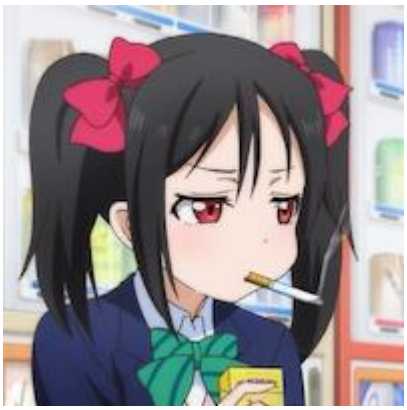
VCB-Studio 教程 13 Resizer (1)

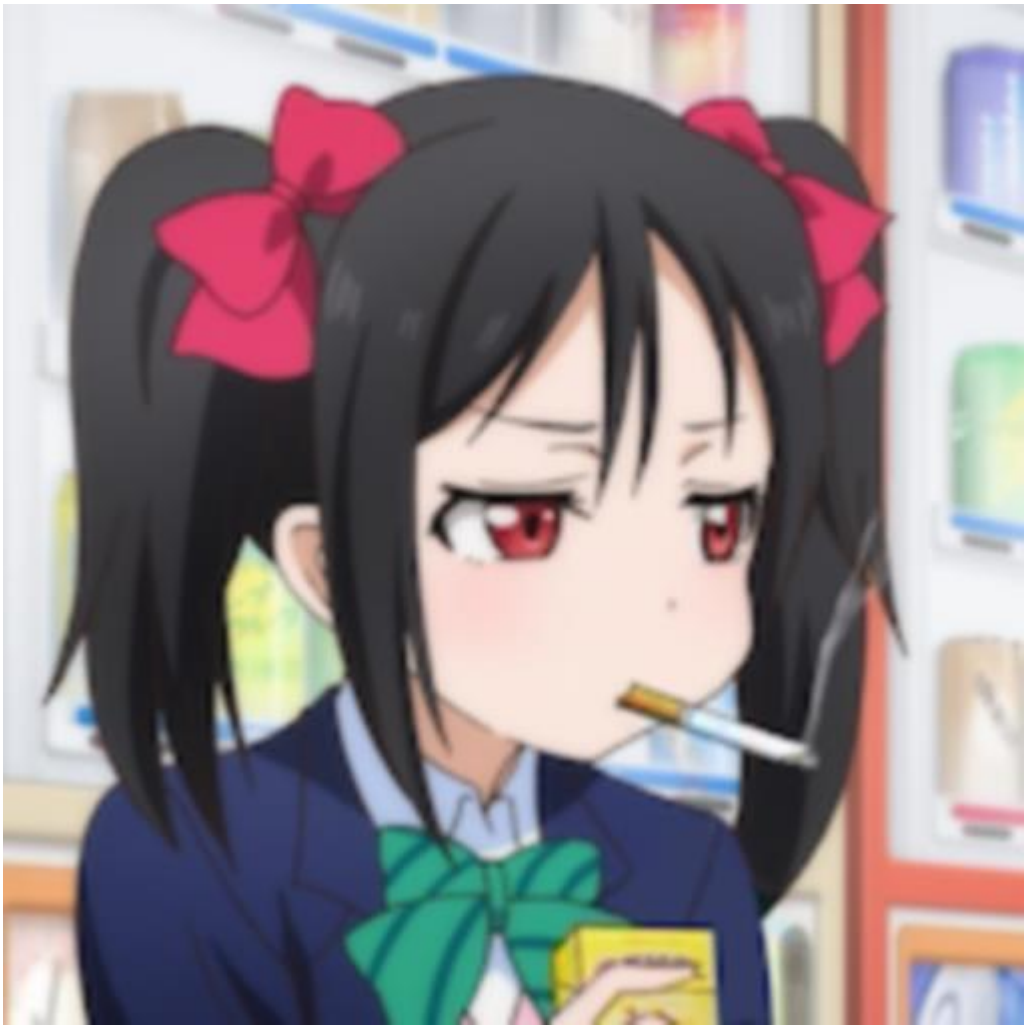
本教程旨在讲述 avs 和 vs 中 Resizer 的基础应用。

1. 图像放大缩小中不可避免的三种 artifacts: blurring(模糊), aliasing(锯齿)和 ringing/haloing(振铃/晕轮)

