VCB-Studio 教程 16　Repair 的用法与 Contra-Sharp

本教程旨在讲述 Repair 的用法和简单扩展。

1. Repair 的原理

Repair 是根据 RemoveGrain 改变而来的，用法是 Repair(filtered, source, mode=1)。

RemoveGrain 总体的做法可以总结为：把中心像素，跟周边 8 个像素比较，让中心像素不要过高或者过低。比如说 RemoveGrain(src, mode=1)，保证输出的每个像素，范围不超过其周边 8 个像素中最小值和最大值。

Repair 的用法很类似，但是表现为，把 flt 的每个像素，跟 src 中对应部分的 9 个像素比较，让 flt 中的像素不要过高或者过低。flt 通常为经过某种处理，可能引入突兀瑕疵的 clip，而 src 是源。

比如说默认的 Repair(flt,src,1)，会讲 flt 的每一个像素，跟 src 对应位置 3x3 像素比较。如果 flt 不超过 src 中 9 个像素的最大值或者最小值，那么不做处理，否则，会把 flt 的像素，替换为 9 个数中最大值或者最小值（取决于哪个更接近）。比如说 9 个数最小值是 28，最大值是 176，那么 flt 像素小于 28 的时候，会被改为 28；在 28-176 之间则不变，大于 176 则会被改为 176。

这是一种限制机制，保证 flt 的画面，跟 src 相比，不能太极端(跟源的领域比，出现过高或者过低的值)。所以 Repair 通常作为一种保护机制，任何可能引入突兀瑕疵的操作，可以用 Repair 做限制。

拓展开，Repair(flt, src, mode=k)，k=1/2/3/4 的作用是 flt 的每一个像素，跟 src 对应位置 3x3 的比较，如果 flt 不超过第 k 大或者第 k 小的，那么保留；否则替换为第 k 大或者第 k 小的。k 越大，越可能被限制，限制力度更高。

那么如果 flt=src，执行 Repair(flt,src,k)会怎样？不失一般性，不妨假设中心像素是偏大的。执行 Repair(flt,src,1)是没有意义的；因为这时候 flt=src, flt 的每个像素，等于 src 的 3x3 中的中心像素。flt 的任何像素永远不会被判定为过大或者过小；所以结果就是 flt/src。

如果 src 的中心像素，在 3x3 的邻域内正好是第 k 大，那么不做调整，这时候比它大的有 k-1 个，这 k-1 个一定都在周边 8 个数字中，相当于中心像素不超过周边 8 个像素中第 k-1 大的；

如果 src 的中心像素，在 3x3 的邻域内是 1...k-1 大，那么需要调整到包括它自己在内，第 k 大的，或者说是周边 8 个像素中 k-1 大的，相当于中心像素被强制限制为周边 8 个像素中 k-1 大的。

这不偏不倚就是 RemoveGrain(src,k-1)。

mode=k=1...4 的时候，flt 中心像素被限制在 k^{th}-Min 和 k^{th}-Max 之间。如果 mode=10+k=11..14，逻辑是 flt 的中心像素被限制在 MIN(k^{th}-Min, src 的中心像素)和 MAX(k^{th}-MAX, src 的中心像素)之间。我们来分析下：

如果 k=1，MIN(1^{st}-Min, src 的中心像素)= 1^{st}-Min，MAX(1^{th}-MAX, src 的中心像素)= 1^{th}-MAX，所以 Repair(mode=11)和 Repair(mode=1)是没有区别的；

如果 k=2，MIN(2^{st}-Min, src 的中心像素)就不一定等于 2^{st}-Min 了。有可能 src 的中心像素就是 9 个像素中最小的，这时候，Repair(mode=12)和 Repair(mode=2)相比，上下限有可能更宽，当且仅当 src 的中心像素是最大/最小值。所以 k=2/3/4 的时候，如果 src 的中心像素本身很大或者很小，Repair(mode=10+k)比起 Repair(mode=k)的限制力度是更弱的。

