音乐可视化实验报告

1. 实验目的

将一段音乐通过图形可视化的展现在频幕上。

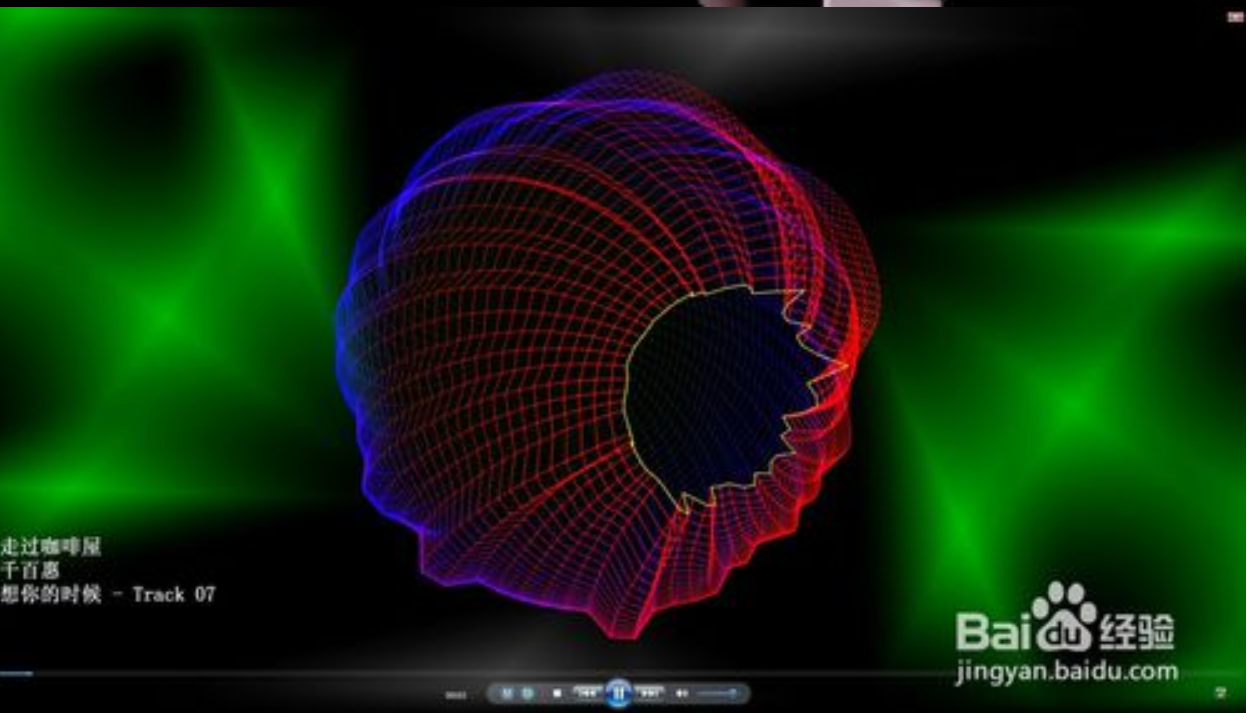
1. 实验工具

我一共做了两个版本的可视化，一个使用c++和openGL实现可视化的图形部分，使用w32mci实现音乐的播放部分，另一个使用python和openGL实现可视化的图形部分，使用pyaudio实现可视化的音乐播放部分。

1. 实现细节

这部分我将分别介绍我实现的两个音乐可视化解决方案的实现细节，其中这两个版本主要的区别为C++的是将音频文件先转化为pcm格式，然后将pcm中的数值全部读入内存进行预处理，再播放mp3文件音频进行可视化的，在音画的同步性上我花费了一些功夫。Python版本使用了pyaudio库，使用两个线程一个进行音乐播放，一个进行可视化操作。音乐播放线程每次读取一个frame然后通过全局变量进行可视化操作，因此很轻易地完成了音画的同步。