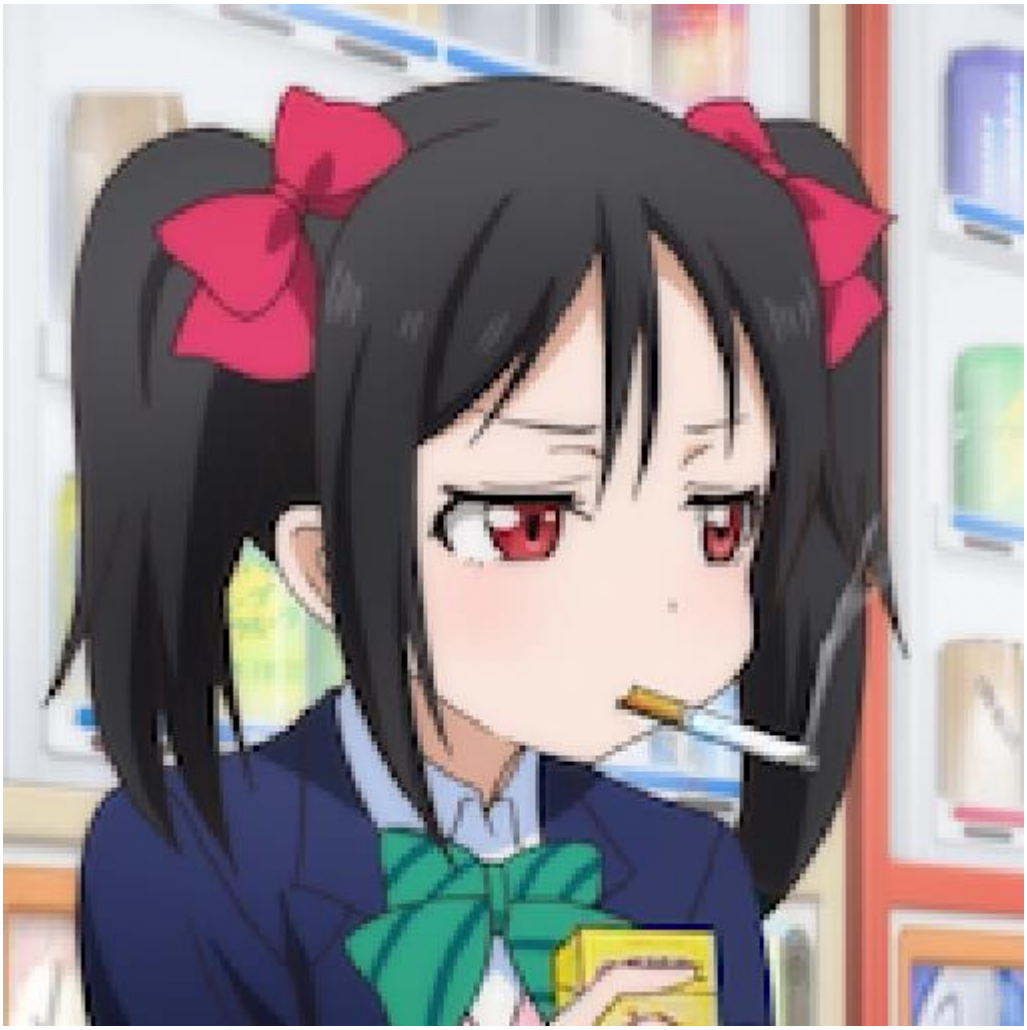
为了保证最佳效果，请在 100%下观看 doc 版本。

这是原图：  ，我们现在把它放大到 2.6 倍

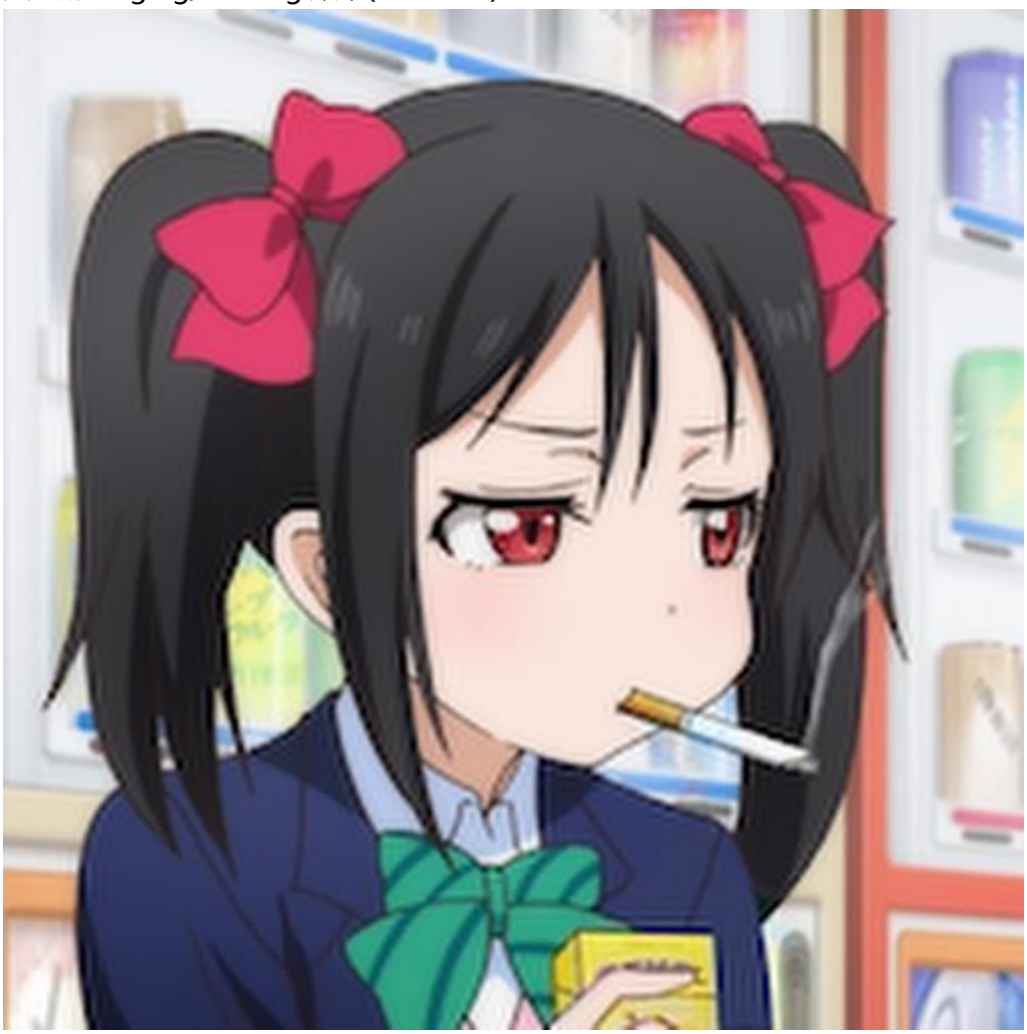
典型的 blurring 效果（guassian p=15）:



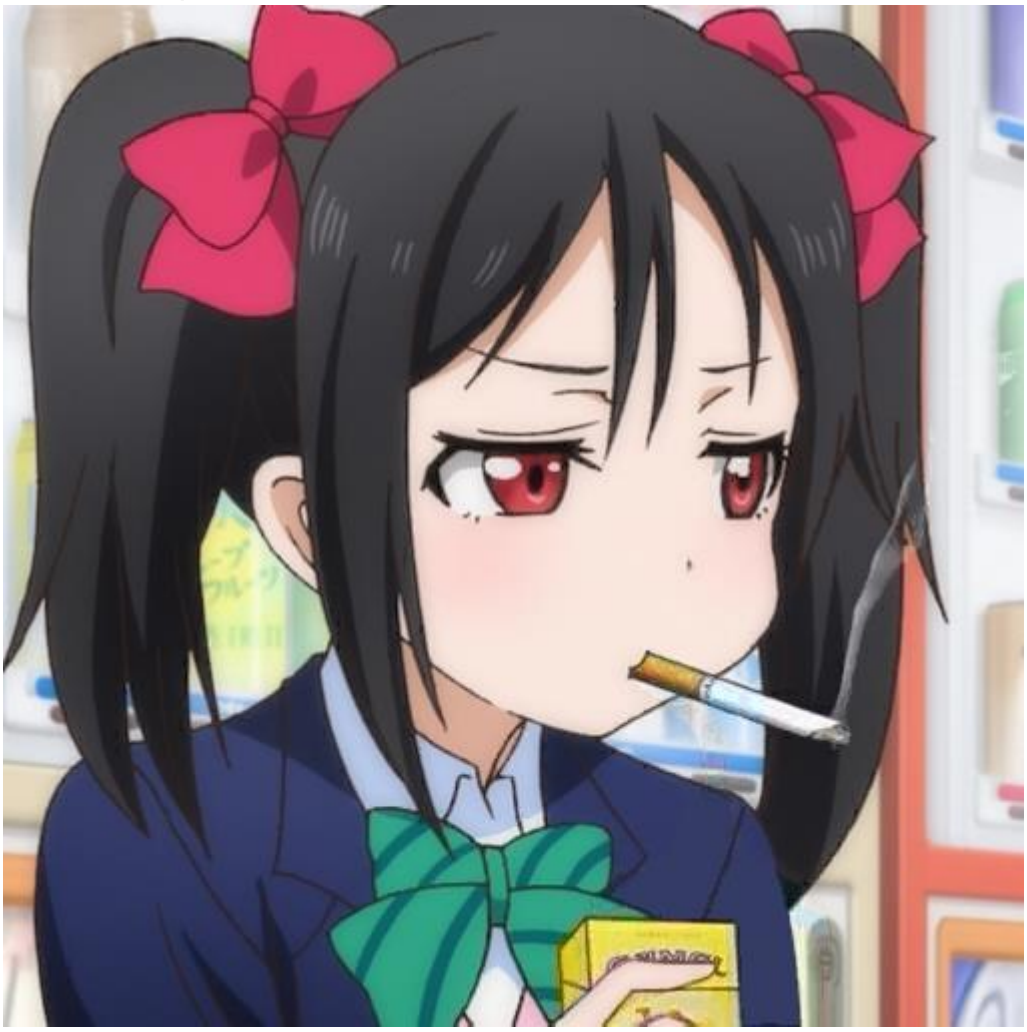
典型的 aliasing 效果(point):