2. Repair 的常见用法

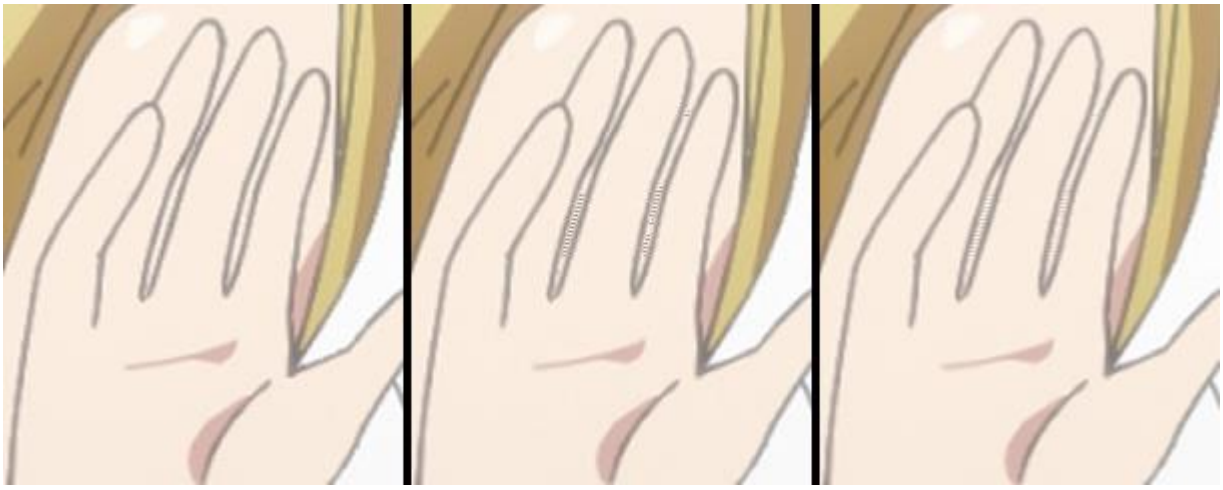
Repair 被广泛用在可能引入突兀瑕疵的地方，比如说时域降噪，可能引入 blending，把前后帧的信息加入本帧。这时候就可以用 Repair 来处理：

```
nr = src.MCTD()

Repair(nr, src, 2)
```

