Shadertoy to Pixera GLSL Tutorial

17 Minutes | 0 Likes | 35 Views

STEP 0: EXPLANATION

Shadertoy GLSL

Shadertoy is a website allowing users to create and share GLSL shaders with unique quality-of-life additions such as Channels for media and built-in timers. Due to this, Shadertoy allows its own slightly altered GLSL format to run code depending on the shader being built or imported. Even so, the platform holds a large variety of unique and useful effects that Pixera users may find desirable to use in their projects.

Pixera GLSL

Pixera's use of effects also involves GLSL code. However, just like Shadertoy, Pixera requires a specific format for the code to run. Due to the software's many uses, several specific lines of code must be precise for the shader to run correctly.

Shadertoy to Pixera

This article will include step-by-step instructions on how to convert most Shadertoy shaders into a Pixera effect. Some steps are optional and are mainly for bug fixing or organization of the code. We will be going over **Benjamin ‘BeRo’ Rosseaux's Color Temperature shader**.

Creative Commons Copyright Information

```
 // Color temperature (sRGB) stuff
// Copyright (C) 2014 by Benjamin 'BeRo' Rosseaux
// Because the german law knows no public domain
```

```
in the usual sense,
// this code is licensed under the CC0 license
//
http://creativecommons.org/publicdomain/zero/1.0/
const float LuminancePreservationFactor = 1.0;
const float PI2 = 6.2831853071;
// Valid from 1000 to 40000 K (and additionally 0
for pure full white)
vec3 colorTemperatureToRGB(const in float
temperature){
 // Values from:
http://blenderartists.org/forum/showthread.php?
270332-OSL-
Goodness&p=2268693&viewfull=1#post2268693
 mat3 m = (temperature <= 6500.0) ?
mat3(vec3(0.0, -2902.1955373783176,
-8257.7997278925690),
      vec3(0.0, 1669.5803561666639,
2575.2827530017594),
      vec3(1.0, 1.3302673723350029,
1.8993753891711275)) :
         mat3(vec3(1745.0425298314172,
1216.6168361476490, -8257.7997278925690),
      vec3(-2666.3474220535695,
-2173.1012343082230, 2575.2827530017594),
      vec3(0.55995389139931482,
0.70381203140554553, 1.8993753891711275));
 return mix(clamp(vec3(m[0] /
(vec3(clamp(temperature, 1000.0, 40000.0)) +
m[1]) + m[2]), vec3(0.0), vec3(1.0)), vec3(1.0),
smoothstep(1000.0, 0.0, temperature));
}
```

STEP 1: IDE

GPUpad IDE

Using an Integrated Development Environment (IDE) with tools for GLSL programming will aid the user in making sure their code works correctly before bringing it into Pixera. GPUpad is the recommended IDE due to a custom project being available emulating how Pixera will use the file in the software. If the effect works on the test image given in GPUpad, it is almost certain to work in Pixera.

Installation and Setup

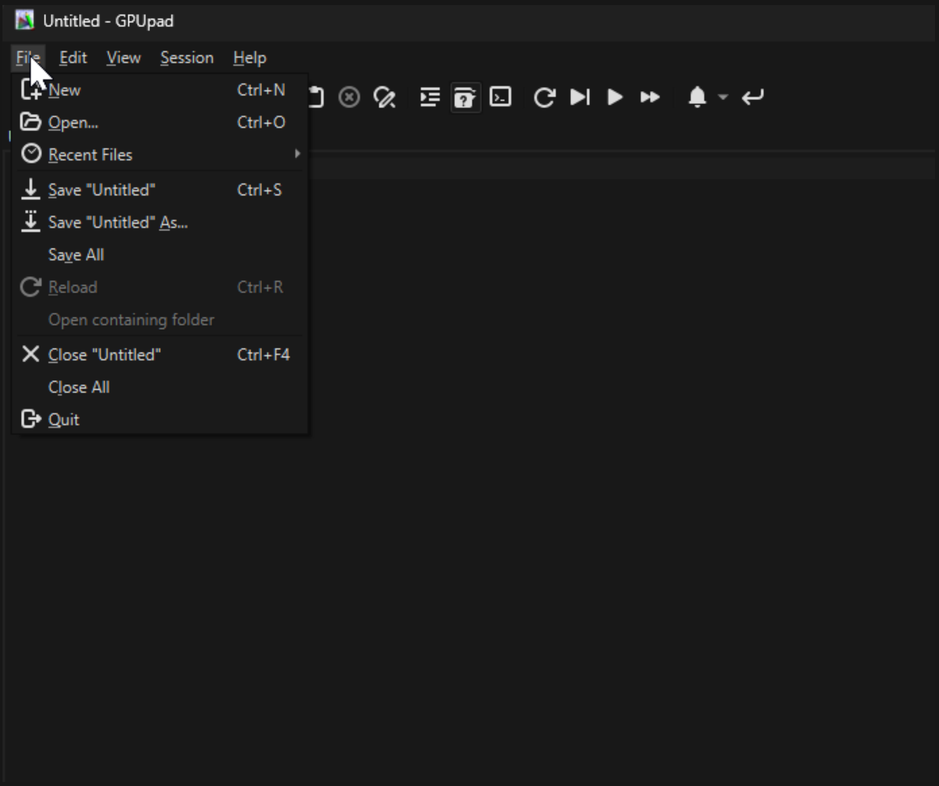
Download and install the GPUpad Windows 64-bit .exe file from GitHub linked here:

[Download GPUpad]

Also, download this starter code file and unzip it into a place of your choosing (It is recommended to change the main folder name to GPUpad for consistency):

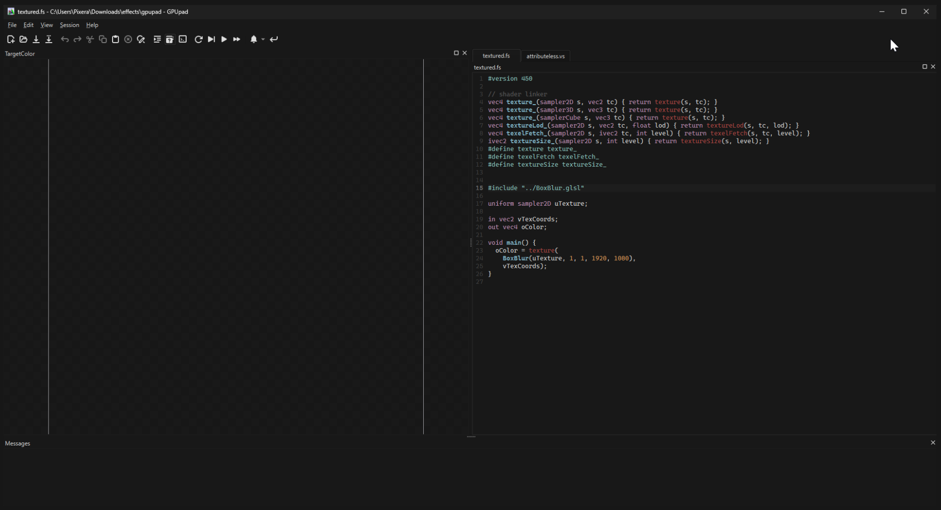
[Download GPUpad Starter Code]

Once it is installed and the starter code is unzipped, open GPUpad and navigate to the top left corner to select "File/Open…" or press "Ctrl + O".



