VCB-Studio教程20: MaskTools (1)

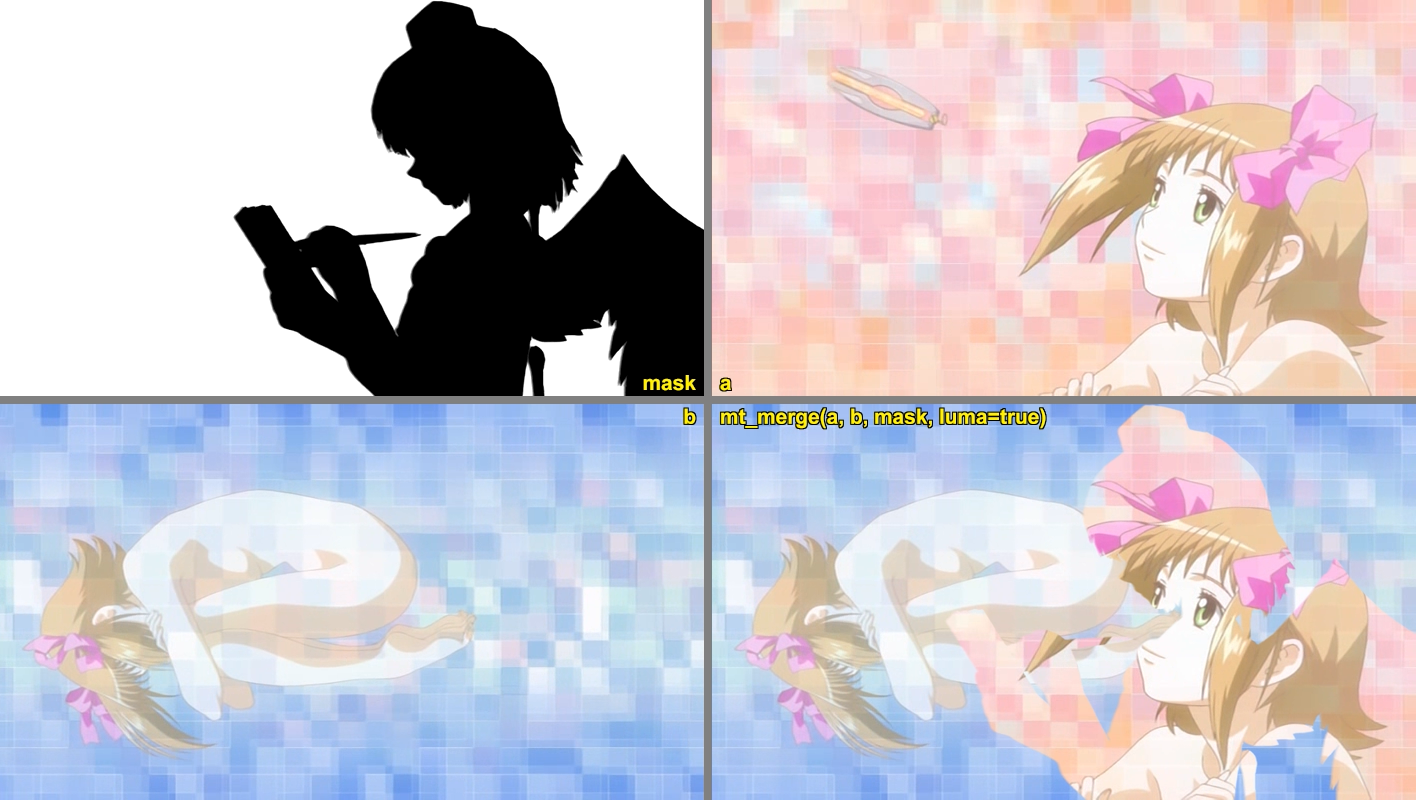
本教程旨在讲述masktools最基本的思路和用法。

1. MaskTools的基本用法

MaskTools的基本用法是mt\_merge(clipa,clipb,bweight)，它通过bweight作为权值，来把clipa和clipb做像素级别的融合：

clipa, b是两个待融合的clip，比如下文的a和b，成品的每一个像素，都由a和b对应位置像素加权融合；

bweight每个像素，决定了输出对应位置，clipb的权值。在8bit下，这个权值就是bweight/256；clipa的权值是1-bweight/256。为了简化理解，我们不妨先来看bweight是0/255两种情况，这样加权出来只有两种可能——要么完全是clipb，要么（几乎）完全是clipa：



mask那图，黑色部分，即射命丸文的框架，数值为0，这部分clipb的权值是0，在成品中完全取自clipa；否则，白色部分，对应值是255，clipb的取值是255/256≈1，图像中取自clipb。