类似 GSMC 等时域降噪滤镜中，就有自带 Repair 的步骤。

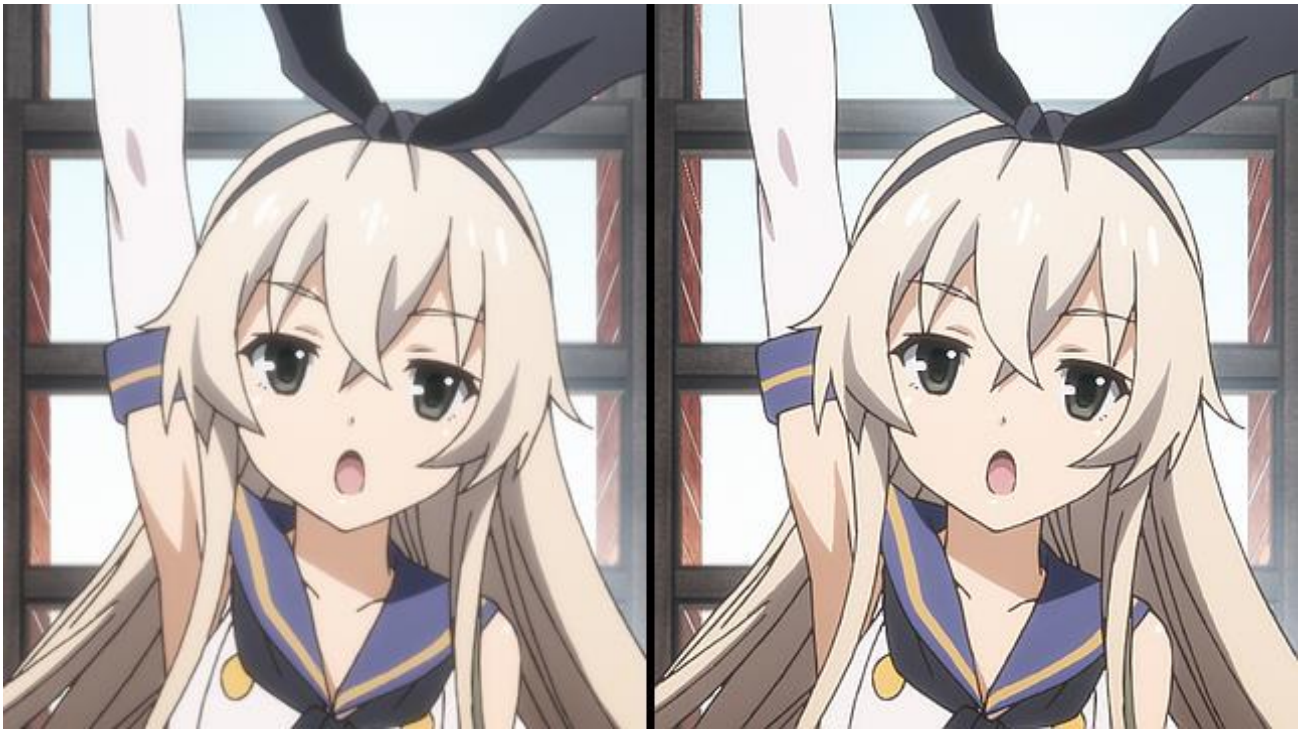
又比如，SangNom 做 AA/Deint 很可能引入瑕疵（左 vs 中），这时候通过 Repair(aa, src, 3)则可以很好的限制这种瑕疵的强度：



