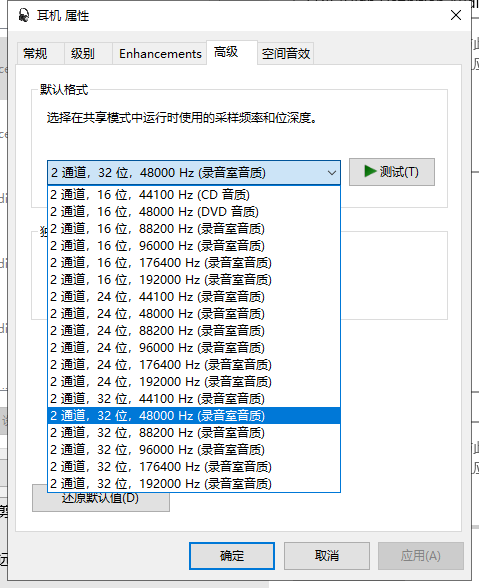
用FFT实现音频分析

这是一个教学文。过去几年，陆陆续续有人问我如何保持那么多年的学习习惯，具体如何学习。老实说，我很少看书，看书也是选择性的看章节。这种情况优点是效率较高，有针对性，缺点是不够系统，会有些知识点的遗漏。本文通过自己从头到尾实现一个FFT音频软件，直观的展示如何找资料，如何分析，如何学习并且实现。先声明：这并非我擅长的领域，我不熟悉信号分析，也不熟悉音频，一切都是从头来。因此，如果有什么错漏的地方，欢迎指正。

首先，要做音频分析，要首先熟悉音频数据。我各方搜索相关资料，各种看，终于弄懂了音频数据的格式。这部分的内容，我基本看的是MS的文档，毕竟这样会比较权威，不容易看到一些错误的文档。

首先，一个关键的信息是采样率。虽然是老生常谈，我还是要说一下。采样率的意思是：每秒钟采样了多少个数据。因为计算机只能处理离散数据，所以这个采样，其实就是连续数据的离散化。音频的采样率有很多种，最常见的有：44100，48000。现在一般的电脑，都是48000.。这个声卡可以设置。随便搜一下百度就有，大概这样：



还有一个双声道，这个就更加不用解释了。所以，一般来说，一秒钟获取到的音频数据，是48000个。由于是双声道，是96000个。

音频数据，一般的解释都是PCM数据，常见的解释是PCM16，意思是16位数据，一个数据是2字节，0-65535之间。但是我的显卡，我获取到的，一直都是float数据，0-1之间。我不知道是资料过时了还是windows一直自己做了这个归一化处理，反正我这边获取到的数据，就是96000 \* sizeof(float)的数据。有知道原因的可以说说。

我这边我自己之前就写过一个PCM音频播放器，录制播放之类的都比较简单，随便看看文档就能写。主要关键字有：*IMMDevice\IAudioClient\IAudioCaptureClient\IAudioClock*。自己搜索这几个关键字，MS官网上就有各种example。如果这还写不出来，那就是c++水平不过关了，只能先去恶补下c++了。

还有一个比较奇葩的问题，那就是有一个设置，是初始化*IAudioClient*的时候，有一个参数，传入的时间单位是100纳秒。这是一个我们比较少用的单位。我们一般是秒，毫秒，纳秒，100纳秒确实不是我们的使用习惯。不过美关系，理解一下，写写就可以了。这块我自己有一些代码注释，加入了我的理解，也贴这里。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

// 这里第三个参数是抄的OBS的代码。我看MS官网的是：10000000，也就是1s。

// 按道理使用MS的才是王道，但是我测试发现抄的代码也没什么毛病，就先算了。OBS传入的是5s。

// 看文档，这里传入的是100纳秒的单位，1秒等于1 \* 10^9那么多纳秒，100纳秒也就是10^7

// 毫秒是10^3，与10^7之间就差10000，所以ConvertMSTo100NanoSec是乘以10000.就是毫秒到100纳秒

// 那么第三个参数意味着什么呢？意味着初始化一个buffer，缓存这个时间间隔的帧数数据。

// 例如传入5s，那么buffer大小就是缓存5s的帧数数据。计算方式是这样的：5 \* 480000 = 240000那么多帧的数据。

// UINT NumFrames = 0;