Navigate to the GPUpad Starter Code folder and in the "gpupad" folder select the “effects.gpjs” file.

A new UI will appear which includes a visual space to the left as well as two opened files, one named “textured.fs” and another names “attributeless.vs” which should look like this:


textured.fs - C:\Users\Pixera\Downloads\effects\gpupad - GPUpad
File Edit View Session Help
TargetColor
textured.fs
attributeless.vs
Messages

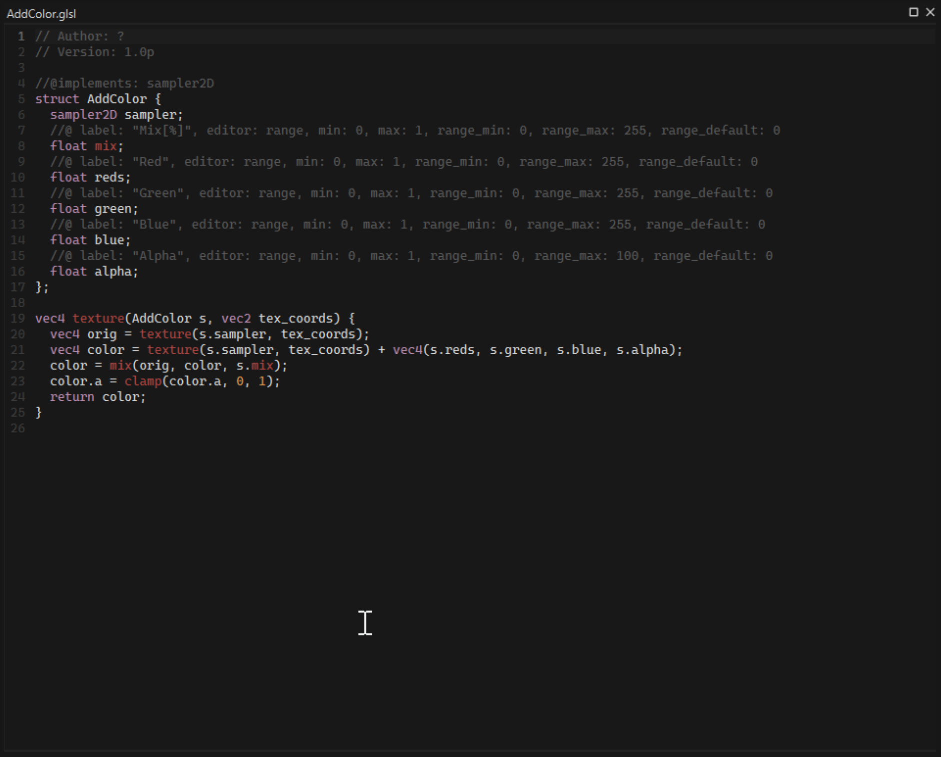
```
#version 450

// shader linker
vec4 texture_(sampler2D s, vec2 tc) { return texture(s, tc); }
vec4 texture_(sampler3D s, vec3 tc) { return texture(s, tc); }
vec4 texture_(samplerCube s, vec3 tc) { return texture(s, tc); }
vec4 textureLod_(sampler2D s, vec2 tc, float lod) { return textureLod(s, tc, lod); }
vec4 texelFetch_(sampler2D s, ivec2 tc, int level) { return texelFetch(s, tc, level); }
ivec2 textureSize_(sampler2D s, int level) { return textureSize(s, level); }
#define texture texture_
#define texelFetch texelFetch_
#define textureSize textureSize_


#include "../BoxBlur.glsl"

uniform sampler2D uTexture;

in vec2 vTexCoords;
out vec4 oColor;

void main() {
  oColor = texture(
    BoxBlur(uTexture, 1, 1, 1920, 1080),
    vTexCoords);
}
```

On the left is the visual feed which will show how the effect is changing an already given image. To see this image press the play button in the top bar. On the right side is some code. The two sections the user must focus on are the "#include "../BoxBlur.glsl"" and the "BoxBlur(uTexture, 1, 1, 1920, 1080)" in the "void main()" method. These two lines of code will need to be changed to whatever format needed for the selected effect to change the image.

The “#include” line just requires the user to change the directory in which the file can be located. In this case the code is pointing to the “BoxBlur.glsl” file which is back one directory from the “effects.gpjs” file. As an example, another effect available in the starter code is the “AddColor.glsl”, which can be added by placing this directory in the quotation marks: “../AddColor.glsl”.

As for the code in void main(), the name of the function being run must be changed to the new effect's name, in this case “AddColor(uTexture, 1, 1, 1920, 1080)”. The parameters in the parentheses describe the values the user wants the effect to use based on its built in struct which will be discussed later in this tutorial. For now know that most effects will use the uTexture variable and a value of 1 in the first two parameter spaces. The other parameters will have to be decided after opening the GLSL file in GPUpad (using “File/Open…”).

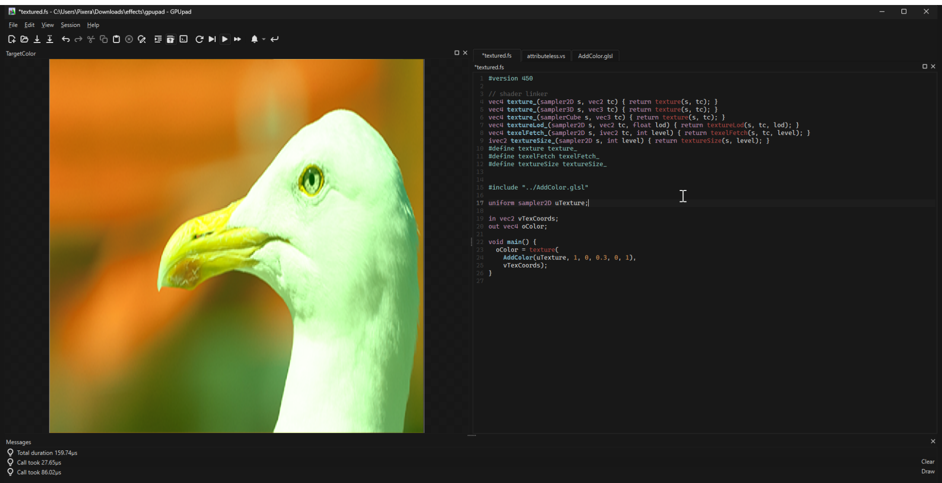
For example, the “AddColor.glsl” when opened will look like this:



```glsl
// Author: ?
// Version: 1.0p

//@implements: sampler2D
struct AddColor {
  sampler2D sampler;
  //@ label: "Mix[%]", editor: range, min: 0, max: 1, range_min: 0, range_max: 255, range_default: 0
  float mix;
  //@ label: "Red", editor: range, min: 0, max: 1, range_min: 0, range_max: 255, range_default: 0
  float reds;
  //@ label: "Green", editor: range, min: 0, max: 1, range_min: 0, range_max: 255, range_default: 0
  float green;
  //@ label: "Blue", editor: range, min: 0, max: 1, range_min: 0, range_max: 255, range_default: 0
  float blue;
  //@ label: "Alpha", editor: range, min: 0, max: 1, range_min: 0, range_max: 100, range_default: 0
  float alpha;
};

vec4 texture(AddColor s, vec2 tex_coords) {
  vec4 orig = texture(s.sampler, tex_coords);
  vec4 color = texture(s.sampler, tex_coords) + vec4(s.reds, s.green, s.blue, s.alpha);
  color = mix(orig, color, s.mix);
  color.a = clamp(color.a, 0, 1);
  return color;
}
```

The uTexture (Reference to the duck image) and the first 1 value mentioned previously represent the sampler and mix variables in “struct AddColor”. This effect also includes four more values that the user can change, reds, green, blue, and alpha. Looking at the min and max values, these floats can be set to as high as 1 and as low as 0.