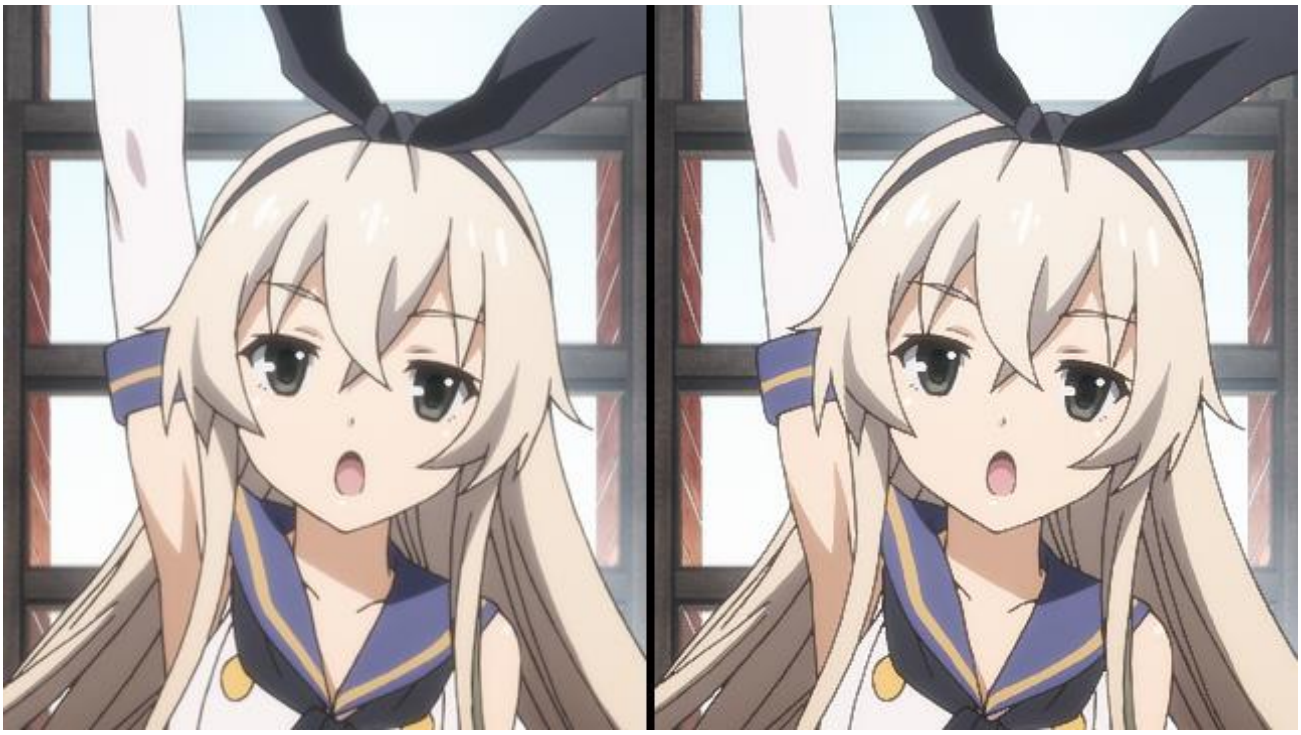
TAA 的后处理中也有用 Repair 来修复瑕疵的做法。

3. Moderate Sharpening

之前咱们说了 Unsharp Mask 用来锐化：



这副效果有点太过分了。这时候我们就可以考虑用 Repair(sharped, src, 1)来试着限制一下：



虽然不至于完全把瑕疵清理干净，但是比起不带限制，ringing 应该基本全部去掉了；只剩下 aliasing 比较明显。