典型的 ringing/haloing 效果(lanczos8)：



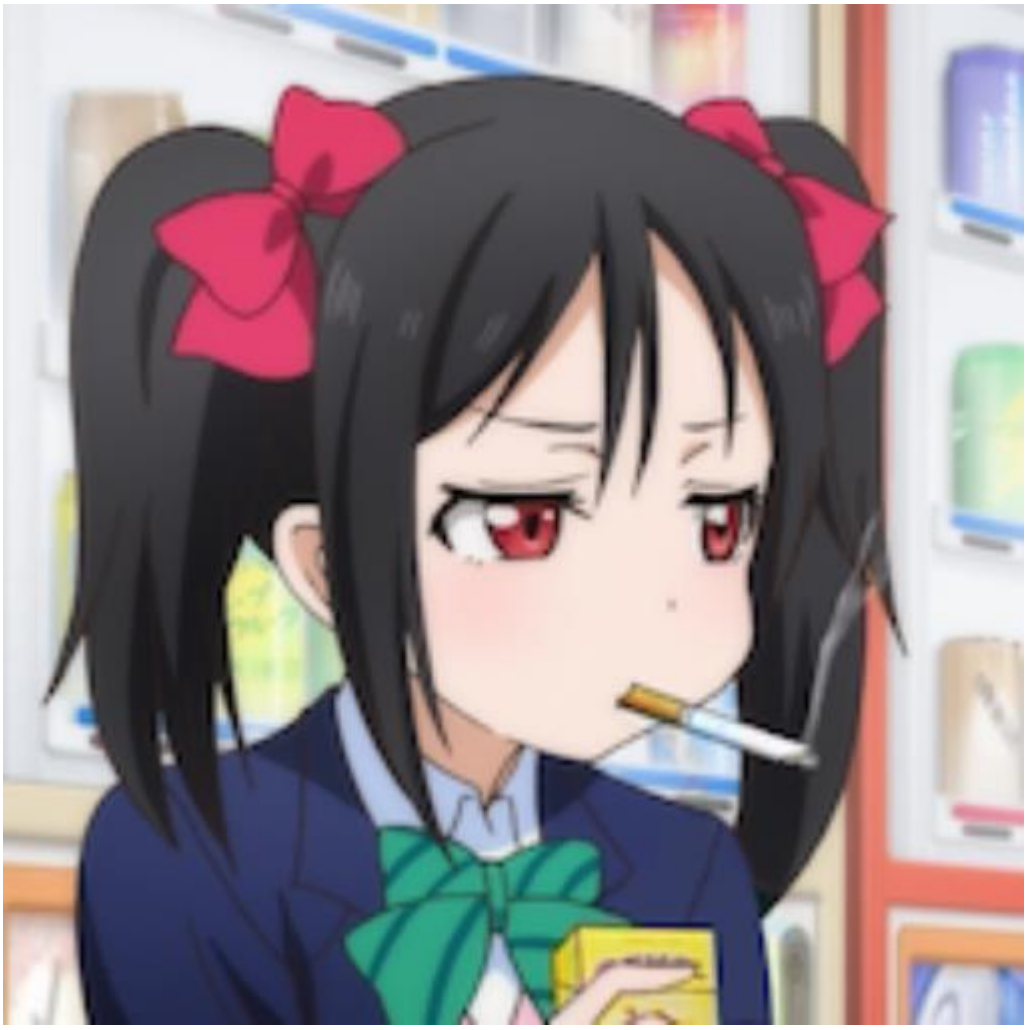
原图（高 sharpness/锐利度）



一个好的缩放算法，就是在尽可能锐利的前提下，尽量控制 blurring, aliasing 和 ringing/haloing。特别是在放大的时候这一点尤为重要。而某些 blurry 的算法，比如 softcubic/guass，因为 blurring 可以掩盖瑕疵的特性，有一些特殊用途（比如可以用来放大 chroma）。
在缩小图像的时候，往往区别不是很明显。

2. Bilinear——最广泛使用的缩放算法。

Bilinear 又称为双线性。BilinearResize(1280,720)是它在 avs 中的用法。效果图如下：



它是简单高效 ,效果不差的算法 ,并且很容易在硬件层面上实现 ,所以被广泛应用。其特点是不出 ringing/haloing，也不 blurry ,但是锐利度很低 ,锯齿多。不适合用来放大 ,但是在一些低码率的编码中 ,如果需要缩小图像 ,Bilinear 是很好的选择。因为它出来的线条锐利度不高，比较节省码率。

3. Bicubic——最广泛使用的高质量缩放算法。

Bicubic 又称为双立方。它比 Bilinear 来的复杂，效果也要略好。avs 中使用方法为 BicubicResize(1280,720,b=0,c=0.5)

b 和 c 是 Bilinear 的两个参数，一般认为，b 代表 blurry 的强度，c 代表 sharpness 的强度。默认值是 b=c=1/3=0.3333…，这两个值是 Mitchell 和 Netravali 测试总结的，最适合人眼的值。所以 b=c=1/3 的 BicubicResize 又被称为标准双立方，或者米切尔算法。

设置 b 和 c 的时候，为了保证图像缩放后的准确性，一般建议 b<0.4，同时 b+2c=1。因为不推荐 b 为负数，所以常规搭配中，b=0, c=0.5 是最锐利的组合。这个组合被称为 Catmull-Rom，是一个很好的高质量 downscale 算法。

当 c>0.6 的时候，图像会被刻意锐化，并且表现出 ringing/haloing，虽然在目视效果上，锐利度的提高往往会抵消 ringing/haloing 带来的副效果。这时候一般依旧是搭配 b=0. 例如 BicubicResize(b=0,c=0.75). 很多播放器会使用这种算法，并且用-c 来代表锐利度。比如双立方（锐利度=-0.6）就表示 BicubicResize(b=0,c=0.6).

你也可以刻意拉大 b 和 c 来看看有啥奇妙的效果，比如可以试试 b=0, c=-5

下图是标准双立方缩放后的值：

avs 的算法中，常用 taps 这个概念来代表确定一个点的值，需要用到多少个它的临近点。Bilinear 算是 1 taps，而 Bicubic 则是 2taps。

无论是 Bilinear 还是 Bicubic，缩放的时候，确定一个像素的值，只用到原图中 2x2=4 像素点。在缩小时候这不是问题，当放大的时候就明显的出现锐度不够的问题。于是我们继续介绍一些常用的多 taps 的 resizer。

Didée 给出过另一组适合缩小的 b 和 c 组合：b=-0.5,c=0.25。Bicubic 通过灵活设置 b 和 c，可以做出 2-taps 的 resizer 能做出的一切效果。mawen1250 曾经推荐用 Catmull-Rom 来处理较好源的 Chroma upscaling.