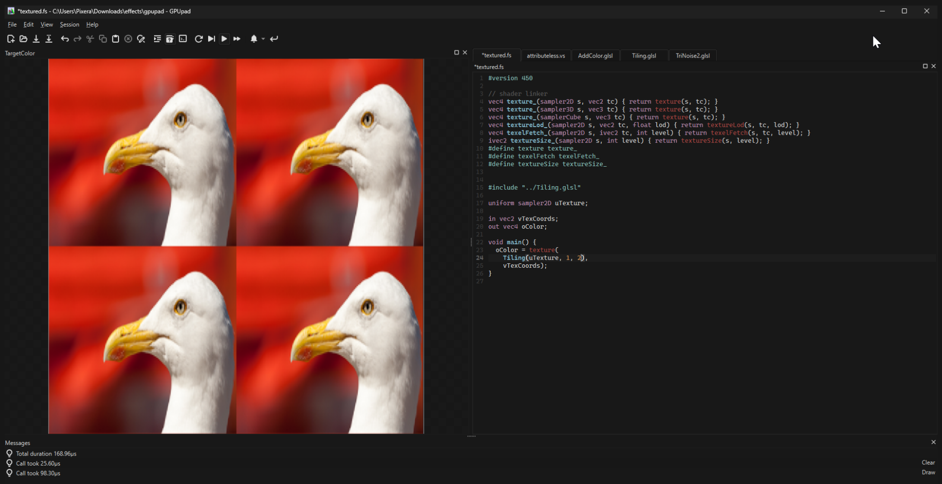
Going back to the "textured.fs" file, you can now replace the values inside the parentheses with the respective struct variable values you would like to be used. Here is an example:


*textured.fs - C:\Users\Pixera\Downloads\effects\gpupad - GPUpad
File Edit View Session Help
TargetColor
*textured.fs
attributeless.vs
AddColor.glsl
Clear
Draw

```
#version 450

// shader linker
vec4 texture_(sampler2D s, vec2 tc) { return texture(s, tc); }
vec4 texture_(sampler3D s, vec3 tc) { return texture(s, tc); }
vec4 texture_(samplerCube s, vec3 tc) { return texture(s, tc); }
vec4 textureLod_(sampler2D s, vec2 tc, float lod) { return textureLod(s, tc, lod); }
vec4 texelFetch_(sampler2D s, ivec2 tc, int level) { return texelFetch(s, tc, level); }
ivec2 textureSize_(sampler2D s, int level) { return textureSize(s, level); }
#define texture texture_
#define texelFetch texelFetch_
#define textureSize textureSize_


#include "../AddColor.glsl"

uniform sampler2D uTexture;

in vec2 vTexCoords;
out vec4 oColor;

void main() {
  oColor = texture(
    AddColor(uTexture, 1, 0, 0.3, 0, 1),
    vTexCoords);
}
```

```
Messages
Total duration 159.74µs
Call took 27.65µs
Call took 86.02µs
```

Here the line “AddColor(uTexture, 1, 0, 0.3, 0, 1)” is used. Again the function name must be the name of the effect, the first two parameters will usually be uTexture and a 1. The third value represents the reds parameter which is set to 0. The fourth value is the green parameter which is set to 0.3. the Fifth value represents the blue parameter which is set to 0. Lastly the sixth value represents the alpha parameter which is set to 1. This combination of values creates a green tint to the image as only the green channel is being used to mix with the uTexture. Here is another example using the “Tiling.glsl” effect:


*textured.fs - C:\Users\Pixera\Downloads\effects\gpupad - GPUpad
File Edit View Session Help
TargetColor
*textured.fs
attributeless.vs
AddColor.glsl
Tiling.glsl
TriNoise2.glsl
Clear
Draw

```
#version 450

// shader linker
vec4 texture_(sampler2D s, vec2 tc) { return texture(s, tc); }
vec4 texture_(sampler3D s, vec3 tc) { return texture(s, tc); }
vec4 texture_(samplerCube s, vec3 tc) { return texture(s, tc); }
vec4 textureLod_(sampler2D s, vec2 tc, float lod) { return textureLod(s, tc, lod); }
vec4 texelFetch_(sampler2D s, ivec2 tc, int level) { return texelFetch(s, tc, level); }
ivec2 textureSize_(sampler2D s, int level) { return textureSize(s, level); }
#define texture texture_
#define texelFetch texelFetch_
#define textureSize textureSize_


#include "../Tiling.glsl"

uniform sampler2D uTexture;

in vec2 vTexCoords;
out vec4 oColor;

void main() {
  oColor = texture(
    Tiling(uTexture, 1, 2),
    vTexCoords);
}
```

```
Messages
Total duration 168.96µs
Call took 25.60µs
Call took 98.30µs
```

In all, GPUpad will make creating the shader simpler with its visual of the effect and ability to log errors quickly for debugging.

STEP 2: SETUP

Beginning and End Comparison

To start, here are two different samples of code. The first is from Shadertoy and the second is the code fully translated into a Pixera effect and organized for efficient use within the timeline. For now, copy the Shadertoy code to a new GLSL file named “ColorTemperature.glsl”. After following this article any user should have a working ColorTemperature effect in Pixera.