这就是 RemoveGrain 的 doc 中说的 Moderate Sharpening，虽然还是很简单粗暴效果有待提高，但是起码提供了一个思路。

4. RGDering

上文中，我们从一个有 ringing 图像(sharp)，和一个没有 ringing，但是清晰度低的图像(src)，运算出一个锐利度高但是没有 ringing 的图像(moderate sharpening)

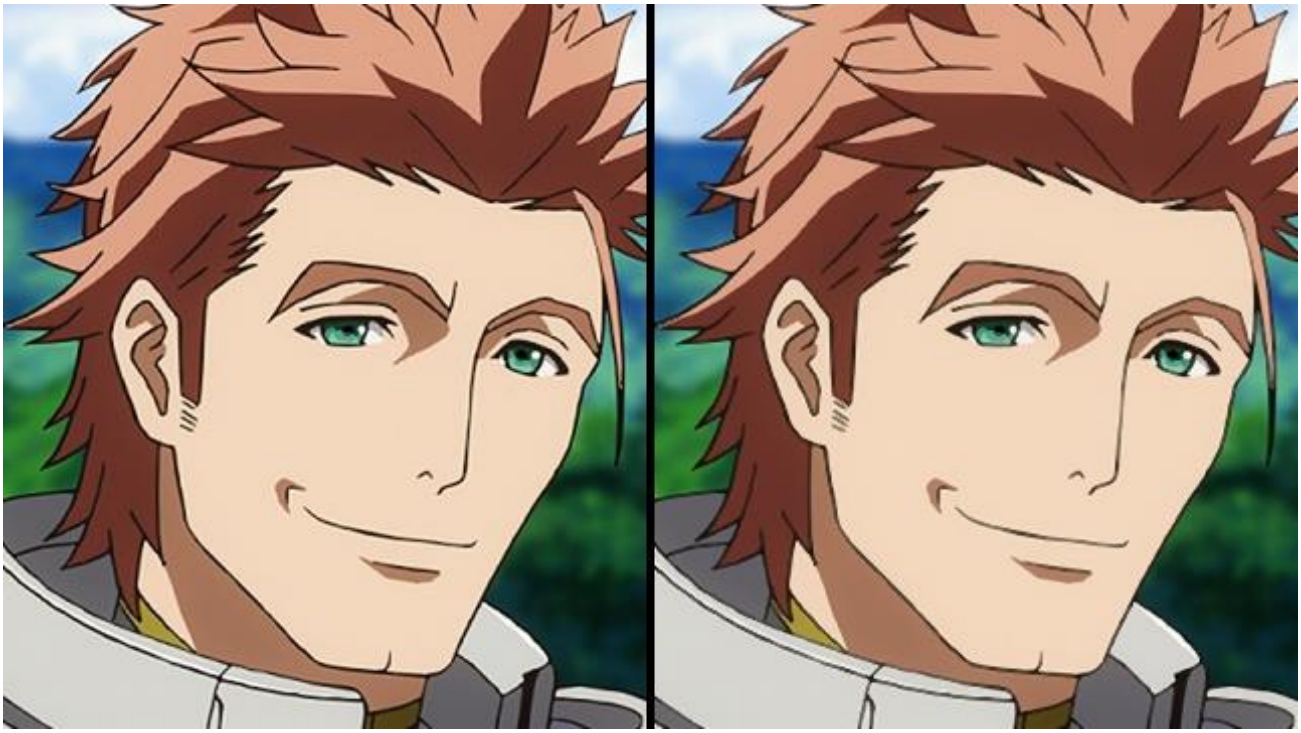
写成函数就是：mdrt_sharp = dering(sharp,src)

