补充资料-FFmpeg内存对齐

FFmpeg简单分析系列----内存对齐简要说明

- 在ffmpeg的使用过程中有时会发现align这个参数，那么这个参数代表什么意思，不同的值会产生什么影响呢

行字节数的计算

- 理解内存对齐之前首先要理解行的概念，视频有宽和高两个概念，这里的宽通常就是指行，但他们的大小并不是一一对应的

- 例如720P（1280*720）的宽为1280，那么它的行是多少呢，如果有人直接告诉你是1280,那么这个人可能也没太明白行的概念，因为他漏了一个很重要的前提条件，那就是对齐数align是多少，如果align是1，那么行确实是1280，如果align是2，那么行也是1280，如果align是4，那么行还是1280

- 到这里可能很多人会仓促得出结论，行和宽是相等的，但实际上只是因为1280这个数字比较特殊，它刚好是2的整数倍，也是4的整数倍，类似这样的分辨率还有1080P（1920*1080），4K(4096*2160)等，标准的分辨率基本是对齐的，因此处理视频一定要留意那些非标准分辨率的
- 那么当我们知道宽高后，怎么计算行数呢？其实有一个公式的，如下
  `r = ceil( w*1.0 / a ) * a`
  其中，r是行数，w是宽，a是对齐数，ceil这个函数表示向上取整，即4.1我们会取值5

- 其实上面的说法还是不严谨，我们都知道一个像素点有rgb三个通道，每个通道占1字节的话，那么一个像素点就会占3个字节，那么1280*720在实际存储时每一行的字节数就不是1280了，而是1280*3=3840，那么计算行数时就会变成

r = ceil(w*3*1.0 / a) * a

- 因此对齐是根据实际存储字节大小来计算的，**如果存储一个像素点不是占3字节，**那么我们先要计算出这一行实际占用的字节数，**再根据计算出的字节数来计算对齐**

- 也就是说**行其实就是表示存储图像一行宽度所需要的字节数**

- 例如3*10的图片，每个像素点占2个字节，对齐数align为4，那么行数是多少呢，从条件可知，每行3个像素点，每个像素点占2字节，那么每行就是6字节，而对齐数是4，6不是4的整数倍，因此6需要补2个字节凑成8,8就是4的整数倍了，那么我们就知道每行在内存中实际占用了8个字节，后两个字节是为了对齐补上的，总共有10行，那么这张图片在实际内存中就占用了8*10=80个字节，而不是60个字节了

- 本来这张图片只需要60个字节，为何要用80个字节来存储呢，这是因为cpu并不能从任意地址开始读取数据，如果不对齐，那么可能需要多次读取才能读到完整数据，因此对齐主要是为了提升性能，典型的空间换取时间

ffmpeg中的align

有了上面的基础后，现在你应该能理解ffmpeg中align参数了，**一般我们设为1，那就是按实际的大小进行存储**，不会对齐。

ffmpeg之所以给了这个参数让人设置，应该就是为了兼容各个硬件平台，因为不是所有的硬件平台都能访问任意地址上的任意数据的，某些硬件平台只能在某些地址处取某些特定类型的数据，否则抛出硬件异常

以ffmpeg的av_image_get_buffer_size为例，你能准确说出下面的结果吗，如果可以，那么证明你确实理解了ffmpeg中的对齐了。

值得注意的是yuv的计算，以w*h的yuv420p为例，他是分三个平面存储三个分量的，而u和v的计算是一致的，也就是说计算出了u即可得到v；对于y来说，它有w行h列,因此需要计算w行的对齐后字节数再乘以h；对于u来说，它有w/2行h/2列（这是因为每4个y共享一组uv）,因此需要计算w/2行的对齐后字节数再乘以h/2；v的计算与u的一模一样，最后将这个三个数相加即可