// mmClient->GetBufferSize(&NumFrames);

// 上面两行代码，就能返回240000

// 这个缓存有什么影响？我估计看你做什么。因为OBS主打直播的，缓存5s没什么问题。一般如果实时处理的话，1s其实也够了。

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

好了，假设我们已经拿到了音频数据，那么，需要做音频分析了。如何实现一个音频分析？稍稍百度一下，你会发现，基本都提到一个概念，也就是FFT。FFT是指快速傅里叶变换。这一块要清晰理解，并且能自己写出来代码，而不是抄，我认为还是有很多技术点的。我这里打算详细讲解整个学习的过程。

首先，FFT是快速傅里叶变换，其实就是离散傅里叶的快速算法。意思是：这个结果跟离散傅里叶（DFT）结果一摸一样，关键在于执行效率更快。所以，要搞FFT，必须首先明白DFT是什么原理，如何实现的，然后再看如何用FFT来优化DFT算法的。

关于DFT，第一步是要明白这是个什么东西。讲这个东西，有一篇知乎最高赞的文章我觉得很值得一读，就是那篇知名的”掐死我吧”。这篇文章全网都是，我这里随便贴一个链接：https://www.sohu.com/a/114138508\_468636。但是，看了这个文章，你知道如何实现一个DFT并且进行音频分析了吗？我是不能的。看了这个文章，最大的收获是：

1、知道了所有的信号，必然由N个周期信号组成。也就是说，任意一个信号，必然由N个正弦波组成（正弦和余弦可以相互转换）。更通俗的举例就是：任何一个离散信号，例如你现在得到了a0, a1, a2……a47999的数据，那么，这些数据，必然可以分解成A + A1sin(x1) + A2sin(x2) + A3sin(x3)……。这里，我为什么用x1，x2，x3……而不是用网上通用的2\*Pi \*k\* n / N来描述？其实，那个2\*pi \*k\* n / N只是一个近似。当N取值很小的时候，其实精度很小。这个理解花了我大量的时间。为什么？其实可以这样理解：你永远无法得出一个直角矩形一样的波形，因为理论上要得到这个波形的正弦波是无穷多个。所以，我们其实只需要“逼近”就可以了。所以，我们教科书一样的答案，其实是分成N个正弦波逼近一个最优解。

2、知道了什么是时域、频率。知道了时域很难找的规律，可以频域找。

扯远了，回头看，我们如何来实现一个DFT？有一个最简单的方案，直接抄代码，抄公式即可。但是，这不是我的风格。

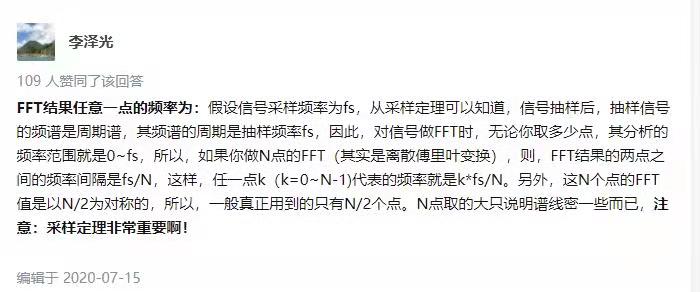
在网上找资料，会告诉你，先把一个单位复数分成N份，然后每一份的虚部和实部是如何计算的，然后用N的数据，乘以这个复数，求和即可。

这里重点关注的，不是这个公式。这个公式网上随处可见，要抄还不容易吗？问题在于，为什么要这样算？这样算为什么能得到结果？

这个问题，我看了无数的资料，看了无数的解释，最终，我终于在知乎上找到了一个大佬的解释，让我觉得是最接近真实的。大佬的链接一下子找不到了，抱歉。这个解释就是：

**我们根据采样得到的点数，把信号分成N份频率的周期函数，然后每个周期函数取样N个点，用这N个点不断与原采样得到的真实数据做点积！结果越大，证明越相似！这个就是DFT的计算方式。**

看图，这里是我认为对于DFT的频率的正确的说法：



这个解释一下子扩宽了我对点积的理解。对于点积的理解，最简单的理解是两个向量点乘，a \*b = |a| \* |b| \* cos(x)，x是两个向量的夹角。

深入一点的理解：是向量a到b的投影或者b到a的投影。

这里的理解是：任意N维的向量点积，理解为两个向量的相似度计算。DFT的原理就是：我现在已经有个N个点，那么我这N个点跟一个确定的周期函数点乘，如果结果很大，那么证明我已知的N个点跟这个周期函数很相当符合，表现出了这个周期函数的特征！