如果我们把第一个输入换成 src，第二个输入换成 src.blur 呢？

blur = src.Minblur(r=1)

Repair(src, blur, mode=1)

这就是 dering 的基本手段 :RGDering。它是 removegrain 的 doc 里自带的。注意它的用法是把 src 放在第一位，blur 放在第二位。用于一个强 ringing 的图像（左）效果如下（右）:



效果还是很明显，虽然伴随着部分的线条虚化和 residual ringing 还是难以避免，不过这比直接轰 blur 破坏力小的多了。

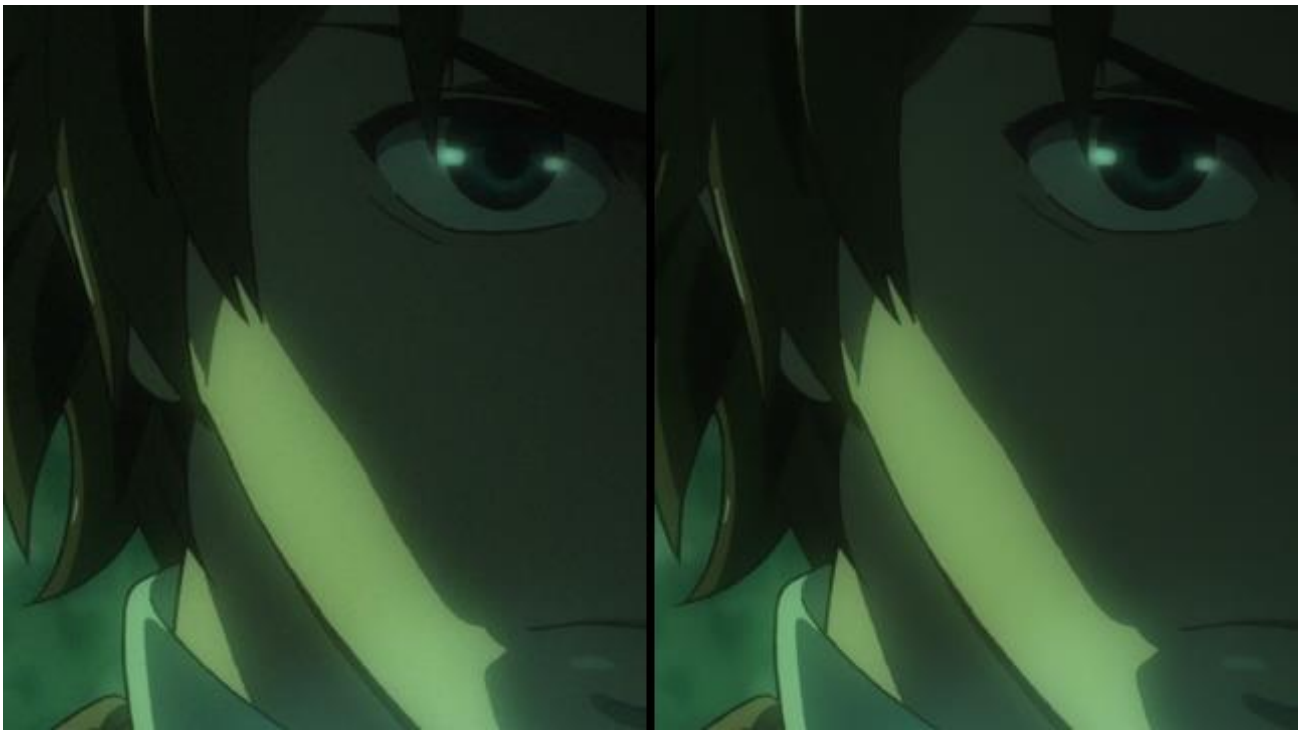
这也解释了为啥之前 Resizer(1)教程中的 non-ringing resizer 用法：

```
upscaled=core.fmtc.resample(src720p, 1920, 1080, kernel="lanczos",taps=4)
upscaled=core.rgvs.Repair(upscaled, core.fmtc.resample(src720p, 1920, 1080, kernel="gauss",a1=100),1)
```