我的可视化灵感来源与我小学时候看到的windows media play的音乐可视化，当时觉得那个可视化做的引人入胜：

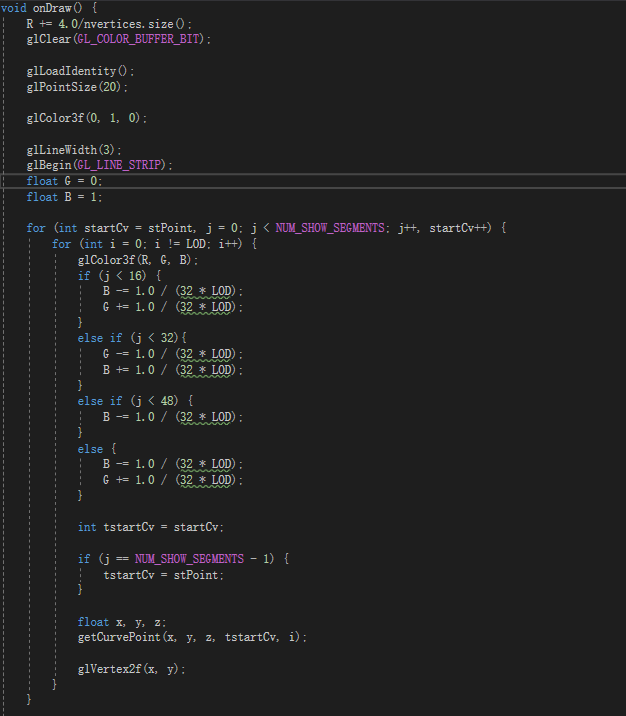


大概效果如上图所示。我主要想实现中间变换的曲线那部分，利用了课上讲道德b样条曲线的算法，使用b样条曲线对音乐进行可视化。

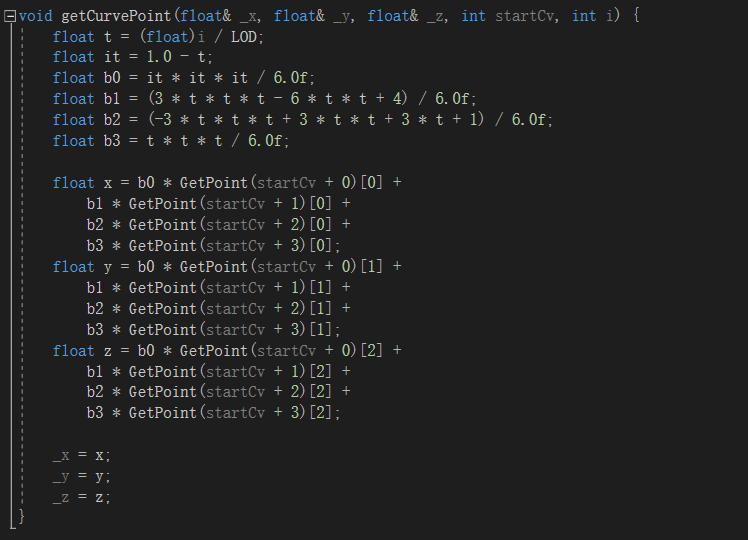
由于c++版本和python版本主要的差别为对音乐播放及音画同步的处理上，所用的b样条生成算法都是一致的，我将先介绍所用的算法，以及公共的数据预处理部分，之后在分别介绍对于音乐播放部分c++和python版本的不同之处。

* 1. 算法

B样条的绘制上我使用了封闭周期的三次B样条曲线，下面代码为C++版本的实现：



其中有两个循环构成外层循环遍历控制点，内层循环将每个B样条曲线均匀分成LOD段，在每个小段调用getCurvePoint获取位与曲线确切位置的点，两点之间使用GL\_LINE\_STRIP直线进行插值。计算曲线上点的坐标代码如下：