我不能确定这个理解是不是一定正确的，但是就我找到的资料看到的，这是最让我信服的解释了。有其他解释的，欢迎指出。

回到音频这个议题。我们已知音频的频率范围一般是20k以内。假设你现在有10k个采样点，要做一个DFT。本质上，就是把20k的频率，分成10k份，每份精度就是2.，那么你现在分别取频率为2、4、6、8……20k的频率的周期函数，每个函数构建出来10k个点，分别取点乘你得到的采样点。然后判断得到的采样点跟设定频率的相似度。这是DFT的本质。

那么问题来了，这里的计算，为什么要用复数？点积就点积了，用实数不好吗？

关于这个问题，那个大佬有详细的解释。我也有自己加入的理解。这里总结一下：

1. 如果用实数，无论你用正弦波，还是余弦波，必然会导致信息的丢失。这个是那个大佬总结出来的经验，并且有长长的例子证明，我就不列出来了。估计专门搞信号分析的大佬比我了解得更透。
2. 神奇的欧拉公式，神奇的复数乘法。因为DFT的前提是信号的时域频域转换，是周期信号展开成傅里叶级数。当你展开的时候，做各种数理推导的时候，自然而然就推导出来了。这里，我不展开专门的数理推导，因为有大佬已经做得很好了。

以上，属于DFT的个人理解：分别用不同频率的信号得到的数据，对原始信号进行点乘，值越大表明包含同频率函数的特征越多。

**但是，这个理解能构建起DFT的基石吗？确定一定以及肯定是不行的。这个我只是便于自己对这个计算实现方式的理解，真正的公式，需要非常全面以及硬核的数学推导！**

这里，我贴一个我认为很不错的数理推导，公式推导的例子，有兴趣的，可以自己去看看：https://zhuanlan.zhihu.com/p/77345128

好了，DFT的计算便捷理解有了，纯数理推导有了，还缺代码吗？网上代码一大堆。我这里也有一份。这是我自己写的，有详细的注释：

bool Dft::CalcDft(float\* OriginalData, int N, Complex\* OutArray)

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

实际算法是这样的：

首先生成一个复数Array，命名为：Cn，这个Array的大小是N，实部和虚部分别为：cos(k \* n \* θ)和-sin(k \* n \* θ)，n∈[0,N-1]。θ = 2Π / N，k∈[0, N-1]

再生成一个Complex(xn, xn)，命名为：Cxn

然后Cn和Cxn做点积计算。

问题来了，如果没有做任何优化，Cn需要生成N次。原因在于：我们需要算k个点积，每一个点积都要重新生成一次。

但是，由于三角函数是周期函数，所以这里可以做优化。

优化的原理是这样的：无论你三角函数怎么算，必然属于k \* n \* θ，当k \* n > N时，减去2PI即可。其实就是k \* n % N即可

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// 先生成Cn

Complex\* Cn = Complex::NewComplexArray(N);

// 再生成Cxn

Complex\* Cxn = new Complex[N];

for (int i = 0; i < N; i++)

{

Cxn[i] = Complex(OriginalData[i], 0);

}

// 假设OutArray的长度必然为N，不确定传入的时候是否清0的时候，这里清一下

*memset*(OutArray, 0, sizeof(Complex) \* N);

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N; j++)

{

int k\_mod = (i \* j) % N;

OutArray[i] += (Cxn[j] \* Cn[k\_mod]);

}

}

delete[] Cn;

delete[] Cxn;

return true;

