

原


OpenGL（十七）Photoshop blend算法 与 图层混合模式

2018年02月10日 10:47:58

松阳

阅读数：1270

更多

 版权声明：本文为 松阳 (blog.csdn.net/fansongy) 原创文章，转载必须注明出处: <https://blog.csdn.net/fansongy/article/details/79303646>

使用混合模式可以制作丰富多彩的效果。而OpenGL中可以轻松开启这种模式， 但更关键的是图形算法。本文参照 Photoshop blend算法 与 如何通 OpenGL中实现混合效果。

OpenGL中开启混合

在OpenGL中可以开启混合模式：

```
1 glEnable( GL_BLEND ); // 启用混合
2 glDisable( GL_BLEND ); // 禁用关闭混合
```

统一说法，参照Photoshop，下方图层、 baseColor 为原色。上方图层、 blendColor 为混合色。下面介绍 Photoshop blend算法 的详细实现。

公式汇总

先上一张公式汇总图，图片来自网络，出于效率和效果的平衡，有些公式跟下文提到的有些出入。

变暗： $C = \min(A, B)$

变亮： $C = \max(A, B)$

正片叠底： $C = \frac{A \times B}{255}$

滤色： $C = 255 - \frac{A \text{反相} \times B \text{反相}}{255}$

颜色加深： $C = A - \frac{A \text{反相} \times B \text{反相}}{B}$

颜色减淡： $C = A + \frac{A \times B}{B \text{反相}}$

线性加深： $C = A + B - 255$

线性减淡： $C = A + B$

叠加：当 $A \leq 128$ 时， $C = \frac{A \times B}{128}$

当 $A > 128$ 时， $C = 255 - \frac{A \text{反相} \times B \text{反相}}{128}$

强光：当 $B \leq 128$ 时， $C = \frac{A \times B}{128}$

当 $B > 128$ 时， $C = 255 - \frac{A \text{反相} \times B \text{反相}}{128}$

柔光：当 $B \leq 128$ 时， $C = \frac{A \times B}{128} + (\frac{A}{255})^2 \times (255 - 2B)$

当 $B > 128$ 时， $C = \frac{A \times B \text{反相}}{128} + \sqrt{\frac{A}{255}} \times (2B - 255)$

亮光：当 $B \leq 128$ 时， $C = A - \frac{A \text{反相} \times (255 - 2B)}{2B}$

当 $B > 128$ 时， $C = A + \frac{A \times (2B - 255)}{2 \times B \text{反相}}$

点光：当 $B \leq 128$ 时， $C = \min(A, 2B)$

当 $B > 128$ 时， $C = \min(A, 2B - 255)$

线性光： $C = A + 2B - 255$

实色混合：当 $A + B \geq 255$ 时， $C = 255$ ，否则为0

排除： $C = A + B - \frac{A \times B}{128}$

差值： $C = |A - B|$

相加： $C = \frac{A + B}{\text{收缩}} + \text{补偿值}$

减去： $C = \frac{A - B}{\text{收缩}} + \text{补偿值}$

注释：

1. 混合模式的数学计算公式，另外还介绍了不透明度。

2. 这些公式仅适用于RGB图像，对于Lab颜色图像而言，这些公式将不再适用。

3. 在公式中

A 代表下面图层的颜色值；

B 代表上面图层的颜色值；

C 代表混合图层的颜色值；

d 表示该层的透明度

正常混合

正常 Opacity

使上方图层完全遮住下方图层。

```
gl_FragColor = baseColor * (1.0-blendColor.a) + blendColor * blendColor.a;
```

使结果更暗

整个变暗块都是通过混合色的叠加，使结果更暗。常用的是变亮变暗，程度正常，正片叠底及滤色，较为柔和。

变暗 Darken

两个图层中较暗的颜色将作为混合的颜色保留，比混合色亮的像素将被替换，而比混合色暗像素保持不变。

```
gl_FragColor = min(baseColor,blendColor);
```

正片叠底 Multiply

整体效果显示由上方图层和下方图层的像素值中较暗的像素合成的图像效果，任意颜色与黑色重叠时将产生黑色，任意颜色和白色重叠时则保持不

```
gl_FragColor = baseColor*blendColor;
```

颜色加深 Color Burn

选择该项将降低上方图层中除黑色外的其他区域的对比度，使图像的对比度下降，产生下方图层透过上方图层的投影效果。

```
gl_FragColor = vec4(1.0) - (vec4(1.0)-baseColor)/blendColor);
```

线性加深 Linear Burn

上方图层将根据下方图层的灰度与图像融合，此模式对白色无效。

```
gl_FragColor = baseColor+blendColor-vec4(1.0);
```