这列给出的是c++的实现，使用了引用的方式进行返回值的传递，python的实现中直接返回了三个值的元组。输入为起始点的下标startCv、以及现在正在绘制的分段编号。算法上使用了均匀划分的均匀周期三次B样条曲线的生成算法，预先计算了调和函数在其支持区间上的权重。由于是三次B样条曲线，其支持区间由四个点组成k=4，其中b0, b1, b2, b3分别为四个控制点的调和函数。然后通过曲线方程计算出x，y，z坐标值。其调和参数由下述公式得到：

可以看到k次b样条仅与紧挨着的k+1个定点相关，将k=3带入可以得到代码中的三个调和权重。

* 1. 数据预处理

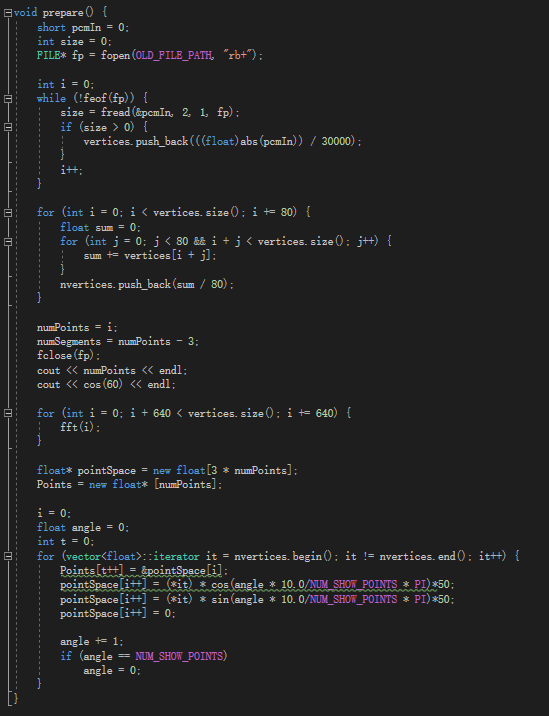
这里主要对于单声道pcm格式的音乐文件进行预处理，在c++和python的实现中略有差异，因此分开介绍。