Shadertoy

```
// Color temperature (sRGB) stuff
// Copyright (C) 2014 by Benjamin 'BeRo' Rosseaux
// Because the german law knows no public domain
in the usual sense,
// this code is licensed under the CC0 license
//
http://creativecommons.org/publicdomain/zero/1.0/
#define WithQuickAndDirtyLuminancePreservation
const float LuminancePreservationFactor = 1.0;
const float PI2 = 6.2831853071;
// Valid from 1000 to 40000 K (and additionally 0
for pure full white)
vec3 colorTemperatureToRGB(const in float
temperature){
 // Values from:
http://blenderartists.org/forum/showthread.php?
270332-OSL-
Goodness&p=2268693&viewfull=1#post2268693
 mat3 m = (temperature <= 6500.0) ?
mat3(vec3(0.0, -2902.1955373783176,
-8257.7997278925690),

vec3(0.0, 1669.5803561666639,
2575.2827530017594),

vec3(1.0, 1.3302673723350029,
1.8993753891711275)) :

mat3(vec3(1745.0425298314172, 1216.6168361476490,
-8257.7997278925690),

vec3(-2666.3474220535695, -2173.1012343082230,
2575.2827530017594),

vec3(0.55995389139931482, 0.70381203140554553,
1.8993753891711275));
 return mix(clamp(vec3(m[0] /
(vec3(clamp(temperature, 1000.0, 40000.0)) +
m[1]) + m[2]), vec3(0.0), vec3(1.0)), vec3(1.0),
smoothstep(1000.0, 0.0, temperature));
}
void mainImage( out vec4 fragColor, in vec2
fragCoord )
{
    vec2 uv = fragCoord.xy / iResolution.xy;
 float temperature = (iMouse.z > 0.0) ?
mix(1000.0, 40000.0, iMouse.x / iResolution.x) :
6550.0; // mix(1000.0, 15000.0, (sin(iTime * (PI2
/ 10.0)) * 0.5) + 0.5);
 float temperatureStrength = (iMouse.z > 0.0) ?
(1.0 - clamp((iMouse.y / iResolution.y) * (1.0 /
0.9), 0.0, 1.0)) : 1.0;
 if(uv.y > 0.1){
   vec3 inColor = texture(iChannel0, uv).xyz;
   vec3 outColor = mix(inColor, inColor *
colorTemperatureToRGB(temperature),
temperatureStrength);
   #ifdef WithQuickAndDirtyLuminancePreservation
     outColor *= mix(1.0, dot(inColor,
vec3(0.2126, 0.7152, 0.0722)) / max(dot(outColor,
vec3(0.2126, 0.7152, 0.0722)), 1e-5),
LuminancePreservationFactor);
   #endif
     fragColor = vec4(outColor, 1.0);
 }else{
   vec2 f = vec2(1.5) / iResolution.xy;
      fragColor =
vec4(mix(colorTemperatureToRGB(mix(1000.0,
40000.0, uv.x)), vec3(0.0),
min(min(smoothstep(uv.x - f.x, uv.x, (temperature
- 1000.0) / 39000.0),

smoothstep(uv.x + f.x, uv.x, (temperature -
1000.0) / 39000.0)),
1.0 - min(smoothstep(0.04 - f.y, 0.04, uv.y),

smoothstep(0.06 + f.y, 0.06, uv.y)))),

1.0);
 }
}
```

Pixera

```
// Author: Cody Luketic

// Color temperature (sRGB) stuff
// Copyright (C) 2014 by Benjamin 'BeRo' Rosseaux
// Because the german law knows no public domain
in the usual sense,
// this code is licensed under the CC0 license
//
http://creativecommons.org/publicdomain/zero/1.0/
const float LuminancePreservationFactor = 1.0;
const float PI2 = 6.2831853071;
// Valid from 1000 to 40000 K (and additionally 0
for pure full white)
vec3 colorTemperatureToRGB(const in float
temperature){
 // Values from:
http://blenderartists.org/forum/showthread.php?
270332-OSL-
Goodness&p=2268693&viewfull=1#post2268693
 mat3 m = (temperature <= 6500.0) ?
mat3(vec3(0.0, -2902.1955373783176,
-8257.7997278925690),
      vec3(0.0, 1669.5803561666639,
2575.2827530017594),
      vec3(1.0, 1.3302673723350029,
1.8993753891711275)) :
         mat3(vec3(1745.0425298314172,
1216.6168361476490, -8257.7997278925690),
      vec3(-2666.3474220535695,
-2173.1012343082230, 2575.2827530017594),
      vec3(0.55995389139931482,
0.70381203140554553, 1.8993753891711275));
 return mix(clamp(vec3(m[0] /
(vec3(clamp(temperature, 1000.0, 40000.0)) +
m[1]) + m[2]), vec3(0.0), vec3(1.0)), vec3(1.0),
smoothstep(1000.0, 0.0, temperature));
}

//@implements: sampler2D
struct ColorTemperature {
 sampler2D sampler;
 //@ label: "Mix[%]", editor: range, min: 0, max:
1, range_min: 0, range_max: 1, range_default: 0.5
 float mix;
 //@ label: "Kelvin Preset", editor: enum,
enum_default: 0, values: "No_Preset,
Candle_1800K, Sunrise_Sunset_2500K,
Tungsten_3000K, Morning_3500K, Moonlight_4000K,
Midday_5500K, Cloudy_6500K, Shade_7000K,
Clear_Sky_10000K"
    int idKelvinPreset;

 //@ label: "Kelvin", editor: range, min: 1000,
max: 10000, range_min: 1000, range_max: 10000,
range_default: 1000
 float kelvin;
 //@ label: "Brightness[%]", editor: range, min:
0, max: 1, range_min: 0, range_max: 1,
range_default: 1
 float luminancePreservationFactor;
};
vec4 texture(ColorTemperature s, vec2 tex_coords)
{
 vec3 inColor = texture(s.sampler,
tex_coords).xyz;
 vec3 outColor = mix(inColor, inColor *
colorTemperatureToRGB(s.kelvin), s.mix);
 switch(s.idKelvinPreset) {
 case 0:
   break;
 case 1:
   outColor = mix(inColor, inColor *
colorTemperatureToRGB(1800.0), s.mix);
   break;
 case 2:
   outColor = mix(inColor, inColor *
colorTemperatureToRGB(2500.0), s.mix);
   break;
 case 3:
   outColor = mix(inColor, inColor *
colorTemperatureToRGB(3000.0), s.mix);
   break;
 case 4:
   outColor = mix(inColor, inColor *
colorTemperatureToRGB(3500.0), s.mix);
   break;
 case 5:
   outColor = mix(inColor, inColor *
colorTemperatureToRGB(4000.0), s.mix);
   break;
 case 6:
   outColor = mix(inColor, inColor *
colorTemperatureToRGB(5500.0), s.mix);
   break;
 case 7:
   outColor = mix(inColor, inColor *
colorTemperatureToRGB(6500.0), s.mix);
   break;
 case 8:
   outColor = mix(inColor, inColor *
colorTemperatureToRGB(7000.0), s.mix);
   break;
 case 9:
   outColor = mix(inColor, inColor *
colorTemperatureToRGB(10000.0), s.mix);
   break;
 default:
   outColor = vec3(1.0);
   break;
 }
 outColor *= mix(1.0, dot(inColor, vec3(0.2126,
0.7152, 0.0722)) / max(dot(outColor, vec3(0.2126,
0.7152, 0.0722)), 1e-5),
s.luminancePreservationFactor);
 return vec4(outColor, 1.0);
}
```

Basic Replacement

Every Shadertoy project holds these lines of code which must be changed to their equivalent listed:

Texture Method

Shadertoy	Pixera
`void mainImage(out vec4 fragColor, in vec2 fragCoord)`	`vec4 texture([Name of File] s, vec2 tex_coords)`

Pixera works with a method called “texture()” which returns a vec4 instead of the typical void method. This texture takes a struct which should be named the same as the GLSL file. It also takes a “vec2 tex_coords” which is equivalent to a normalized uv. Due to this, tex_coords takes the place of uv in Shadertoy code, allowing you to delete this following line completely:

```
vec2 uv = fragCoord.xy / iResolution.xy;
```

fragColor and Return Statement

Shadertoy	Pixera
`fragColor = vec4(outColor, 1.0);`	`return vec4(outColor, 1.0);`

Due to the texture method returning a vec4, the final line setting fragColor in most Shadertoy code must be changed to just return a vec4.

iResolution

Shadertoy	Pixera

`vec2 Radius = 9 / iResolution.xy`	`vec2 iResolution = vec2(s.width, s.height);`
	
	`vec2 Radius = 9 / iResolution.xy`

Many Shadertoy projects contain the iResolution variable which is native to the software. iResolution contains the display's dimensions in pixels. For Pixera, the user must create their own iResolution variable using width and height variables set on the timeline.