4. Lanczos——适合放大的缩放算法。

Lanczos 算法是一个广谱，偏放大的算法。avs 中用法是 LanczosResize(1920,1080) 或者 Lanczos4Resize(1920,1080)。后者表示用 4taps Lanczos，锐度更高，锯齿更少，ringing 也更多。
实际使用的时候，Lanczos 做放大一般搭配 non-ringing 算法。这个我们后续再说；下图是 non-ringing Lanczos 4 的拉大效果：

5. Spline——最广谱的缩放算法。

Spline 系列的设计理念是接近 Lanczos 的锐利度，同时控制 ringing 等问题。Spline 系列也有 taps 的区别，Spline36Resize 是 3tpas，适合缩小，Spline64Resize 是 4taps，适合放大。

Spline 系列适合比例不大的时候放大缩小的使用。一般放大的时候也建议搭配 non-ringing 算法。下图是 non-ringing spline 64 的效果：

6. nnedi3_resize16——最好的事实放大算法。

nnedi3 是一个反交错滤镜，基于 nnedi 插值放大算法。
在之前的教程中我们讲过，反交错关键就是将高度切了一半的场景给拉回去，nnedi3 最基础的功能，是将一个图像纵向拉伸到两倍。
那么既然能拉伸两倍，就能通过多次拉伸，拉伸到 4 倍，8 倍……
另外，如果你把图像旋转 90°，然后拉伸，然后再转回去，你还可以用它实现横向拉伸。
所以 nnedi3 的功能就是，将一个图像横向拉伸 2^a 倍，纵向拉伸 2^b 倍。

nnedi3 是一个不会产生、相反还会消除 aliasing 的算法。在需要对付锯齿的地方，比如反交错/抗锯齿操作，nnedi3 是一个常用滤镜。
nnedi3 会产生一点点的 ringing，非常少，少到可以忽略不计。
这个算法在拉伸时候，对线条的效果，对 ringing/aliasing 的控制，完胜 lanc/spline/jinc 这些常规算法。
nnedi3 拉伸后的图像锐利度很高。

现在 avs 版的 nnedi3 似乎只接受 8bit 输入，只能输出 8bit。因此在非线条部分，精度不如已经在 16bit/32bit 下实现的算法。比较容易导致的问题就是色带，那是典型精度不足的症状。好在这种问题不常见，不用过分担心。
nnedi3_resize16 的拉升原理是，用 nnedi3 来处理线条部分保证效果，用高精度 resizer 来处理非线条部分以提升精度。

nnedi3 用来放大到任何一个分辨率的原理 ,先用 nnedi3 拉到一个分辨率 ,再用常规 resizer 做 downscale/upscale：

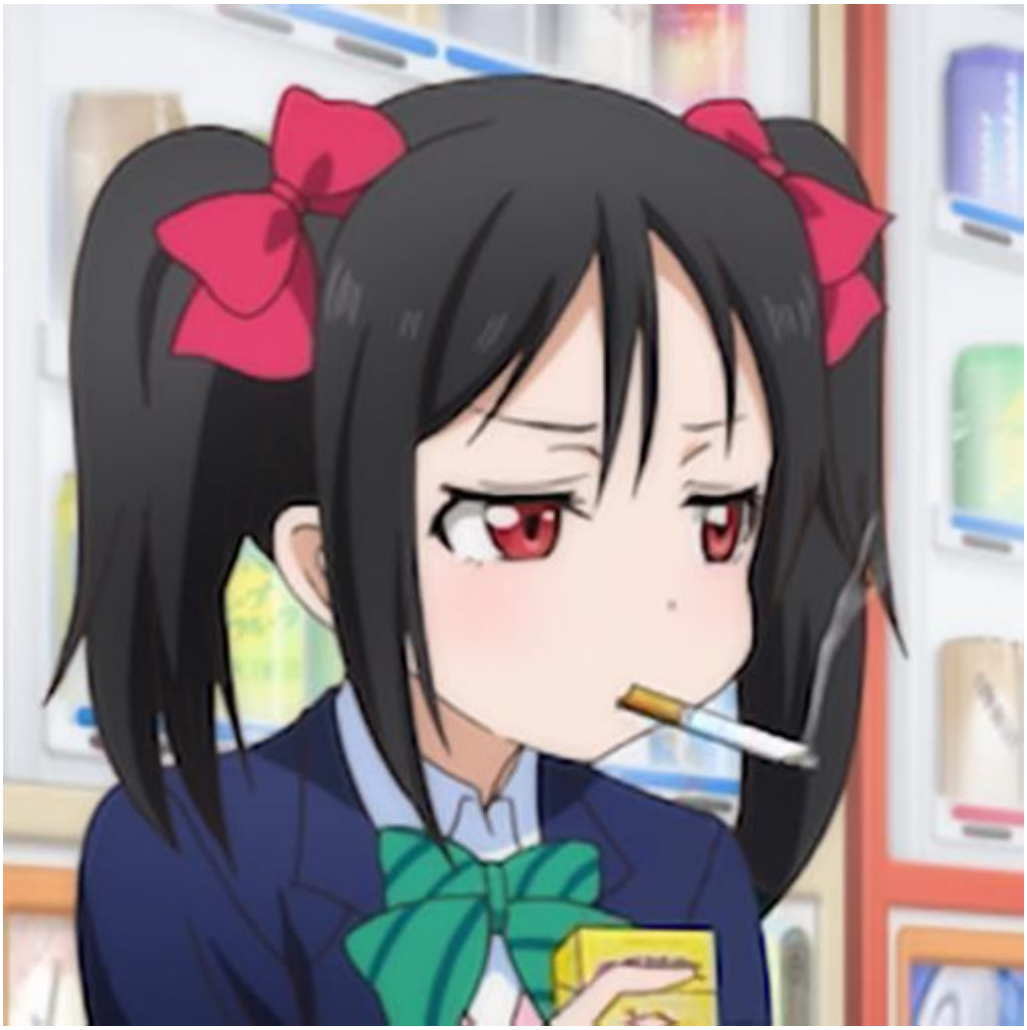
比如我们想将一个 720p 拉伸到 1080p:
1、用 nnedi3 将 1280x720 的分辨率拉升到 2560x1440
2、用常规 downscale 算法缩到 1920x1080