* + 1. C++数据预处理

C++实现方式中数据的预处理比较复杂。对于给定的音乐文件，我先手动使用ffmpeg进行格式转换，变为pcm格式的文件。转换的参数为：

ffmpeg -i test.mp3 -acodec pcm\_s16le -f s16le -ac 1 -ar 16000 16k.pcm

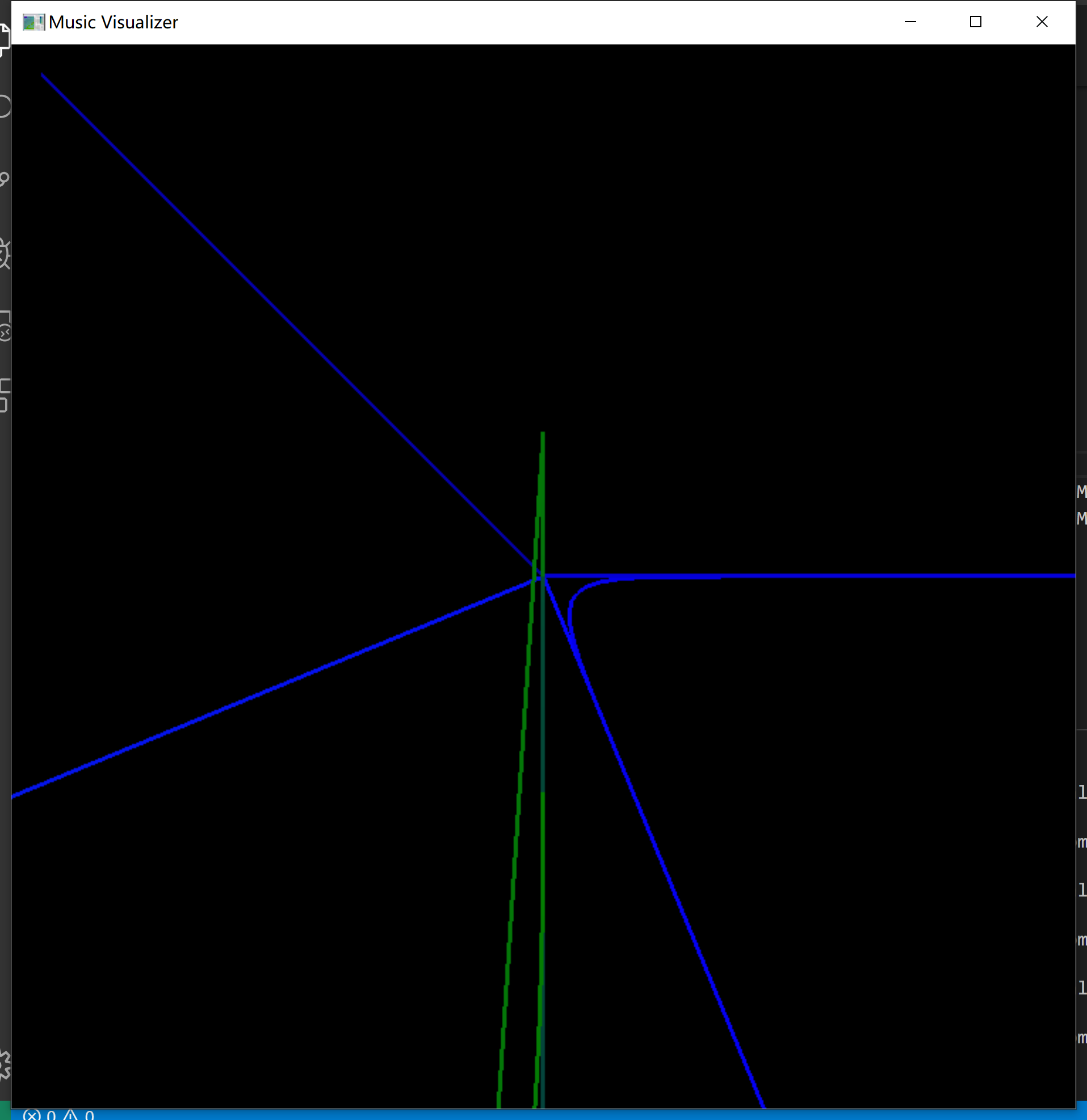
限定了采样频率为16k，这个参数在之后有很重要的作用。然后我将整个pcm文件读入内存中，将每个采样值进行如下预处理工作：