如果clipb的值介于0-255中间，那么图像就会根据ab加权平均。如果权值变动从0-255是平滑渐变的，这样可以起到自然地过渡的效果，比如下图的中间部分就是典型的加权融合：



如果让夏娜那张作为clipb，炮姐那张作为clipa, 那么bweight大概是这样：



mt\_merge除了这三个参数外，还有Y/U/V三个参数来控制三个平面的计算。vs中这个通过planes=[0,1,2]来控制。avs版还有个参数luma，当luma=true的时候，强制Y=U=V=3，且三个平面均使用bweight的luma平面来计算权值。vs版对应的参数是first\_plane = true，表示是否使用bweight的第一个平面计算权值。不过vs中开启first\_plane = true并不会强制三个平面都计算，而是每一个参与运算的平面都适用统一的权值。

avs中，16bit版本的是Dither\_merge16和Dither\_merge16\_8，后者表示mask是8bit。

2. MaskTools的计算细节(选读)

假设在8bit精度下，MaskTools的计算过程是这样的：

res = round[(256-bweight)/256 \* clipa + bweight/256 \* clipb]

实际上，masktools是做不到让clipa的权值0%的。不过这不是什么问题：clipa的权值最少可以是1/256；clipa的最值是255，那么造成的计算误差<=255/256<1，rounding后最多产生1的误差，不是什么大问题。

ps：如果clipa全是x，clipb全是0。我希望clipa的权值是0%，即图像全部取clipb=0。实际上clipa权值是1/256,那么clipa是多少时候，图像结果出来会是1呢？

答：round(x/256) = 1， x>=128。也就是当某个像素，|clipa-clipb|≥128的时候，才有可能造成1的误差。

浮点数运算是很拖累效率的；我们把公式变动一下：

res = round{[(256-bweight) \* clipa + bweight \* clipb]/256}

这样，计算(256-bweight) \* clipa + bweight \* clipb，纯粹是整数运算。

接下来，我们把四舍五入(rouding)转为截位（truncating），不过我们需要手动给结果+0.5，因为round(a) = trunc(a+0.5):

res = trunc{[(256-bweight) \* clipa + bweight \* clipb]/256 + 0.5}

再变形一下：

res = trunc{[(256-bweight) \* clipa + bweight \* clipb + 128]/256}

现在我们有个很满意的结果： 对于整数a和b，c=a/b记录精确结果，并trunc(c)，结果就是多数编程语言中的a/b (整数除法)

所以，我们可以把trunc彻底拿去了：

res = [(256-bweight) \* clipa + bweight \* clipb + 128]/256

这就是taro的教程中给的表达式。我们不妨验证一下上文的问题：

如果clipa全是x，clipb全是0。我希望clipa的权值是0%，即图像全部取clipb=0。实际上clipa权值是1/256,那么clipa是多少时候，图像结果出来会是1呢？

[1 \* x + 128]/256=1

x+128 >= 256, x >= 128

因此，前后结果是完全一致的。

3. MaskTools用于edge和nonedge融合

显然，MaskTools在预处理中的作用可不是用来融合两张毫不相干的图片的。MaskTools的作用主要是区分edge和nonedge，然后把它们融合。简单说：

edge = src.sharp() #锐化

nonedge = src.deband() #去色带

edgemask = src.edgemask() #edgemask假设是一个可以生成mask的函数，结果在线条区域为255，否则为0

mt\_merge(nonedg,edge,edgemask) # nonedg,edge,edgemask 记住顺序，这样使用的时候你不用去多想这三个clip的顺序，而是很自然的记住逻辑：平面，线条，线条mask

生成edgemask，一个简单的方法是：

mask = src8.tcannyMod(mode=1,sigma=1.2,sobel=true)

mask = mask.mt\_binarize(3).mt\_expand().mt\_expand().mt\_inpand().removeGrain(20,-1)

tcannyMod(mode=1,sigma=1.2,sobel=true)，是先用GaussianBlur(1.2)对图像做一个Blur，然后结合sobel算子，用梯度判断线条。出来的图像每个像素介于0-255之间，越高表明“线条指数”越强。这个计算只涉及Y平面，因为UV的信息一般较少，不如直接用Y的信息来引导UV。