使结果更亮

变亮 Lighten

使上方图层的暗调区域变为透明，通过下方的较亮区域使图像更亮。

```
gl_FragColor = max(baseColor,blendColor);
```

滤色 Screen

该项与“正片叠底”的效果相反，在整体效果上显示由上方图层和下方图层的像素值中较亮的像素合成的效果，得到的图像是一种漂白图像中颜色的效果。

```
gl_FragColor = vec4(1.0) - ((vec4(1.0)-baseColor)*(vec4(1.0)-blendColor));
```

颜色减淡 Color Dodge

和“颜色加深”效果相反，“颜色减淡”是由上方图层根据下方图层灰阶程序提升亮度，然后再与下方图层融合，此模式通常可以用来创建光源中心点极亮的

```
gl_FragColor = baseColor/(vec4(1.0)-blendColor);
```

线性减淡 Linear Dodge

根据每一个颜色通道的颜色信息，加亮所有通道的基色，并通过降低其他颜色的亮度来反映混合颜色，此模式对黑色无效。

```
gl_FragColor = baseColor+blendColor;
```



## 溶合颜色

叠加柔光这块是使50%灰度上的颜色更亮，50%灰度下的颜色更暗，主要用于增加对比度的，常用的是叠加和柔光，柔光更柔和。

### 叠加 Overlay

此项的图像最终效果最终取决于下方图层，上方图层的高光区域和暗调将不变，只是混合了中间调。

```
1  vec4 lumCoeff=vec4(0.2125,0.7154,0.0721,1.0);
2  float luminance = dot(baseColor.rgb,lumCoeff.rgb);
3  if(luminance < 0.45)
4  {
5      gl_FragColor = 2.0 *baseColor * blendColor;
6  }
7  else if(luminance >0.55)
8  {
9      gl_FragColor = vec4(1.0)-2.0* ((vec4(1.0)-baseColor)*(vec4(1.0)-blendColor));
10 }
11 else
12 {
13     vec4 colorT1 = 2.0 *baseColor * blendColor;
14     vec4 colorT2 = vec4(1.0)-2.0* ((vec4(1.0)-baseColor)*(vec4(1.0)-blendColor));
15     gl_FragColor = mix(baseColor,blendColor,(luminance-0.45)*10);
16 }
```

### 柔光 Soft Light

使颜色变亮或变暗让图像具有非常柔和的效果，亮于中性灰底的区域将更亮，暗于中性灰底的区域将更暗。

```
gl_FragColor = 2.0 * baseColor * blendColor + baseColor*baseColor -2.0*baseColor*baseColor*blendColor;
```

### 强光 Hard Light

此项和“柔光”的效果类似，但其程序远远大于“柔光”效果，适用于图像增加强光照射效果。

```
1  vec4 lumCoeff=vec4(0.2125,0.7154,0.0721,1.0);
2  float luminance = dot(blendColor.rgb,lumCoeff.rgb);
3  if(luminance < 0.45)
4  {
5      gl_FragColor = 2.0 *baseColor * blendColor;
6  }
7  else if(luminance >0.55)
8  {
9      gl_FragColor = vec4(1.0)-2.0* ((vec4(1.0)-baseColor)*(vec4(1.0)-blendColor));
10 }
11 else
12 {
13     vec4 colorT1 = 2.0 *baseColor * blendColor;
14     vec4 colorT2 = vec4(1.0)-2.0* ((vec4(1.0)-baseColor)*(vec4(1.0)-blendColor));
15     gl_FragColor = mix(baseColor,blendColor,(luminance-0.45)*10);
16 }
```

### 亮光 Vivid Light

根据融合颜色的灰度减少对比度，可以使图像更亮或更暗。

```
gl_FragColor = baseColor + baseColor * (2*blendColor - vec4(1.0)) / (2*(vec4(1.0)-blendColor));
```

### 线性光 Linear Light

根据混合颜色的灰度，来减少或增加图像亮度，使图像更亮。

```
gl_FragColor = baseColor + 2 * blendColor - vec4(1.0);
```

点光 Pin Light

如果混合色比50%灰度色亮，则将替换混合色暗的像素，而不改变混合色亮的像素;反之如果混合色比50%灰度色暗，则将替换混合色亮的像素，而不改变的像素。

```
gl_FragColor = min(baseColor,2*blendColor - vec4(1.0));
```

色差颜色

差值 Difference

上方图层的亮区将下方图层的颜色进行反相，暗区则将颜色正常显示出来，效果与原图像是完全相反的颜色。

```
gl_FragColor = vec4(abs(blendColor-baseColor).rgb,1.0);
```