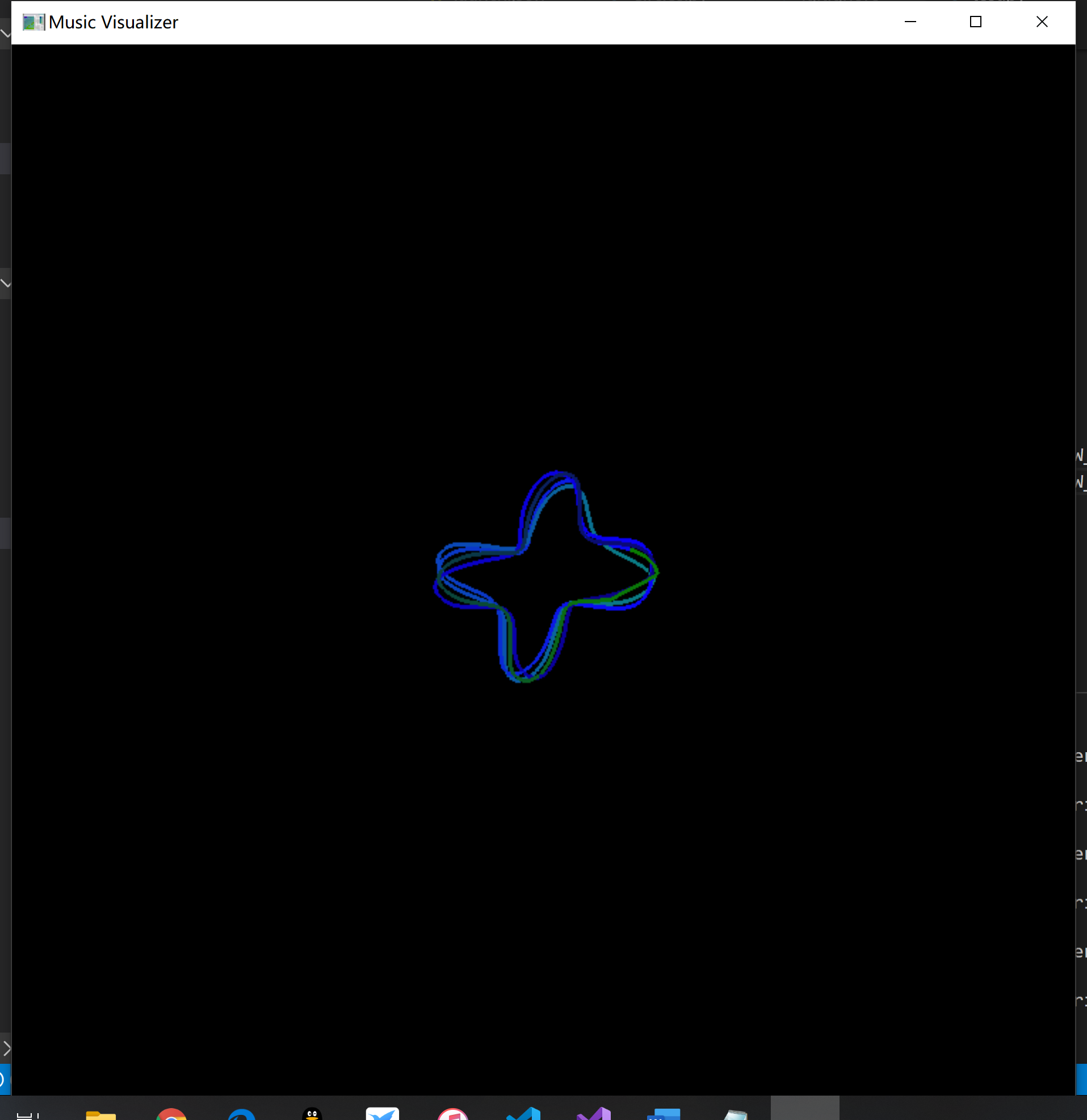
先将每个采样点的数指除以30000然后取绝对值压缩到较小的数据范围中，然后对所有的数据点进行了平均的下采样，下采样的思路为将相近的80个数值点压缩成为其均值这样可以减少所有需要可视化的数据点的数量。最后由于我想在频幕上显示出环状的条带，因此我将每个数值点按照64为周期乘以响应的角度，将这个数指作为半径转化为极坐标点，这样生成的控制点可以产生绕原点分布的B样条圆环。