C++ 复制代码

```cpp
int res = 0;
  res=av_image_get_buffer_size(AV_PIX_FMT_RGB24,1920,1080,1);
  log(AV_LOG_INFO, "av_image_get_buffer_size %d.", res);//3*1920*1080=6220800
  res=av_image_get_buffer_size(AV_PIX_FMT_RGB24, 1920, 1080, 2);
  log(AV_LOG_INFO, "av_image_get_buffer_size %d.",
res);//3*1920*1080=6220800，这里由于3*1920刚好是2的整数倍，因此不会产生多余的对齐空间
  res = av_image_get_buffer_size(AV_PIX_FMT_RGB24, 1920, 1080, 4);
  log(AV_LOG_INFO, "av_image_get_buffer_size %d.",
res);//3*1920*1080=6220800，这里由于3*1920刚好是4的整数倍，因此不会产生多余的对齐空间

  res = av_image_get_buffer_size(AV_PIX_FMT_RGB24, 1280, 720, 1);
  log(AV_LOG_INFO, "av_image_get_buffer_size %d.", res);//3*1280*720=2764800
  res = av_image_get_buffer_size(AV_PIX_FMT_RGB24, 1280, 720, 2);
  log(AV_LOG_INFO, "av_image_get_buffer_size %d.", res);//3*1280*720=2764800,
这里由于3*1280刚好是2的整数倍，因此不会产生多余的对齐空间
  res = av_image_get_buffer_size(AV_PIX_FMT_RGB24, 1280, 720, 4);
  log(AV_LOG_INFO, "av_image_get_buffer_size %d.", res);//3*1280*720=2764800,
这里由于3*1280刚好是4的整数倍，因此不会产生多余的对齐空间

  res = av_image_get_buffer_size(AV_PIX_FMT_RGB24, 6, 10, 1);
  log(AV_LOG_INFO, "av_image_get_buffer_size %d.", res);//3*6*10=180
  res = av_image_get_buffer_size(AV_PIX_FMT_RGB24, 6, 10, 2);
  log(AV_LOG_INFO, "av_image_get_buffer_size %d.", res);//3*6*10=180,这里由于
3*6刚好是2的整数倍，因此不会产生多余的对齐空间
  res = av_image_get_buffer_size(AV_PIX_FMT_RGB24, 6, 10, 4);
  log(AV_LOG_INFO, "av_image_get_buffer_size %d.",
res);//ceil(3*6/4.0)*4.0*10=200,这里由于3*6不是4的整数倍，向上取整
ceil(3*6/4.0)*4=20，因此每行产生了2个字节的对齐

  res = av_image_get_buffer_size(AV_PIX_FMT_RGB24, 5, 10, 1);
  log(AV_LOG_INFO, "av_image_get_buffer_size %d.", res);//3*5*10=150
  res = av_image_get_buffer_size(AV_PIX_FMT_RGB24, 5, 10, 2);
  log(AV_LOG_INFO, "av_image_get_buffer_size %d.",
res);//ceil(3*5/2.0)*2*10=160,这里由于3*5不是2的整数倍，向上取整
ceil(3*5/2.0)*2=16，因此每行产生了1个字节的对齐
  res = av_image_get_buffer_size(AV_PIX_FMT_RGB24, 5, 10, 4);
  log(AV_LOG_INFO, "av_image_get_buffer_size %d.",
res);//ceil(3*5/4.0)*4*10=160,这里由于3*5不是4的整数倍，向上取整
ceil(3*5/4.0)*4=16，因此每行产生了1个字节的对齐

  res = av_image_get_buffer_size(AV_PIX_FMT_YUV420P, 8, 8, 1);
  log(AV_LOG_INFO, "av_image_get_buffer_size %d.", res);//8*8+4*4*2=96
  res = av_image_get_buffer_size(AV_PIX_FMT_YUV420P, 8, 8, 2);
  log(AV_LOG_INFO, "av_image_get_buffer_size %d.", res);//8*8+4*4*2=96
  res = av_image_get_buffer_size(AV_PIX_FMT_YUV420P, 8, 8, 4);
  log(AV_LOG_INFO, "av_image_get_buffer_size %d.", res);//8*8+4*4*2=96

  res = av_image_get_buffer_size(AV_PIX_FMT_YUV420P, 6, 8, 1);
```

```
38    log(AV_LOG_INFO, "av_image_get_buffer_size %d.", res);//6*8+3*4*2=72
39    res = av_image_get_buffer_size(AV_PIX_FMT_YUV420P, 6, 8, 2);
40    log(AV_LOG_INFO, "av_image_get_buffer_size %d.", res);//6*8+4*4*2=80，这里对于y来说，6刚好是2的整数倍，因此y刚好对齐；而对于u来说，每行只有3个u,为了对齐必须补一个字节，变成4；v与u是一样的
41    res = av_image_get_buffer_size(AV_PIX_FMT_YUV420P, 6, 8, 4);
42    log(AV_LOG_INFO, "av_image_get_buffer_size %d.", res);//8*8+4*4*2=96，这里对于y来说，6不是4的整数倍，因此y必须补2个字节凑成8后才对齐；而对于u来说，每行只有3个u,为了对齐必须补一个字节，变成4；v与u是一样的
```

ffmpeg的linesize

linesize其实就是我们上文提及到的行字节数，在我们解码出数据后，经常会遇到这个linesize,既然我们知道了align的概念，就该明白这个linesize就是为了让你取出真实的数据的

解码后的数据中可能是经过对齐的，既然有对齐，那就是数据里加多了一些为了对齐而多余的字节，**如果我们想最后显示视频数据，那么这些多余的数据势必要进行剔除掉，那么怎么剔除呢，linesize就是来帮你干这事的，有了它，你就可以一行一行比较，然后把每行最后为了对齐而补的字节删除，还原出视频的真实数据**