}

好了，DFT搞明白了，现在是要搞FFT。

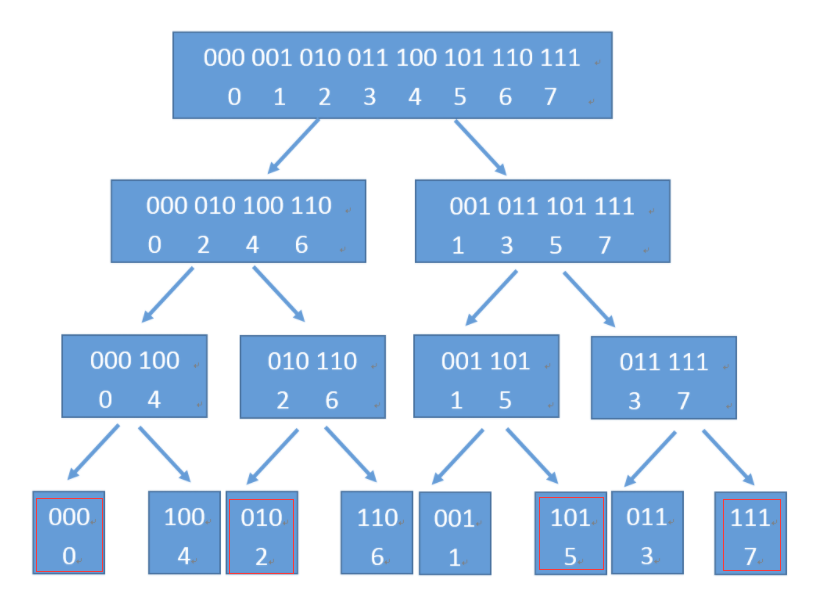
我们已知，FFT不是新的变换，而是DFT算法的快速计算方式。我们已知，DFT理解为两个复向量做点积，这个计算还能优化吗？为什么能优化？关键在于，我们这两个不是任意的两个向量，第二个函数是有规律的周期函数。如果是任意的两个复向量点乘，那就没什么好优化的了，洗洗睡吧。能优化的原因，在于单位复数的乘法，只是一个圆上的旋转。所以才能得到公式：

A(x) = a0\*x^0 + a1\*x^1 + a2\*x^2 + a3\*x^3……an-1\*x^n-1；

把这个公式奇数偶数项不断分解，可以无限细分下去。这里，我就不做阐述了，因为网上有大把这类资料，已经讲得很好了，我非常不擅长在文档里打各种数学符号和公式。有兴趣的可以看看这个链接，这个链接这部分我认为讲解得很好，其他部分讲得一般。

链接在这里：https://blog.csdn.net/enjoy\_pascal/article/details/81478582

所以，FFT最终的方式，其实是不断的递归实现的。不过，这里有优化方案。第一个优化方案，是开局的排序。这里，我实现了两个排序方式。第一种是最传统的，好理解，但是效率低。第二种，是二进制回文的方案。这个方案效率很高，但是没那么容易理解。这里，一图胜千言：



图片来自网络，侵删。

首先，我们要理解到，FFT其实是递归分治，也就是说，要把奇数项和偶数项分开。在第一排的时候，0，2，4，6是偶数项，1，3，5，7是奇数项。所以第二排，是把这两个分开了。但是对于第二排的第一个来说，奇数项变成了2，6；偶数项变成了0，4.所以把0，4和2，6在第三排分开了。我们可以根据这样不断奇数偶数的分开，递归即可。只是说，这个算法效率会低一点。由于这个递归只需要算一次——为什么？因为你算一次，就把索引分好了，只要N不变，保存好这个索引，后面的数据根据这个索引去取数即可。所以效率低点其实无所谓。不过这里还是把效率高的算法也说一下。（上面的奇数偶数项有点乱，不过理解实际意思即可）

仔细观察上图，红色方框的内容，我们会发现，0、2、5、7其实位置是不变的。我们再看这几个数的二进制值，分别为：000、010、101、111。我们再次发现，这些数其实就是回文数。回文数的意思是：从前到后是和从后到前是一样的。这就好比以前看过的很多中文骚对联，典型的例如：上海自来水来自海上、明天到艹场艹到天明……。

所以，这个算法的核心是找出回文数，不变，非回文数，交换一下。

.先排序之后，几个循环就不断相乘相加就可以了。这类代码网上已经到处都是。不过我还是建议自己写一遍。我是自己全部写了一遍，其中也是完全看懂了理论。那个排序我两个算法都写了。这里，我就不贴实际代码了，没太大必要。