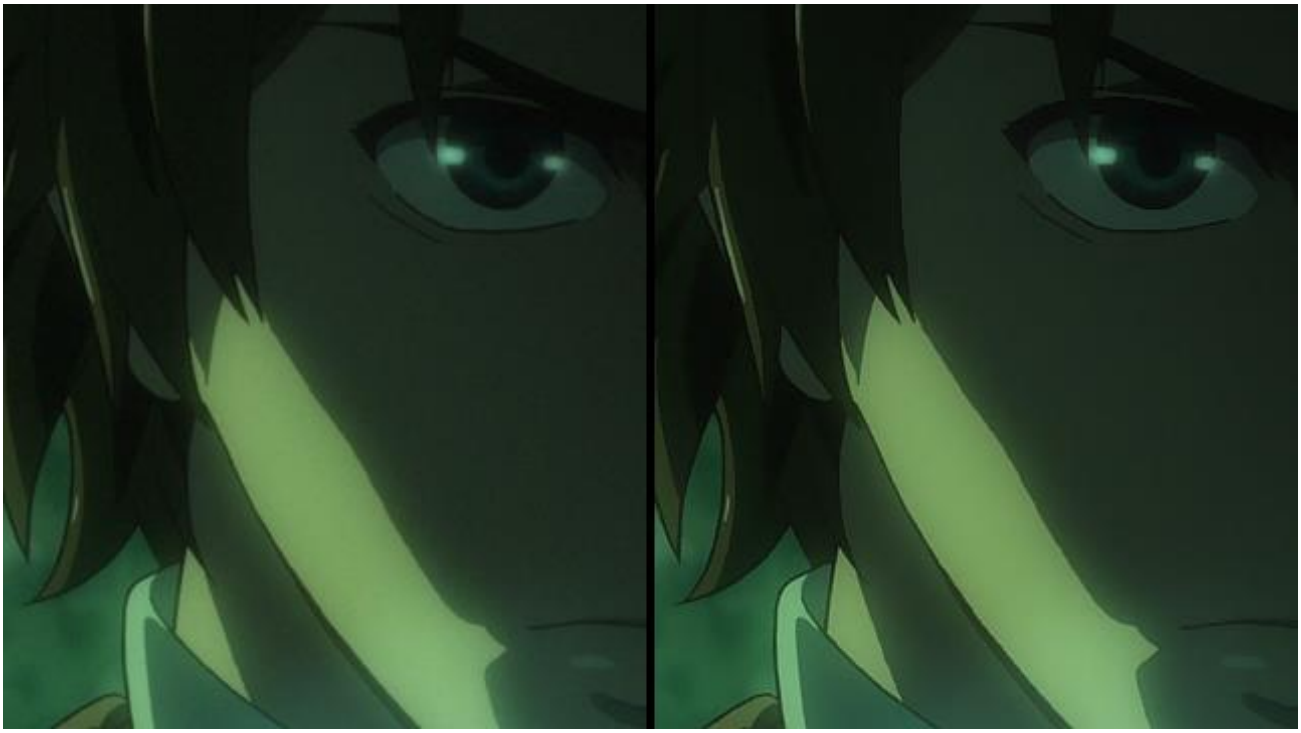
本质上是用 GaussResize 做一个理论上不会出 ringing 的 clip，然后用 Repair 去做一个 RGDering。在实际使用中，可以通过调节 a1(p)的值，这个值越小，GaussResize 出来的结果越柔和，dering 的强度越高。