From here, all instances of fragColor, fragCoord, and uv must be removed or replaced by the parts explained previously.

After these changes, the code should look like this:

```
// Color temperature (sRGB) stuff
// Copyright (C) 2014 by Benjamin 'BeRo' Rosseaux
// Because the german law knows no public domain
in the usual sense,
// this code is licensed under the CC0 license
//
http://creativecommons.org/publicdomain/zero/1.0/
#define WithQuickAndDirtyLuminancePreservation
const float LuminancePreservationFactor = 1.0;
const float PI2 = 6.2831853071;
// Valid from 1000 to 40000 K (and additionally 0
for pure full white)
vec3 colorTemperatureToRGB(const in float
temperature){
 // Values from:
http://blenderartists.org/forum/showthread.php?
270332-OSL-
Goodness&p=2268693&viewfull=1#post2268693
```

```
 mat3 m = (temperature <= 6500.0) ?
mat3(vec3(0.0, -2902.1955373783176,
-8257.7997278925690),

vec3(0.0, 1669.5803561666639,
2575.2827530017594),

vec3(1.0, 1.3302673723350029,
1.8993753891711275)) :

mat3(vec3(1745.0425298314172, 1216.6168361476490,
-8257.7997278925690),

vec3(-2666.3474220535695, -2173.1012343082230,
2575.2827530017594),

vec3(0.55995389139931482, 0.70381203140554553,
1.8993753891711275));
 return mix(clamp(vec3(m[0] /
(vec3(clamp(temperature, 1000.0, 40000.0)) +
m[1]) + m[2]), vec3(0.0), vec3(1.0)), vec3(1.0),
smoothstep(1000.0, 0.0, temperature));
}
vec4 texture(ColorTemperature s, vec2 tex_coords)
{
 vec2 iResolution = vec2(s.width, s.height);
 float temperature = (iMouse.z > 0.0) ?
mix(1000.0, 40000.0, iMouse.x / iResolution.x) :
6550.0; // mix(1000.0, 15000.0, (sin(iTime * (PI2
/ 10.0)) * 0.5) + 0.5);
 float temperatureStrength = (iMouse.z > 0.0) ?
(1.0 - clamp((iMouse.y / iResolution.y) * (1.0 /
0.9), 0.0, 1.0)) : 1.0;
 if(tex_coords.y > 0.1){
   vec3 inColor = texture(iChannel0,
tex_coords).xyz;
   vec3 outColor = mix(inColor, inColor *
colorTemperatureToRGB(temperature),
temperatureStrength);
   #ifdef WithQuickAndDirtyLuminancePreservation
     outColor *= mix(1.0, dot(inColor,
vec3(0.2126, 0.7152, 0.0722)) / max(dot(outColor,
vec3(0.2126, 0.7152, 0.0722)), 1e-5),
LuminancePreservationFactor);
   #endif
   return vec4(outColor, 1.0);
 }else{
   vec2 f = vec2(1.5) / iResolution.xy;
      fragColor =
vec4(mix(colorTemperatureToRGB(mix(1000.0,
40000.0, tex_coords.x)), vec3(0.0),
min(min(smoothstep(tex_coords.x - f.x,
tex_coords.x, (temperature - 1000.0) / 39000.0),

smoothstep(tex_coords.x + f.x, tex_coords.x,
(temperature - 1000.0) / 39000.0)),

1.0 - min(smoothstep(0.04 - f.y, 0.04,
tex_coords.y),

smoothstep(0.06 + f.y, 0.06, tex_coords.y)))),

1.0);
 }
}
```

Creating the Struct

Every Pixera effect includes a struct that must be named the same as the file's name. In this case, let's create the struct like this:

```
//@implements: sampler2D
struct ColorTemperature {
 sampler2D sampler;
 //@ label: "Mix[%]", editor: range, min: 0, max:
1, range_min: 0, range_max: 1, range_default: 0.5
 float mix;
 //@ label: "Kelvin Preset", editor: enum,
enum_default: 0, values: "No_Preset,
Candle_1800K, Sunrise_Sunset_2500K,
Tungsten_3000K, Morning_3500K, Moonlight_4000K,
Midday_5500K, Cloudy_6500K, Shade_7000K,
Clear_Sky_10000K"
```

```
    int idKelvinPreset;

 //@ label: "Kelvin", editor: range, min: 1000,
max: 10000, range_min: 1000, range_max: 10000,
range_default: 1000
 float kelvin;
 //@ label: "Brightness[%]", editor: range, min:
0, max: 1, range_min: 0, range_max: 1,
range_default: 1
 float luminancePreservationFactor;
};
```

sampler2D

sampler2D is used to hold data about media being used in GLSL. In Pixera's case, the code around the struct involving sampler2D references the current media playing on the layer. This also includes any effects already being applied to the layer. sampler2D allows effects to be layers together onto whatever media the user is playing.

Timeline Variables

The variables shown here, including mix, luminancePreservationFactor, and kelvin, will become timeline variables that can be changed in Pixera depending on the specified data given in their above comments. There are a variety of types that can be given for the struct variables. Here is a list of possibilities:

Slider

```
//@ label: "Name", editor: range, min: 0, max: 1,
range_min: 0, range_max: 1, range_default: 0.5
```

Creates a slider going from min to max which starts at range_default.

Sampler

```
//@ label: “Sampler”, editor: sampler
sampler2D samplerMask
```

Creates a parameter allowing the user to drag media into the space which can be used in the GLSL code.

Boolean/Switch

```
//@ label: "Border is Smooth", editor: bool,
bool_default: true
bool isSmooth;
```

Creates a dropdown allowing the user to select true or false for simple on-and-off or switching mechanisms in the code.

Discrete/Uneditable

```
//@ label: "Number of Frames", editor: discrete,
discrete_min: 0, discrete_max: 3,
discrete_default: 1
int ctFrames;
```

Creates a visual of the value to allow information to be easily seen in the timeline.

Enum/Prebuilt Options

```
//@ label: "Frame Color", editor: enum,
enum_default: 1, values: “White, Black, Red,
Green, Blue, Yellow”
int idColorFrame;
```

Creates a dropdown of the listed options which the user can select. These options are stored as IDs starting from 0. Can be used to change paths through code using if or switch statements.

mix

The mix variable is commonly used in Pixera Effects as it allows the user to fade the effect in and out by using the mix method inherent with GLSL. Here is an example:

```
return mix(colorIn, colorOut, s.mix);
```

Depending on the value of mix set in the timeline, the effect can either be fully hidden or take over the media completely. The “s” relates back to the parameter of the “vec4 texture(ColorTemperature s, vec2 tex_Coords)” since “s” is an instance of the ColorTemperature struct. To use any variable in the struct, just add “s."Variable Name"” where necessary.