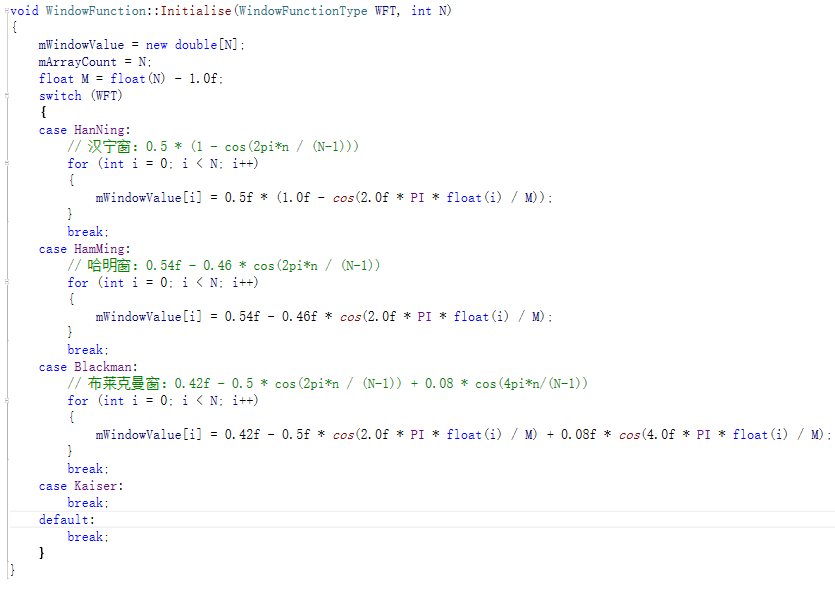
做到这一步，我以为用这个做音频分析足够了，然后兴冲冲的写了一个音频分析，却发现Noise效果非常不理想。我是怎么做的呢？首先获取到音频数据，然后当我获取够了1024个数据的时候，把这1024个数据去做FFT，得到结果之后，求模，显示曲线到屏幕上。

这个有什么问题呢？首先是曲线抖动太大，不平滑。其实是不均匀。前面曲线很大，后面没什么曲线。后来我找了专门做音频的丁神，他给我指明了问题所在。

1. 人类能听到的声音，确实是理论上能到20k的频率。但是，绝大部分声音，尤其是音乐，几乎都在10k以前。而一般的人声，就更低频了。例如女高音一般也就1k的频率。所以，音频分析之后，会发现，后面的频率的数据几乎都没什么意义。
2. 音频分析的时候，由于你获取到的音频，并没有什么可能刚好是周期函数，因此，要加窗函数。不加窗函数的话，会导致泄露。
3. 音频数据要重叠部分，降低抖动。
4. 得到结果后，做一个插值，这样就能保证最终曲线的平滑。

为什么要加窗函数和什么是窗函数？知乎上有大佬是专家，我也是看了他的文章才搞懂的，这里。我就不班门弄斧了，直接贴链接比较合理。毕竟这不是我的专长，我也不熟悉信号分析。链接在这里：https://zhuanlan.zhihu.com/p/24318554

我选用的是汉宁窗。当然了，我其他窗也都写了，要用的话都可以。我听大佬说汉宁窗跟哈明窗就足够了。窗函数的计算很简单，大概在这样：



加了窗函数之后，我再考虑了音频数据的叠加。我是这样做叠加的：

1. 一次采用2048个数据做FFT计算，重叠数据是512个。这样一来，一个数据段前后一起重叠了1024个数据。
2. 计算得到FFT结果之后，忽略直流分量，然后剩下2047个数据，只取前面200个。我算了一下，这样其实取到了频率是2000Hz的数据了，女高音才1.1kHz左右，只单纯对一般的歌曲做分析的话，足够了。
3. 得到结果之后，我做了十次缓冲，加权求平均。

做完以上的步骤，最终结果出来了，我很满意。节奏感、高音、高频、声音的稳定性、颤音的表现，都一目了然。

把以上部分全部用GPU实现一遍。

这里，我用的是DX12的compute shader实现。这部分实现，还是有一定难度的。这里要吐槽一下英伟达。显卡价钱天天涨，搞到我都不敢用CUDA了，万一某天买不起N卡了呢？说到了都是泪。不过老实说，用DX其实会更好一些，毕竟不挑显卡。万一我某天把代码放出来，A卡的跑不了，略坑。