在不允许用 nnedi3 拉伸 4 倍的前提下，将一个 848x480 的 480p 拉伸到 1080p:
1、用 nnedi3 将 848x480 拉升到 1696x960
2、用常规 upscale 算法拉升到 1920x1080

avs 中的使用方法：

```
#  处理到某一步骤是 stacked 16bit
nnedi3_resize16(3840,2160,nns=4,lsb_in=true,lsb=true)
#处理完后依旧是 stacked 16bit
```

这是 nnedi3_resize16 放大后的结果：

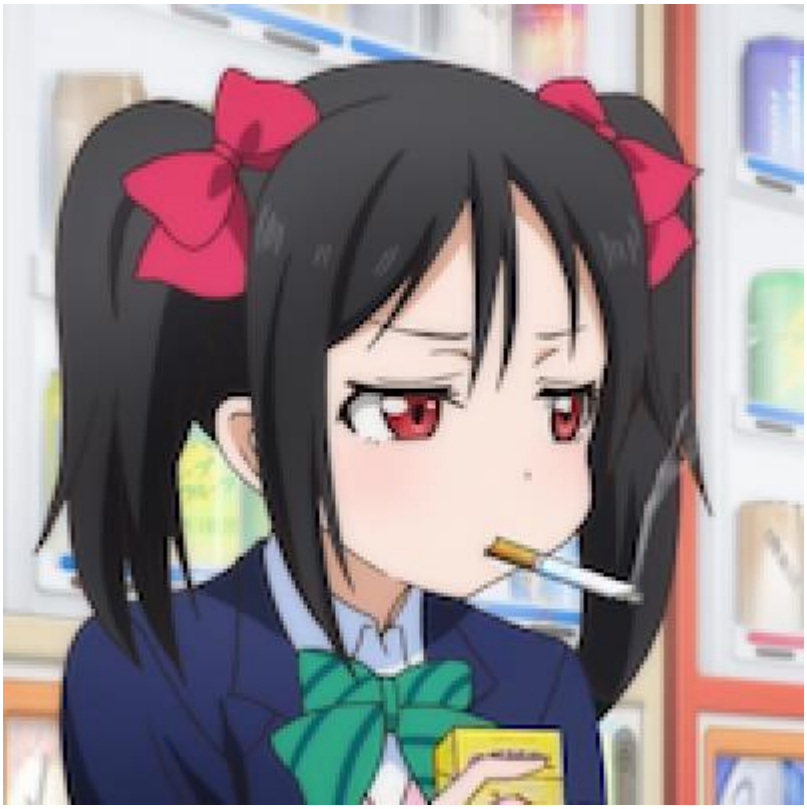


7. 如何保证放大观看的过程中没有干扰——PointResize

point-resize 中文名叫做临近采样，说白了直接找原图中最对应位置的像素点糊上去。后果前面也给了，aliasing 逆天。但是 point-resize 最适合需要放大后仔细观看图片的场合：它可以完美的把一个像素放大成 4 个/9 个/16 个...像素，在这个过程中，没有计算精度的问题。