5. Contra-Sharp/补偿性锐化

contrasharp 是很常用的弥补手段，常常用于 AA/deband/降噪/dering 等柔化处理后，来补偿画面的锐度，强化被削弱的细节。比如下图是用 Bilateral 降噪的结果(core.bilateral.Bilateral(src16, sigmaS=1.5,sigmaR=0.015, algorithm=2))：



确实很有效的去掉了噪点，但是画面也被一定程度的柔化，一些弱线条（比如头发）损失较大。我们考虑对降噪后的画面做一个 unsharpmask(11):



嗯……虽然画面锐利度回来了，柔化的线条救回来不少，但是，太锐利了，已经出现了过度锐化的痕迹。我们想让锐化后的锐利度不要超过源，该怎么办呢？

一个想法是，锐化的强度 diff，不要超过 src-nr 的值，也就是降噪所损失的强度。即对于任何像素(无视符号正负)：

diff <= src-nr

上述可以用表达式求值实现；也可以近似用 Repair 实现：diff <= src-nr <= src-nr 的 3x3 像素中最大值。

即，原图是
ABC
DEF
GHI

降噪后是
abc
def
ghi

我们希望中心像素的锐化强度，<= E-e

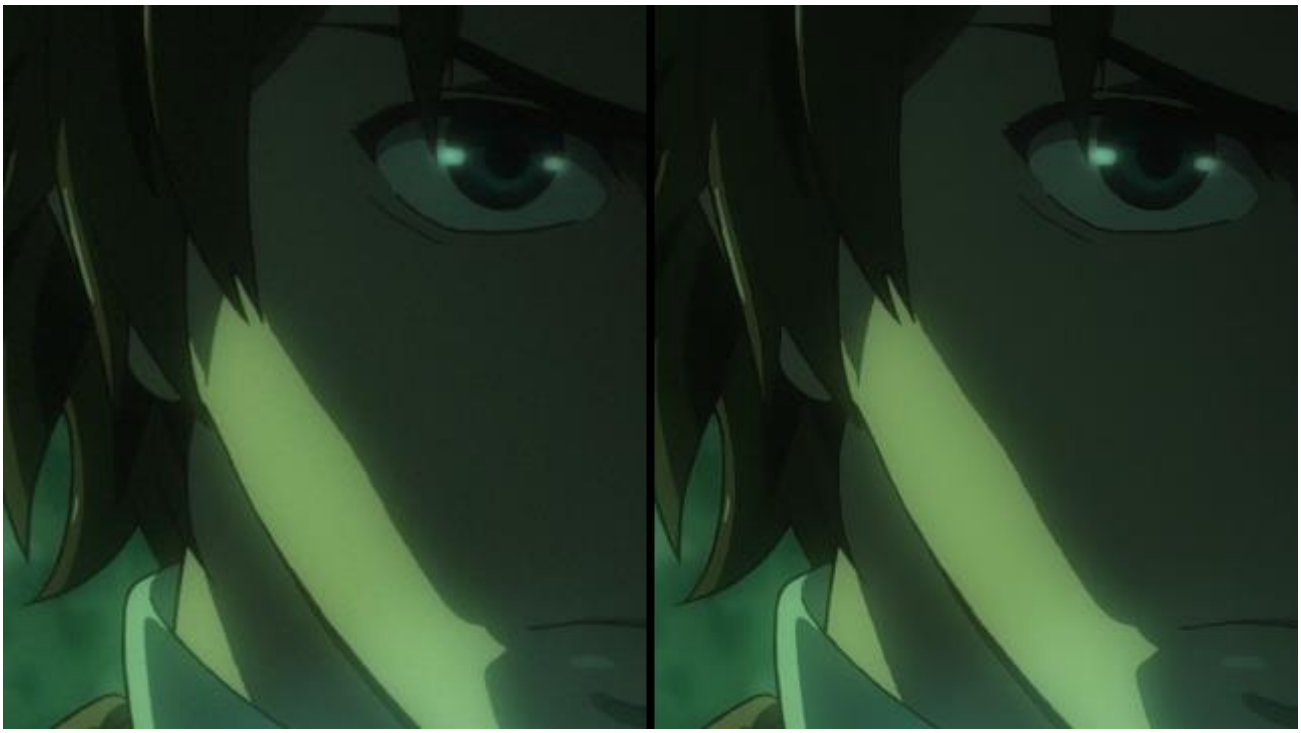
这样锐化后的值 <= e+E-e = E

我们考虑一个放大的近似，中心像素锐化强度<=max(A-a, B-b, ……I-i)

怎么实现呢?如下

```
nr16 = core.bilateral.Bilateral(src16, sigmaS=1.5,sigmaR=0.015, algorithm=2)
noise = core.std.MakeDiff(src16,nr16)
blur = core.rgvs.RemoveGrain(nr16, 11)
diff = core.std.MakeDiff(nr16,blur)
diff = core.rgvs.Repair(diff,noise,1)
res = core.std.MergeDiff(nr16,diff)
```

结果如下：



效果就比之前的好不少了。

Contra-Sharp 被广泛用在不同场合，来对处理后的视频，相对于源做一个补偿性锐化。以上就是它最简单的实现方式；这个实现方式并非严格的保证锐化程度不超过源，而是用了一个近似，允许一定程度放大锐化强度（原则上锐化强度不超过自身损失，现在是允许不超过周边 3x3 像素的损失，即 locally restricted 而非 pixel-wise restricted），这在目视效果上一般是很讨喜的。

不过如上实现可能存在的问题是：当 9 个像素的损失，全部大于锐化强度的时候，锐化强度会被放大到它们当中的最小者。等于说这时候锐化强度会被放大。这个问题的解决方案是：

```
diff = nr-nr.blur()
sharpdiff = Repair(diff,noise,1)
sharpdiff = min(sharpdiff, diff) #保证 repair 后的强度，不超过原来打算做锐化的强度
return nr+sharpdiff
```

这点我们会在表达式求值中继续说。

有另一种写法是这么写的：

```
diff = nr-nr.blur()
sharp = nr+diff
return Repair(sharp,src,1) #相当于基于 nr，用 src 做限制，做一个 Moderate Sharpening
```

参照上文说明，这种做法，锐化结果是被限制为 max(A,B…I)，锐化强度，是被限制在 max(A-e, B-e, ……I-e)，即把中心像素当做降噪后值，来算邻域每个像素的降噪损失。

这样的做法，可以视作降噪本身已经把 3x3 的邻域给抹平了，全都抹成了中心像素。显然，相对于之前的做法，这样做是放大降噪损失，从而削弱限制力度的。锐化出来的图像一般比起 Contra-Sharp 的更锐利，并且带有 Moderate Sharpening 那种锯齿风格。