After adding the struct and replacing the original variables with the structs, the code should look like this:

```
// Color temperature (sRGB) stuff
// Copyright (C) 2014 by Benjamin 'BeRo' Rosseaux
// Because the german law knows no public domain
in the usual sense,
// this code is licensed under the CC0 license
//
http://creativecommons.org/publicdomain/zero/1.0/
#define WithQuickAndDirtyLuminancePreservation
const float LuminancePreservationFactor = 1.0;
const float PI2 = 6.2831853071;
// Valid from 1000 to 40000 K (and additionally 0
for pure full white)
vec3 colorTemperatureToRGB(const in float
temperature){
 // Values from:
http://blenderartists.org/forum/showthread.php?
270332-OSL-
Goodness&p=2268693&viewfull=1#post2268693
```

```
 mat3 m = (temperature <= 6500.0) ?
mat3(vec3(0.0, -2902.1955373783176,
-8257.7997278925690),

vec3(0.0, 1669.5803561666639,
2575.2827530017594),

vec3(1.0, 1.3302673723350029,
1.8993753891711275)) :

mat3(vec3(1745.0425298314172, 1216.6168361476490,
-8257.7997278925690),

vec3(-2666.3474220535695, -2173.1012343082230,
2575.2827530017594),

vec3(0.55995389139931482, 0.70381203140554553,
1.8993753891711275));
 return mix(clamp(vec3(m[0] /
(vec3(clamp(temperature, 1000.0, 40000.0)) +
m[1]) + m[2]), vec3(0.0), vec3(1.0)), vec3(1.0),
smoothstep(1000.0, 0.0, temperature));
}
//@implements: sampler2D
struct ColorTemperature {
 sampler2D sampler;
 //@ label: "Mix[%]", editor: range, min: 0, max:
1, range_min: 0, range_max: 1, range_default: 0.5
 float mix;
 //@ label: "Kelvin Preset", editor: enum,
enum_default: 0, values: "No_Preset,
Candle_1800K, Sunrise_Sunset_2500K,
Tungsten_3000K, Morning_3500K, Moonlight_4000K,
Midday_5500K, Cloudy_6500K, Shade_7000K,
Clear_Sky_10000K"
    int idKelvinPreset;

 //@ label: "Kelvin", editor: range, min: 1000,
max: 10000, range_min: 1000, range_max: 10000,
range_default: 1000
 float kelvin;
 //@ label: "Brightness[%]", editor: range, min:
0, max: 1, range_min: 0, range_max: 1,
range_default: 1
 float luminancePreservationFactor;
};
vec4 texture(ColorTemperature s, vec2 tex_coords)
{
 vec2 iResolution = vec2(s.width, s.height);
 float temperature = (iMouse.z > 0.0) ?
mix(1000.0, 40000.0, iMouse.x / iResolution.x) :
6550.0; // mix(1000.0, 15000.0, (sin(iTime * (PI2
/ 10.0)) * 0.5) + 0.5);
 float temperatureStrength = (iMouse.z > 0.0) ?
(1.0 - clamp((iMouse.y / iResolution.y) * (1.0 /
0.9), 0.0, 1.0)) : 1.0;
 if(tex_coords.y > 0.1){
   vec3 inColor = texture(iChannel0,
tex_coords).xyz;
   vec3 outColor = mix(inColor, inColor *
colorTemperatureToRGB(temperature),
temperatureStrength);
   #ifdef WithQuickAndDirtyLuminancePreservation
     outColor *= mix(1.0, dot(inColor,
vec3(0.2126, 0.7152, 0.0722)) / max(dot(outColor,
vec3(0.2126, 0.7152, 0.0722)), 1e-5),
s.luminancePreservationFactor);
   #endif
   return vec4(outColor, 1.0);
 }else{
   vec2 f = vec2(1.5) / iResolution.xy;
      fragColor =
vec4(mix(colorTemperatureToRGB(mix(1000.0,
40000.0, tex_coords.x)), vec3(0.0),
min(min(smoothstep(tex_coords.x - f.x,
tex_coords.x, (temperature - 1000.0) / 39000.0),

smoothstep(tex_coords.x + f.x, tex_coords.x,
(temperature - 1000.0) / 39000.0)),

1.0 - min(smoothstep(0.04 - f.y, 0.04,
tex_coords.y),

smoothstep(0.06 + f.y, 0.06, tex_coords.y)))),

1.0);
```

```
  }
}
```

STEP 3: ADDING VARIABLES AND SPECIFIC CHANGES

Other Possible Native Shadertoy Variables

There are a few more Shadertoy variables native to the software which either should be deleted or transformed into variables inside the struct.

#define, #ifdef, #endif

These C definitions, although functional with Pixera, should be adapted as variables in the struct for better organization and usage in Pixera. Many times these statements are easily replaced with their respective type or/and if statements. Here is an example of a replacement:

Shadertoy	

```
#define
WithQuickAndDirtyLuminancePreservation

void main( out vec4 fragColor, in vec2
fragCoord) {

#ifdef
WithQuickAndDirtyLuminancePreservation

outColor *= mix(1.0, dot(inColor,
```
	```
	struct ColorTempera

	label: "Border is S
	bool_default: true

	withQuickAndDirtyLu


	texture(ColorTemper
	{
	```

```
vec3(0.2126, 0.7152, 0.0722)) /
max(dot(outColor, vec3(0.2126, 0.7152,
0.0722)), 1e-5),
s.luminancePreservationFactor);

#endif


return vec4(outColor, 1.0);

}
```

```
if(s.withQuickAndDi
{

outColor *= mix(1.0
vec3(0.2126, 0.7152
max(dot(outColor, v
0.0722)), 1e-5),
s.luminancePreserva


return vec4(outColo
```

Here the “#define” is replaced with a bool variable added to the struct. Looking at the struct we already created, we will be taking out the "WithQuickAndDirtyLuminancePreservation" completely due to the luminancePreservationFactor variable already in the struct. When it is set to 0 it performs the same function as the bool variable.

iChannel0 and iTime

These two variables are native to Shadertoy allowing inputs of media and tracking of time respectively. “iChannel"Index"” can be replaced by a sampler2D variable in the struct, most of the time if the Shadertoy references only one iChannel that means you will just use the original sampler variable. The iTime reference can be made into a slider variable and then modified during runtime.

In this case, we will be replacing the “iChannel0” with the s.sampler while deleting iTime completely due to the next change.

iMouse and Original Temperature Selection

In the Shadertoy implementation (Found Here) the shader allows the user to click and move their mouse across the image to change the temperature of the effect. In the case of Pixera, we want the effect to influence the entire screen, and the changing of data is only done in the timeline. Also, the use of iMouse is not possible in Pixera through the code as that is a variable native to Shadertoy, making the following lines unusable:

```
float temperature = (iMouse.z > 0.0) ?
mix(1000.0, 40000.0, iMouse.x / iResolution.x) :
6550.0; // mix(1000.0, 15000.0, (sin(iTime * (PI2
/ 10.0)) * 0.5) + 0.5);
float temperatureStrength = (iMouse.z > 0.0) ?
(1.0 - clamp((iMouse.y / iResolution.y) * (1.0 /
0.9), 0.0, 1.0)) : 1.0;
```

Luckily the kelvin variable created in the struct and “mix()” method native to GLSL can take the place of these lines. The user will be able to modify the kelvin value from the timeline instead of using the mouse. This is the same for the temperatureStrength as the mix variable takes its place directly.