排除 Exclusion

创建一种与“差值”模式类似但对比度更低的效果。3lian.com,与白色混合将反转基色值，与黑色混合则不发生变化。

```
gl_FragColor =vec4((baseColor + blendColor).rgb - (2.0*baseColor*blendColor).rgb,1.0);
```

减去 Subtract

```
gl_FragColor = vec4(baseColor.rgb-blendColor.rgb,1.0);
```

划分 Divide

```
gl_FragColor = baseColor/blendColor;
```

单图处理

平滑

与高斯模糊相同，算子都为1

```
1 float kernel[9];
2 kernel[6]=1;kernel[7]=1;kernel[8]=1;
3 kernel[3]=1;kernel[4]=1;kernel[5]=1;
4 kernel[0]=1;kernel[1]=1;kernel[2]=1;
5 int index=0;
6 for(int y=0;y<coreSize;y++)
7 {
8     for(int x=0;x<coreSize;x++)
9     {
10         vec4 currentColor=texture2D(U_MainTexture,V_Texcoord+vec2((-1+x)*texel0ffset,(-1+y)*texel0ffset));
11         color+=currentColor*kernel[index++];
12     }
13 }
14 color/=9.0;
15 gl_FragColor=color;
```

👍  
0

💬

📖

🔖

📱

<

>

锐化

同上面的平滑效果，其算子为

```
1 kernel[6]=0;kernel[7]=-1;kernel[8]=0;
2 kernel[3]=-1;kernel[4]=4;kernel[5]=-1;
3 kernel[0]=0;kernel[1]=-1;kernel[2]=0;
4 // 如果不明显
5 gl_FragColor = baseColor + color;
```

边缘检测

同上面的平滑效果，其算子为

```
1 kernel[6]=0;kernel[7]=1;kernel[8]=0;
2 kernel[3]=1;kernel[4]=-4;kernel[5]=1;
3 kernel[0]=0;kernel[1]=1;kernel[2]=0;
4 // 如果不明显
5 gl_FragColor = baseColor + color;
```

总结

本文介绍了 Photoshop blend算法 的OpenGL实现方式。主要是编写各种Shader。在实际使用的过程中，不一定严格按照公式来编写，只要效果看着正用，有时候没准还会获得出乎意料的效果呢。

 想对作者说点什么

Photoshop图层混合(Layer Blending)模式的算法实现

阅读数 1007

转自：<http://avnpc.com/pages/photoshop-layer-blending-algorithm>Photoshop的图层混合(LayerBlending)是实现各种...

博文 来自：[zylxadz的专栏](#)

Photoshop图层混合模式的计算公式

阅读数 907

转自<http://blog.csdn.net/pizi0475/article/details/8241774>PS和Nuke的叠加模式计算算法相差甚远，最近想在Nuke中...

博文 来自：[司马懿的西山居](#)

Photoshop图层混合模式计算公式大全

阅读数 3325

Photoshop图层混合模式计算公式大全关于photoshop的图层混合模式，大家一定都非常熟悉了，我在这里把各种混...

博文 来自：[隔壁老王的博客](#)

OpenGL实现多层绘制（Layered Rendering）

阅读数 4121

多层绘制可以一次性渲染多张纹理，还能实现一些更为方便的功能，比如只调用一次shader就完成多个不同视口图像...

博文 来自：[鸣桐小记](#)

OPENGL如何实现绘制多个图层，可实现图层叠加或单独显示？

软件左边导航栏，有多个图层，是否可以选择显示某一个图层或某几个图层？ 有高手晓得不？

论坛

想问问大家opengl的图层怎么做？

我想做一个类似gps定位导航仪之类的软件，但是我的监测范围比较小，仅用于奥运场馆。我们在场馆内所有的移动工具（...

论坛

贴图层OpenGL——glUniform1i 用法

阅读数 2549

[cpp] 1. 为了使用第二个纹理（以及第一个），我们必须改变一点渲染流程，先绑定两个纹理到对应的纹理单元，然...

博文 来自：[linuxheik的专栏](#)

glsl颜色混合算法

阅读数 833

multiply:result=base\*blend;screen:result=vec4(1.0)-((vec4(1.0)-blend)\*(vec4(1.0)-base));darken:resul...

博文 来自：[yvhkyiu的专栏](#)

GLSL/C++ 实现滤镜效果

阅读数 4001

glsl实现滤镜效果

博文 来自：[吹来的寒风](#)

背景图片和颜色混合叠加多种混合模式

阅读数 456

//同时设置背景图片和颜色background:#5bd5a0url(&quot;su1.jpg&quot;);//背景混合模式background-blend...

博文 来自：[虫\\_X的学习经历分享](#)