其中下采样的原因有两点：

* 一方面是因为音画同步需要减少每一帧显示的数据量
* 另一方面是因为如果不使用下采样数值点之间差异很大，得到的b样条环带凹凸不平有点刺球的感觉：

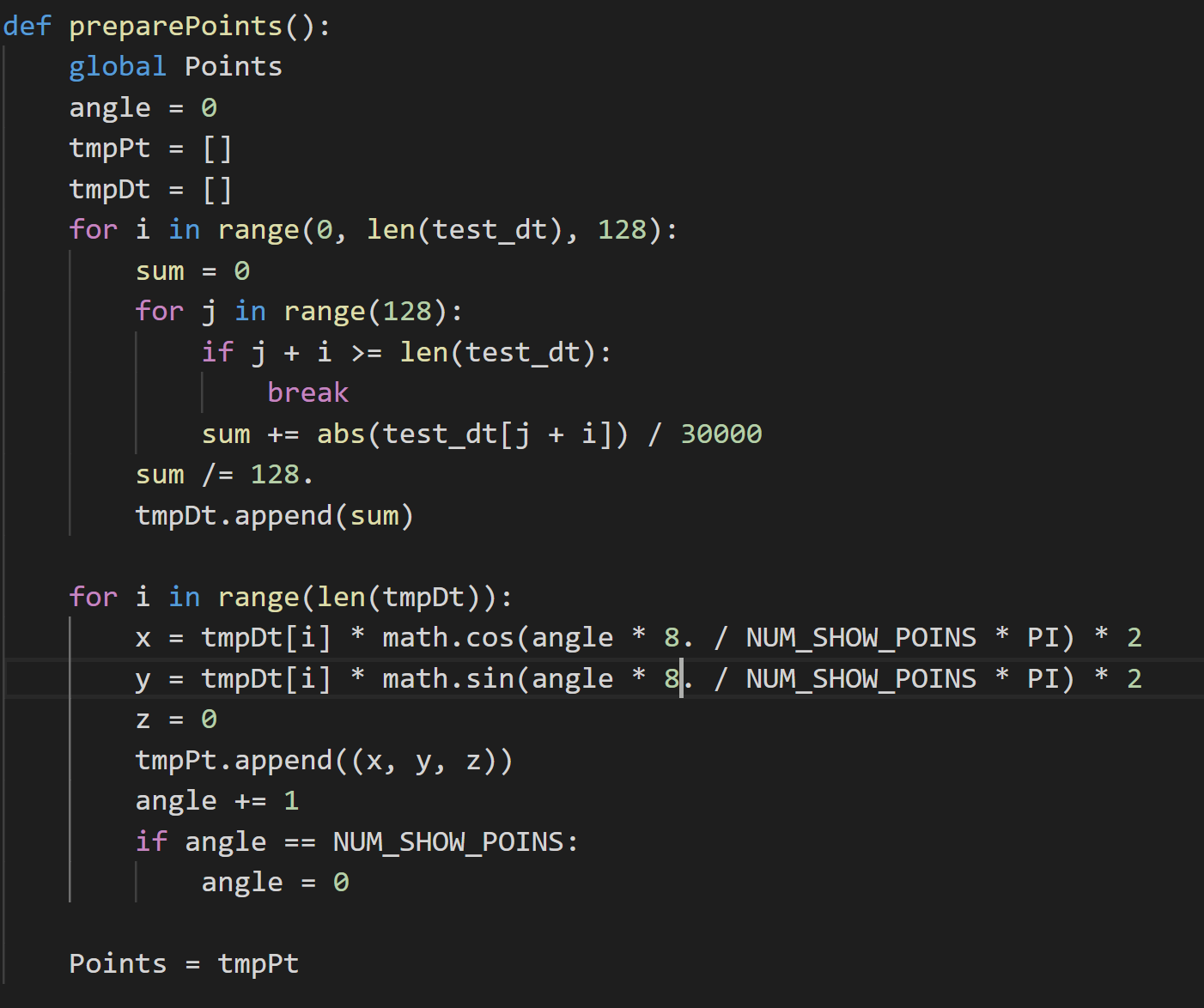


使用了平均值下采样以后得到的数据值之间的差异会显著减小，因此使用之后得到的图像如下：



* + 1. Python数据预处理

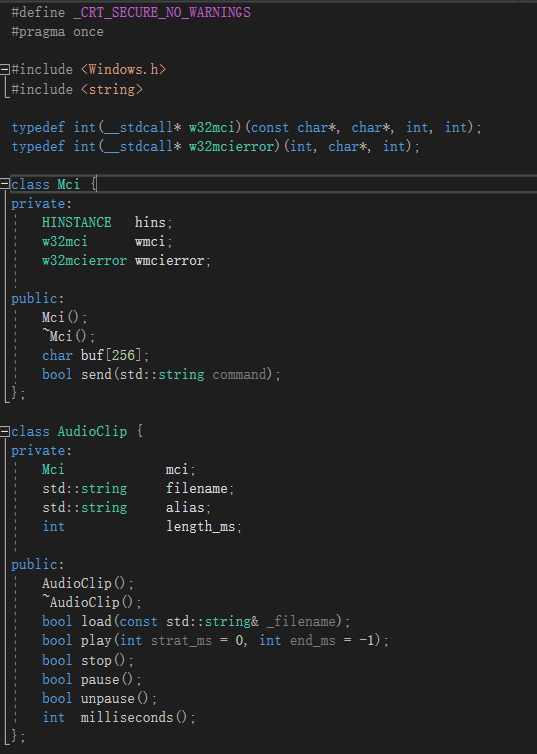