PointResize 只适合整数倍的放大，而且是为了仔细检查图像像素级别瑕疵的放大。

这是用 point-resize 放大到 2x 的后果。可以看到原图本身就带有次像素级别的锯齿（这在高分辨率素材做 downscale 时候很常见）：

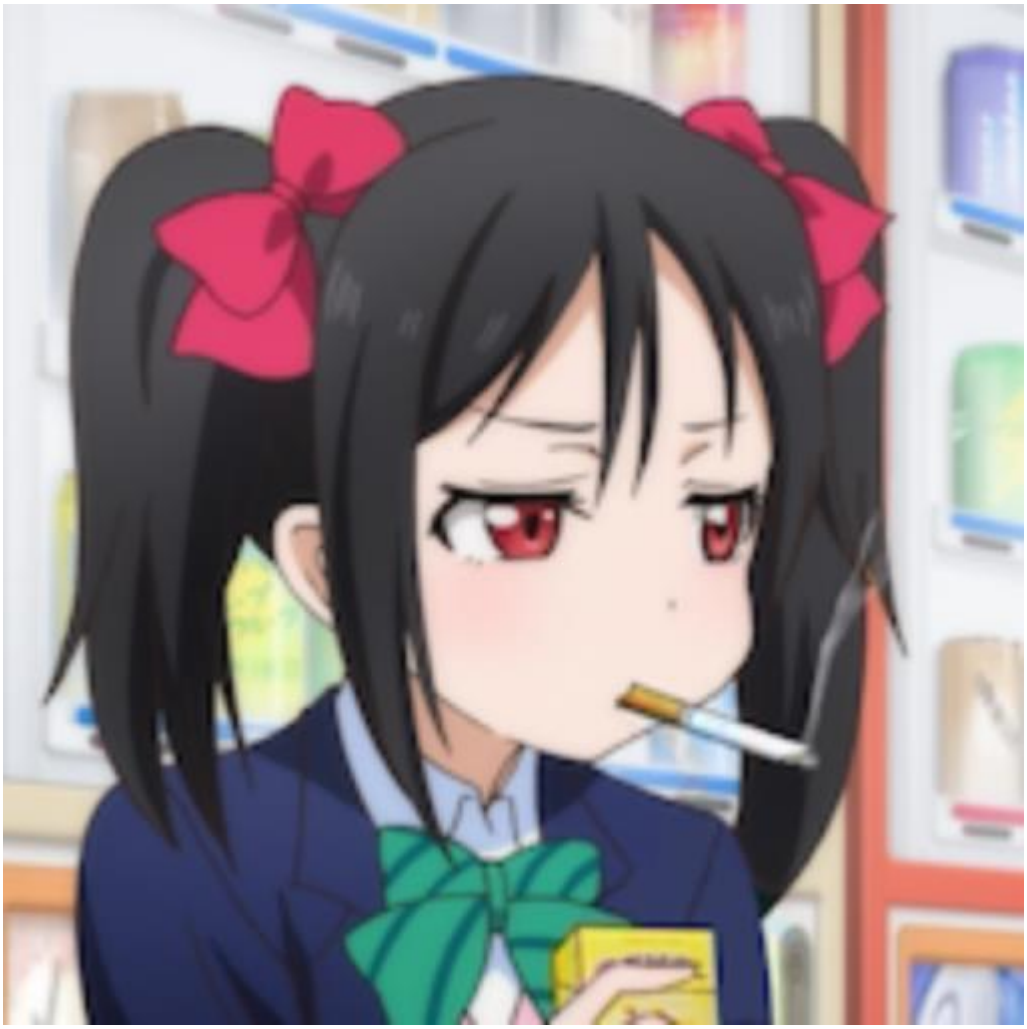


8. 用 blurring 换 ringing 的算法：softcubic/Gaussian

softcubic 是 bicubic 的变种，满足 b+c=1 且 b>=0.5
softcubic 50 就是 BicubicResize(b=0.5,c=0.5)
softcubic 75 则是 BicubicResize(b=0.75,c=0.25)，以此类推。softcubic 100 则是最模糊的。

GaussResize 则是另一种保证不出 ringing 的算法。使用方法是 GuassResize(1920,1080,p=30)
p=1~100,表示锐利度。一般保证很 blurry 的效果用 p=15~30， p 很高的时候则接近 Bilinear 的效果。

这是用 GausResize(p=30)拉出来的效果：



softcubic 经常被用于 chroma upscaling，而 Gaussian Resize 则用于缩放的时候 non-ringing 的实现。

9. avs 中 Resizer 的 16bit 实现——Dither_resize16

Dither_resize16 是 Dither_tools 里提供的函数，可以在 16bit 下做高精度的 resize 处理。这对图像中的低频区域非常有帮助，可以避免因为 resize 精度不足产生的各种问题。

Dither_resize16 只接受 stacked 16bit 输入，输出也是 stacked 16bit。

它通过 kernel 来指定算法(默认 Spline) ,taps 来指定 tap 数量(默认 3) ,a1/a2 来指定其余参数(kernel 是 bicubic 时候默认 1/3 1/3)
比如 Bilinear 算法：Dither_resize16(1280,720,kernel="bilinear")
Catmull-Rom 算法：Dither_resize16(1280,720,kernel="bicubic",a1=0,a2=0.5)
spline36 算法： Dither_resize16(1280,720) #全默认就好
Lanczos 4 算法：Dither_resize16(1920,1080,kernel="lancsoz",taps=4)
Gaussian 算法： Dither_resize16(1920,1080,kernel="gauss",a1=50)

一般来说，resize 应该永远在 16bit 下处理，所以除了某些特殊场合，一概建议在 16bit 下做 resize

10. vs 中 Resizer 的实现

vs 自带的 Resizer 是基于 zimg 的。可供选择的种类有 Bilinear，Bicubic，Point，Spline16，Spline36，Lanczos。
通过 filter_param_a, filter_param_b 来调整 bicubic 下 b 和 c，以及 Lanczos 的 taps。
doc：http://www.vapoursynth.com/doc/functions/resize.html

vs 自带的 resizer 输出精度和输入精度一致。

除了自带的 resizer，还可以用 fmtc.resample。它几乎是 avs 中 dither_resize16 的复刻版。fmtc.resample 的运算精度和输出精度永远是 16bit 整数或者 32bit 浮点（取决于输入是整数和浮点）。所以类似：

```
src8 = core.lsmas.LWLibavSource("00000.m2ts")
down16 = core.fmtc.resample(src8,1280,720)
```

down16 出来就是 16bit 整数。

11. Resizer 的 non-ringing 用法

用 Lanczos/Spline 做拉升的时候，经常建议搭配 non-ringing 算法。如果是 avs 内置的 resize，可以这么写：

Lanczos4Resize(1920,1080).repair(GaussResize(1920,1080,p=100),1)