用GPU实现有一个大坑，这里提一下：使用CPU的时候，都是用的double来实现的，但是在GPU里，只能用float（不要跟我说用什么asfloat），在SM6.0之前，讲什么也能用double我觉得都是耍流氓。无奈我的显卡只能支持到SM5.1，只好用float了。

然后我发现，用float跟用double的计算结果，曲线基本一致，但是数值差别较大。我估计是不断叠加导致的误差累积？检查了很久，也没发现代码有什么问题，GPU代码也不是那么方便调试，后来只好作罢了。

到了这里，其实这部分已经全部完成，你已经完全独立实现了一个FFT分析音频的软件。你会发现，这个软件是如此优秀，可以媲美网上大部分同类软件。甚至比很多\*\*\*音乐软件的nosie感觉更好。很多人都不断重申不要造轮子，其实你造了这个轮子，你可以按照自己的想法各种魔改，优化等等，也不需要花太多时间，代码量非常少，为什么不呢？为什么执着于用SDK，寻找各种库呢？别人的库真的写得更好吗？好在哪里？

知乎上有一个月经话题，就是40岁以上的程序员是不是混不下去了，要转行送外卖了。老实说，我年过40，从来不觉得自己混不下去了，我觉得我能写到60岁。如果你能保持一年完全搞懂一个FFT这样的实现，以及完全理解周边相关的知识点，十几年积累下来，搞懂了十几二十个这样的领域，何至于混不下去？怕的就是调用了十几年各种SDK的。

好了，不继续说教了，没什么意义，再说下去要歪题了，每个人为自己的选择负责。

下面说一个跟本题有关的我觉得有点意思的东东。首先，我愿意押上我全部的声誉保证（此处雷布斯微微一笑，嘴角勾起弧度）：这个软件我完全用了我上面提到的所有算法，没有做任何其他的取巧优化等等。这个软件让我看明白了很多我以前不明白的，听不懂的东西。这里总结一下。以下会涉及到一些明星，如果谁觉得被侵犯了，不用联系知乎删帖，直接联系我，我直接删掉。

1. 张靓颖是所有我测试过的女歌手里最牛逼的。那首《天下无双》后面那个“啊……”的部分，直接高音突破了天际！超越所有我听过的男女歌手。
2. 如果以张靓颖为标准，打分为1.0，那么0.6的歌手有那么三两个，我印象深刻的有沈娜的中国红，还有一个新人叫黄霄云的，都很厉害。
3. 0.4-0.5的女歌手有不少，例如韩红，谭维维，黄绮珊，谭晶，李雨儿（代表作：雨花石）等。基本上，知名的女高音歌手，都能到这个段。
4. 以前看《歌手》的时候，觉得信被淘汰完全过份，500评委都是聋的传人。但是用这个软件分析了一下，不得不承认，信绝对是录音棚歌手，现场500人耳朵没问题。
5. 之前看一个评论，说什么萨顶顶的《左手指月》虽然很好听，但是高音被黄霄云吊打了云云，我表示萨顶顶原唱更好听。但是分析了一下，确实高音部分被吊打了。萨顶顶的高音估计到0.4都到不了。
6. 以前听很多翻唱，觉得好屌啊，吊打原唱之类的。典型的就是迪玛希的歌剧2。然后我分析了一下，发现原唱真的是原唱，迪玛希那个连洛天依都比不过。

以上是纯数据分析，绝对不涉及到歌曲好听不好听这种事。毕竟好听不好听，跟高音无关，跟很多因素有关，例如歌唱的技巧，感情，音色还有个人的西好等等。这本来就是很主观的事情。我会把这个软件放出来，有兴趣的可以自己去分析看看。

这个软件的所有逻辑实现上面已经说得很清楚了，觉得分析得不对的，请指出算法哪里有问题，或者指出这样的算法不客观等等，而不希望纠结于结果变成吵架，那就没什么意思了。

软件下载地址在我的github上，链接在这里：，里面那个Release.rar就是了。打开这个软件，直接播放音乐即可看到。