Python实现的数据预处理过程与c++大致相同，但是不同的是对于下采样的个数，经过实验得到使用128个点进行平均值下采样后得到的图像最好而且视觉效果最佳，处理代码如下：



别的坐标转换的方式与之前相同。

* 1. 音乐播放
     1. C++音乐播放

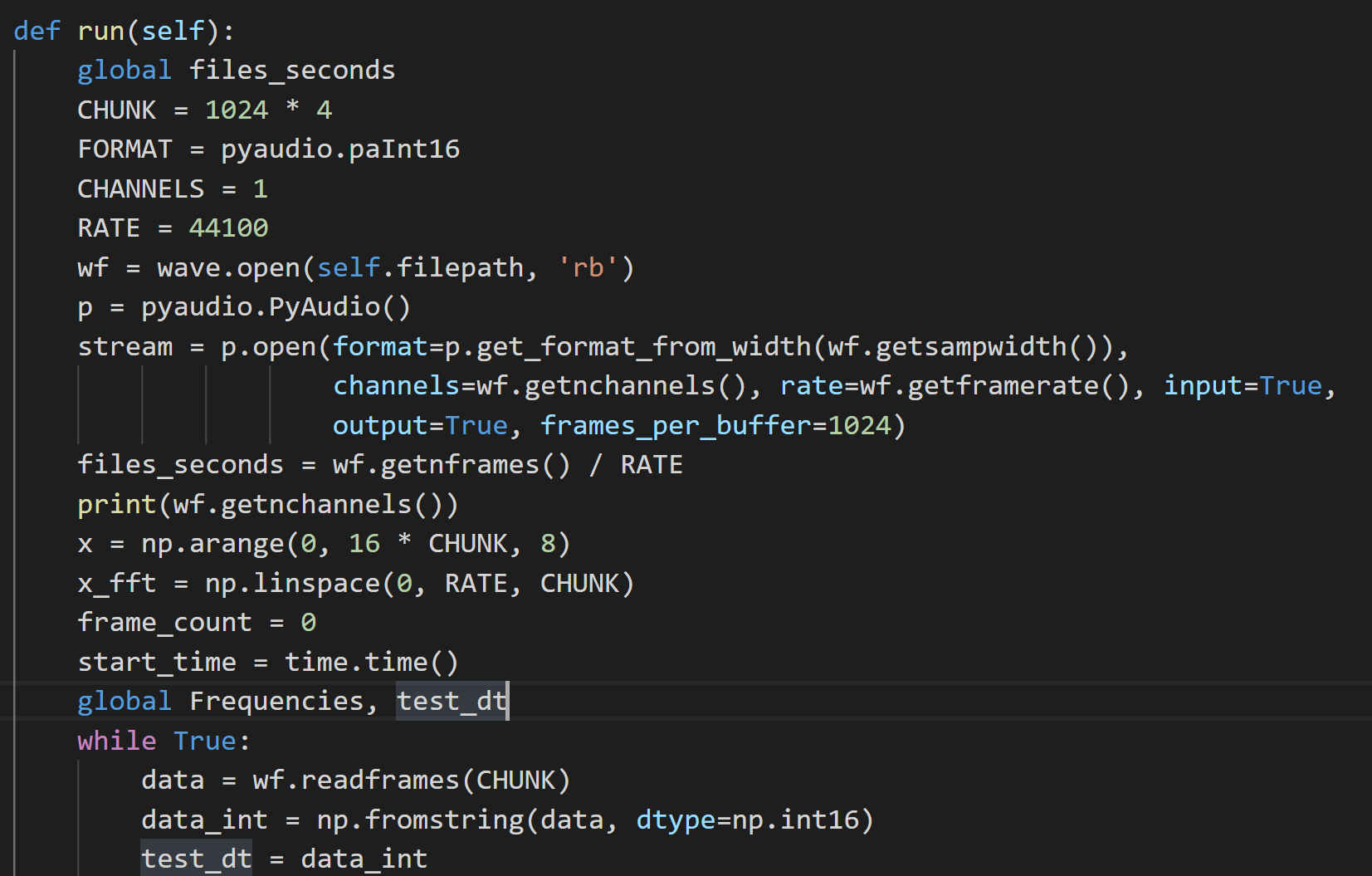
C++音乐播放上我使用了windows自带的win32mci结构，因此无法移植到别的操作系统上，使用系统库我实现了音乐播放类music：