Alongside the iMouse variable is an if statement in the code which cuts the bottom of the media off allowing the slider to be visible. This if statement and the else section can be deleted as it only works with the iMouse variable.

After making these specific changes to the code large chunks of the program should have been deleted and the code should look like this:

```
// Color temperature (sRGB) stuff
// Copyright (C) 2014 by Benjamin 'BeRo' Rosseaux
// Because the german law knows no public domain
in the usual sense,
// this code is licensed under the CC0 license
//
http://creativecommons.org/publicdomain/zero/1.0/
#define WithQuickAndDirtyLuminancePreservation
const float LuminancePreservationFactor = 1.0;
const float PI2 = 6.2831853071;
```

```
// Valid from 1000 to 40000 K (and additionally 0
for pure full white)
vec3 colorTemperatureToRGB(const in float
temperature){
 // Values from:
http://blenderartists.org/forum/showthread.php?
270332-OSL-
Goodness&p=2268693&viewfull=1#post2268693
 mat3 m = (temperature <= 6500.0) ?
mat3(vec3(0.0, -2902.1955373783176,
-8257.7997278925690),

vec3(0.0, 1669.5803561666639,
2575.2827530017594),

vec3(1.0, 1.3302673723350029,
1.8993753891711275)) :

mat3(vec3(1745.0425298314172, 1216.6168361476490,
-8257.7997278925690),

vec3(-2666.3474220535695, -2173.1012343082230,
2575.2827530017594),

vec3(0.55995389139931482, 0.70381203140554553,
1.8993753891711275));
 return mix(clamp(vec3(m[0] /
(vec3(clamp(temperature, 1000.0, 40000.0)) +
m[1]) + m[2]), vec3(0.0), vec3(1.0)), vec3(1.0),
smoothstep(1000.0, 0.0, temperature));
}
//@implements: sampler2D
struct ColorTemperature {
 sampler2D sampler;
 //@ label: "Mix[%]", editor: range, min: 0, max:
1, range_min: 0, range_max: 1, range_default: 0.5
 float mix;
 //@ label: "Kelvin Preset", editor: enum,
enum_default: 0, values: "No_Preset,
Candle_1800K, Sunrise_Sunset_2500K,
Tungsten_3000K, Morning_3500K, Moonlight_4000K,
Midday_5500K, Cloudy_6500K, Shade_7000K,
Clear_Sky_10000K"
```

```
    int idKelvinPreset;

 //@ label: "Kelvin", editor: range, min: 1000,
max: 10000, range_min: 1000, range_max: 10000,
range_default: 1000
 float kelvin;
 //@ label: "Brightness[%]", editor: range, min:
0, max: 1, range_min: 0, range_max: 1,
range_default: 1
 float luminancePreservationFactor;
};
vec4 texture(ColorTemperature s, vec2 tex_coords)
{
 vec3 inColor = texture(s.sampler,
tex_coords).xyz;

 vec3 outColor = mix(inColor, inColor *
colorTemperatureToRGB(s.kelvin), s.mix);
 outColor *= mix(1.0, dot(inColor, vec3(0.2126,
0.7152, 0.0722)) / max(dot(outColor, vec3(0.2126,
0.7152, 0.0722)), 1e-5),
s.luminancePreservationFactor);
 return vec4(outColor, 1.0);
}
```

STEP 4: FINAL TOUCHES AND IMPROVEMENTS

enum Usage for Preset effects

Before being finished there is one more struct variable mentioned that is not implemented yet, that being “idKelvinPreset”.

```
 //@ label: "Kelvin Preset", editor: enum,
enum_default: 0, values: "No_Preset,
Candle_1800K, Sunrise_Sunset_2500K,
Tungsten_3000K, Morning_3500K, Moonlight_4000K,
Midday_5500K, Cloudy_6500K, Shade_7000K,
Clear_Sky_10000K"
int idKelvinPreset;
```

Adding an enum dropdown allowing the user to pick between premade versions of the effect is useful for quick changes and examples of possible uses of the effect.

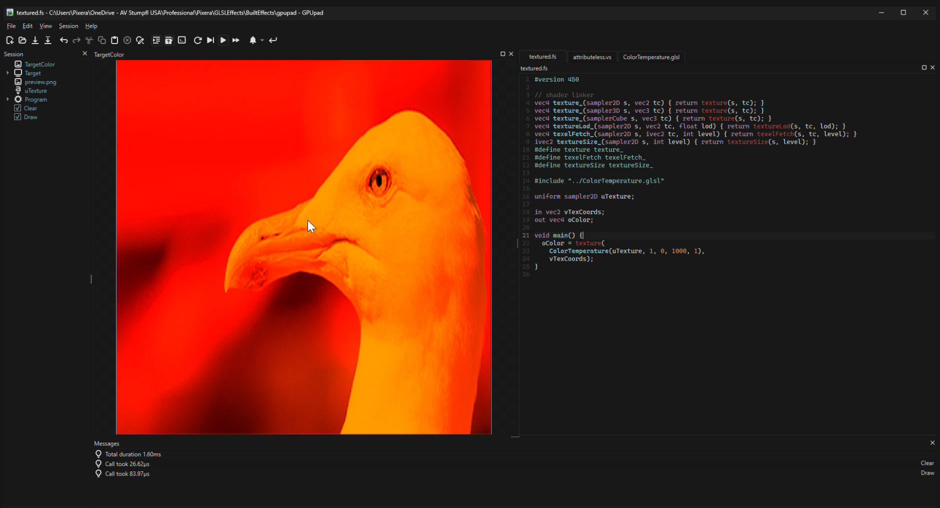
In this case, the "idKelvinPreset" gives multiple preset kelvin values describing the similar light type involved. When selected in the dropdown, that preset takes over the effect making the kelvin slider inactive. This is why a “No_Preset” option is available so the user can go back to manual control.

As described earlier, the enums are referenced into the code as index values, allowing a switch statement to easily change what path the code goes down depending on what selection is made.