随后，我们给定一个阀值，比如3，大于这个阀值的我们判定为edge，否则判定为nonedge。这样，我们通过Binarize这个操作，将图像二值化（0/255）。avs中的mt\_binarize(x)是将图像的地方设置为255，而vs中std.Binarize(x)则是将图像>x的地方设置为255，还是有点区别的。有兴趣的不妨这时候输出看看，这时候输出的mask，通常高亮于线条的两边，线条的中间反而是空的。这是因为出现坡度的地方其实不是线条中心。所以，我们通过expand操作和blur操作来后续加工：

expand：图像中每个像素，替换为周遭9个像素的最大值。用在mask上的效果就是mask向外扩张一个像素，很适合用来填补“空心线”。它的逆操作是inpand，每个像素替换为最小值。在多次expand基本填满空心线条之后，就用inpand给外围长得过肥的mask“瘦身”

类似的滤镜有mt\_infate和mt\_deflate，不过这个是把中心像素替换为周遭像素的平均值，如果平均值更大。deflate则是当平均值更小的时候替换。

RemoveGrain(20)作blur用，将edge和nonedge的边缘smooth一下。

这就是一个简单的0/1 mask设计（0/1表示edge的判定基本上只有edge和非edge两种判断，而不带太多“过渡”）。一般来说，如果你想调整框出来多少，你可以调整tcannymod里sigma，可以调整阀值，还可以通过expand/inpand/blur的后加工来实现。

更多mask设计的方法和思路我们以后还会说。

用avs的16bit实现，将上述表达出来：

mask = src8.tcannyMod(mode=1,sigma=1.2,sobel=true)

mask = mask.mt\_binarize(3).mt\_expand().mt\_expand().mt\_inpand().removeGrain(20,-1)

nr16 = Dither\_removegrain16(src16,20,11)

diff = dither\_sub16(src16, nr16, dif=true) # nr16和diff分别代表低频和高频信息

edge = dither\_add16(src16,diff, dif=true)

edge = dither\_limit\_dif16(edge,src16,thr=4.0,thrc=3.0,elast=3.0) #unsharp mask后接limitdiff

nr16.f3kdb( 8, 48 32, 32, 0, 0, input\_mode=1, output\_mode=1)

f3kdb(16, 32, 24, 24, 0, 0, input\_mode=1, output\_mode=1)

nonedge = Dither\_limit\_dif16(src16, thr=0.35, elast=2.5) #nr-deband后接limitdiff

dither\_merge16\_8(nonedge,edge,mask,luma=true) #合并

自己试着用vs写一写。

4. MaskTools用于自适应edge处理

除了融合edge与nonedge，masktools还可以利用连续性mask(区别于0/1mask)传达的“锐利度”，来对图像做自适应的处理。比如说unsharp mask，我们希望限制它在最锐利的地方强度，除了之前说的Repair和LimitDiff，还有一种方法就是利用mask限制：

利用tcannymod来评估图像中锐利程度，数值越高表明越锐利；

越锐利的地方，锐化的强度越小，也就是锐化后的图像比例越低。

mask = src8.tcannyMod(mode=1,sigma=0.5,sobel=true,chroma=1) #开启chroma处理

mask = mask.mt\_expand().mt\_expand().mt\_inpand().removeGrain(20,-1) #这次不做二值化，而直接利用其连续性。

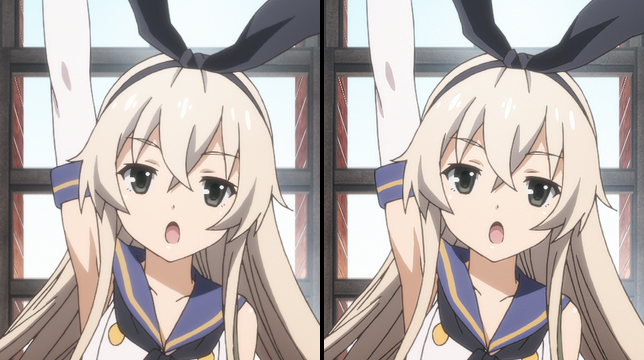
nr16 = Dither\_removegrain16(src16,20,11)

diff = dither\_sub16(src16, nr16, dif=true) # nr16和diff分别代表低频和高频信息

edge = dither\_add16(src16,diff, dif=true)

Dither\_merge16\_8(edge, src16, mask, luma=false) #YUV三个平面各自用自己的评估来锐化。注意，这次edge放在最前面，因为设计中，mask值越低的地方，锐化后的edge比例越高。

效果如下：



思考：为什么锐化的时候，不采用luma平面算mask，然后用luma=true，让三个平面都采用luma生成的mask信息？