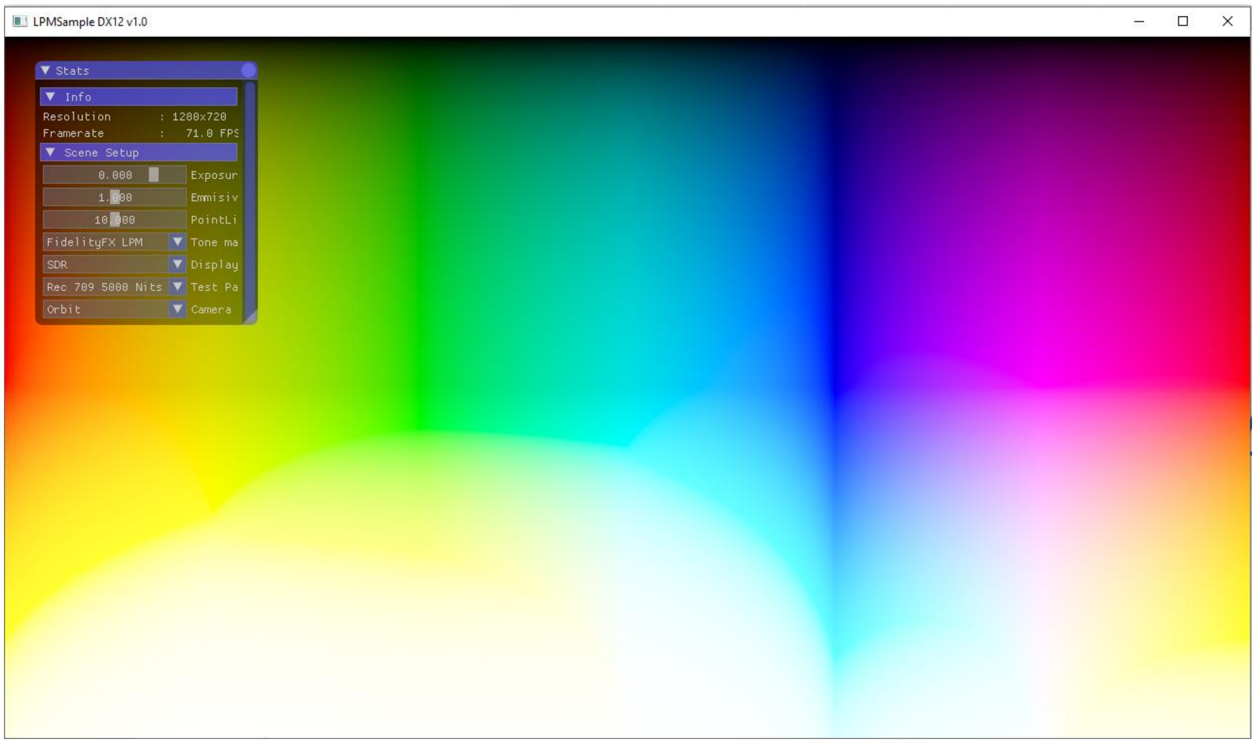
Crosstalk(1, 1/2, 1/(16,32,64,128))



Resolution : 1280x720
Framerate : 72.9 FPS
0.000
1.000
10.000
FidelityFX LPM
SDR
Rec 709 5000 Nits
Orbit

Second option is to reduce red and tint blue towards cyan.

Crosstalk(1/2, 1, 1/(16,32,64,128))



Resolution : 1280x720
Framerate : 71.0 FPS
0.000
1.000
10.000
FidelityFX LPM
SDR
Rec 709 5000 Nits
Orbit

Summary

As you can, LPM allows for extensive tuning and customization for HDR and wide gamut content along with easy set up prefab configurations for multiple display modes like SDR, HDR10 and Freesync Premium Pro.