就是在后面加一个.repair(GaussResize(xxxx,yyyy,p=100),1)

vs 中则必须依赖 fmtc：

upscaled=core.fmtc.resample(src_720, 1920, 1080, kernel="lanczos",taps=4)
upscaled=core.rgvs.Repair(upscaled, core.fmtc.resample(src_720, 1920, 1080, kernel="gauss",a1=100),1)

如果使用 avs 中的 dither_resize16，则更简单：Dither_resize16nr()…
加一个 nr 就好（nr=non-ringing）

12. 修改 Dither_convert_yuv_to_rgb()中 chroma upscaling 算法

Dither_convert_yuv_to_rgb()默认使用的是 Bicubic 算法。你可以在 chromak 参数中调节算法，taps/a1/a2 中调节参数。

比如对画质不是非常好的源，使用 softcubic 60 做 chroma upscaling：
Dither_convert_yuv_to_rgb(a1=0.6,a2=0.4)

对画质非常好的源，使用 non-ringing Spline64/Lanczos4 做 chroma upscaling：
Dither_convert_yuv_to_rgb(chromak="spline",taps=4,noring=true)
也可以用 Catmull-Rom:
Dither_convert_yuv_to_rgb(a1=0,a2=0.5)

或者干脆用 nnedi3_resize16 来处理 chroma 并转为 RGB24:

nnedi3_resize16(output="RGB24",nns=4)

13. 不同 Resizer 背后的数学原理

见 http://svn.int64.org/viewvc/int64/resamplehq/doc/kernels.html
以下我简单的翻一下：

为了让你能更好的了解并选择 resize 算法，这个页面把许多常用算法图表化。没有所谓的“最好算法”，既然这些算法被人研究，那么它们都是有用的，只是适用的场合和使用者的偏好各不一样罢了。

先从一个通用的算法开始，比如 Bicubic 或者 Lanczos，如果你觉得它看上去不够好，和其他的算法比较一下，然后找出更符合你需要和口味的算法。

X 轴代表了理想取样点（比如说，按照公式计算，resizer 后的图像应该对应原图(1.75,1)的位置，那么(1.75,1)就是理想采样点）和实际取样点（但是实际原图中只有(1,1)和(2,1)两个像素，没有小数坐标，那么这两个点就是实际取样点）间的距离(分别为 0.75 和 0.25)。0.0 这个位置表示 resize 的时候最理想样点正好落在原图某个像素上，1.0 表示理想取样点和原图中有的像素，在缩放后的图像里面偏差了 1 个像素。以此类推。

Y 轴代表了在不同距离段采取怎样的加权。注意了有些算法有负数权值。

注意：看图需要 HTML5，IE6 这种滚粗。

以下是我个人一些观察：

Blurring 发生在曲线总体有个正数的偏差。你看下 Bilinear，如果任何算法的权值给的比 Bilinear 高，那么几乎必然需要在某个地方有负数的权值来抵消掉。softcubic100 就不去做这个抵消，所以这玩意糊你一脸。

sharpness，锐利度，跟采样点数量和权值的绝对值总和（就是上上下下曲线围成的总面积）呈正相关。Bilinear 就是最不 sharp 的，尽管它不算 blurry。

如果有负数权值，一定会引起 ringing 效果。负数权值越多越严重，比如 Lanczos 4

13. Resizer 中的裁剪和位移

所有 resizer 中还有 4 个参数： src_left, src_top, src_width, src_height

这四个参数的用法和效果和 crop 是基本一致的，可以用来在 resize 前做裁剪。不同的是，resizer 中并非是物理裁剪，而是重采样过程中的边界值设定问题。所以 resizer 中可以设置小数，而 crop 中必须是整数。

src_width, src_height 可以是正数，表示保留视频的宽度和高度，如果是负数则表示切割掉原视频右边和下边若干像素。但是 src_left 和 src_top 则并非如此，这两个数字是 resize 的时候，左侧和上侧的偏移值，可正可负。这两个可以用来做非整数像素的视频平移。

自己找个 1080p 的视频，试一下这两个 avs 的效果：

Crop(960,540,0,0)
Spline36Resize(1920,1080)

和 Spline36Resize(1920,1080,src_left=960,src_top=540)

Resizer 这四个参数，和 Crop 有着微妙的区别，建议多在 avspmod 中尝试和对比，通过实际操作掌握 resizer 这些参数的设置规律。

src_left 是正数的时候，表示对视频向左偏移。因为视频左边被切掉了一块，导致的后果是整个视频必须向左移动来填补缺失的部分。
src_left 是负数的时候，表示对视频向右偏移。因为视频左边多出来一块空白（resizer 会合理插值），导致的后果是整个视频必须向右移动来放下这块东西。

src_top 是正数的时候，表示向上偏移，反之是向下偏移。

在 resizer 中 src_left、src_top 实际的作用不是切割或者插值，而是指定重采样时，原图的起始位置。比如 src_left=1, src_top=-1 就意味着，把原图(1, -1)这个位置作为 resize 时的原点(0, 0)。因此，我们能够设定小数的位置作为原点，并且那些被“切掉”的部分依然会对 resize 结果产生影响。