After these final additions, the code should look like this:

```
 // Color temperature (sRGB) stuff
// Copyright (C) 2014 by Benjamin 'BeRo' Rosseaux
// Because the german law knows no public domain
in the usual sense,
// this code is licensed under the CC0 license
//
http://creativecommons.org/publicdomain/zero/1.0/
const float LuminancePreservationFactor = 1.0;
const float PI2 = 6.2831853071;
// Valid from 1000 to 40000 K (and additionally 0
for pure full white)
vec3 colorTemperatureToRGB(const in float
temperature){
 // Values from:
http://blenderartists.org/forum/showthread.php?
270332-OSL-
Goodness&p=2268693&viewfull=1#post2268693
 mat3 m = (temperature <= 6500.0) ?
mat3(vec3(0.0, -2902.1955373783176,
-8257.7997278925690),
      vec3(0.0, 1669.5803561666639,
2575.2827530017594),
      vec3(1.0, 1.3302673723350029,
1.8993753891711275)) :
         mat3(vec3(1745.0425298314172,
1216.6168361476490, -8257.7997278925690),
      vec3(-2666.3474220535695,
-2173.1012343082230, 2575.2827530017594),
      vec3(0.55995389139931482,
0.70381203140554553, 1.8993753891711275));
 return mix(clamp(vec3(m[0] /
(vec3(clamp(temperature, 1000.0, 40000.0)) +
m[1]) + m[2]), vec3(0.0), vec3(1.0)), vec3(1.0),
smoothstep(1000.0, 0.0, temperature));
}
//@implements: sampler2D
struct ColorTemperature {
 sampler2D sampler;
 //@ label: "Mix[%]", editor: range, min: 0, max:
1, range_min: 0, range_max: 1, range_default: 0.5
 float mix;
 //@ label: "Kelvin Preset", editor: enum,
enum_default: 0, values: "No_Preset,
Candle_1800K, Sunrise_Sunset_2500K,
Tungsten_3000K, Morning_3500K, Moonlight_4000K,
Midday_5500K, Cloudy_6500K, Shade_7000K,
Clear_Sky_10000K"
    int idKelvinPreset;

 //@ label: "Kelvin", editor: range, min: 1000,
max: 10000, range_min: 1000, range_max: 10000,
range_default: 1000
 float kelvin;
 //@ label: "Brightness[%]", editor: range, min:
0, max: 1, range_min: 0, range_max: 1,
range_default: 1
 float luminancePreservationFactor;
};
vec4 texture(ColorTemperature s, vec2 tex_coords)
{
 vec3 inColor = texture(s.sampler,
tex_coords).xyz;
 vec3 outColor = mix(inColor, inColor *
colorTemperatureToRGB(s.kelvin), s.mix);
 switch(s.idKelvinPreset) {
 case 0:
   break;
 case 1:
   outColor = mix(inColor, inColor *
colorTemperatureToRGB(1800.0), s.mix);
   break;
 case 2:
   outColor = mix(inColor, inColor *
colorTemperatureToRGB(2500.0), s.mix);
   break;
 case 3:
   outColor = mix(inColor, inColor *
colorTemperatureToRGB(3000.0), s.mix);
   break;
 case 4:
   outColor = mix(inColor, inColor *
colorTemperatureToRGB(3500.0), s.mix);
   break;
 case 5:
   outColor = mix(inColor, inColor *
colorTemperatureToRGB(4000.0), s.mix);
   break;
 case 6:
   outColor = mix(inColor, inColor *
colorTemperatureToRGB(5500.0), s.mix);
   break;
 case 7:
   outColor = mix(inColor, inColor *
colorTemperatureToRGB(6500.0), s.mix);
   break;
 case 8:
   outColor = mix(inColor, inColor *
colorTemperatureToRGB(7000.0), s.mix);
   break;
 case 9:
   outColor = mix(inColor, inColor *
colorTemperatureToRGB(10000.0), s.mix);
   break;
 default:
   outColor = vec3(1.0);
   break;
 }
 outColor *= mix(1.0, dot(inColor, vec3(0.2126,
0.7152, 0.0722)) / max(dot(outColor, vec3(0.2126,
0.7152, 0.0722)), 1e-5),
s.luminancePreservationFactor);
 return vec4(outColor, 1.0);
}
```

This should be the visual in GPUpad once tested with the given values:



```
#version 450

// shader linker
vec4 texture_(sampler2D s, vec2 tc) { return texture(s, tc); }
vec4 texture_(sampler3D s, vec3 tc) { return texture(s, tc); }
vec4 texture_(samplerCube s, vec3 tc) { return texture(s, tc); }
vec4 textureLod_(sampler2D s, vec2 tc, float lod) { return textureLod(s, tc, lod); }
vec4 texelFetch_(sampler2D s, ivec2 tc, int level) { return texelFetch(s, tc, level); }
ivec2 textureSize_(sampler2D s, int level) { return textureSize(s, level); }
#define texture texture_
#define texelFetch texelFetch_
#define textureSize textureSize_

#include "../ColorTemperature.glsl"

uniform sampler2D uTexture;

in vec2 vTexCoords;
out vec4 oColor;

void main() {
  oColor = texture(
    ColorTemperature(uTexture, 1, 0, 1000, 1),
    vTexCoords);
}
```

```
Messages
Total duration 1.60ms
Call took 26.62µs
Call took 83.97µs
```

STEP 5: PIXERA USE AND TROUBLESHOOTING

Adding Into Pixera

After dragging the GLSL file into Pixera an effect will automatically be made and can be attached to a layer. From here a few possibilities will occur.

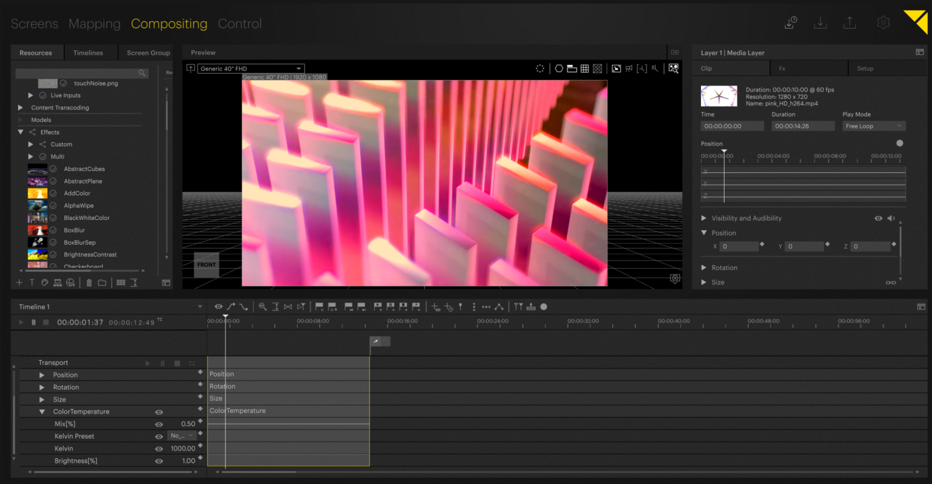
1. The effect works without an issue and all the struct variables show in the timeline, changing the effect as they are changed.
2. The effect appears to show up okay in the timeline however the media on the screen stutters.
3. The effect appears to show up okay in the timeline however the media on the screen goes black.
4. The effect does not show the struct variables in the timeline and causes one of the two screen issues described above.

In the case that 2-4 occur, it is because the GLSL shader code has an error within. This error could either be an error with the GLSL code or with how Pixera requires its code to be formatted. Here are the most common errors leading to these problems:

1. Using the struct variable name instead of s."Variable Name":
    a. float num = thisNumber; → float num = s.thisNumber;
2. Misspellings, especially in the struct variable comments.
3. Forgetting semicolons, commas, or quotes and using extra parentheses in code and comments:
    a. //@ label: Mix[%]", editor: range min0, max: 1, range_min: 0, ranmax 1, range_default: 0.5 → //@ label: "Mix[%]", editor: range, min: 0, max: 1, range_min: 0, range_max: 1, range_default: 0.5
    b. float num = 0 → float num = 0;

Continuously check the code, comment out sections, and take time to find these errors until the effect runs correctly.

This should be the visual in Pixera once tested with the given values:



Timeline 1
00:00:01:37 00:00:12:49
Transport
Position
Rotation
Size
ColorTemperature
Mix[%] 0.50
Kelvin Preset No_..
Kelvin 1000.00
Brightness[%] 1.00

See Also

- Effects
- Effect Guide
- Managing Effects
- GLSL Shader for Pixera

Pixera 1.9.139 | 25. September 2023

Was this article helpful?