需要使用时先使用load函数加载mp3文件，然后使用play函数进行播放音乐。

* + 1. Python音乐播放

Python实现中的音乐播放方式使用了pyaudio库，每次获取一个frame并写入到流输出中以播放音乐：



* 1. 可视化优化
     1. **音画同步**

Python的音画同步比较简单，由于每次取一个音频frame的数据然后播放，对于python只需要考虑每次输出这一个frame的数据因此音画一定保证同步。这边将着重介绍c++中我为了使音画同步使用的技巧。

前面提到了输入文件的pcm采样频率为16kHz，因此每秒的数据量为16k个数值，为了达到音画同步就需要每秒可视化出16k个数值点。同时要注意的有两点，首先为了使画面具有连贯性，画面不能简单的每秒更新一次，对于每秒的帧率我参考了标准电影的每秒24帧的水平，保证每秒输出的数据超过24帧。其次是为了美观的考虑，同一时刻可视化出的信息量不能太多，也就是每一帧显示的数值点不能太多。

为了达到音画同步，我需要解决的问题有：

* 减少每秒需要可视化的数据量
* 控制图像输出的时间间隔

对于第一点，我使用了平均值下采样的方式，将每80个连续值用这80个点的平均值替代，这样将一秒钟内需要可视化的数据量缩减为了原来的1/80，只需要可视化200个数据点。其次是选择帧率，我选择每秒显示50帧（50fps），这样得到每一帧与前一帧的数值差距为4个数值点。这样就把每一帧需要显示的数值压力减轻了许多。

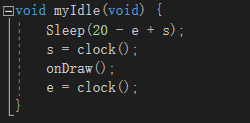
最后我的可视化参数为：

* 每一帧显示64个数值点
* 每秒显示50帧
* 下采样为1/80
* 每秒滑过200个数值点

由于我每一帧显示64个，与前一帧不同的只有四个数值点，因此保证了大部分图像不变，由于我前面周期性的角度转换映射，实际视觉效果在没有激烈的声音变化情况下会呈现以种花纹在旋转的感觉。由于均值下采样保留了突出数值点对局部的影响，在有激烈的声音变化的条件下会呈现以闪现的大线条轮廓的感觉。

由于我选择了50fps的显示参数，因此在理想情况下每一帧显示完成后需要暂停20ms再显示下一帧。但实际上由于画面的绘制需要处理时间，因此如果按照固定的20ms去暂停在处理时间的误差积累下会造成较大的音画不同步的问题。假设每一帧的处理时间为2ms，音频时间为3min，则会造成3\*60\*50\*2=18000ms即18秒的延迟。在音乐播放完成后还有18秒的画面显示是无法接受的，因此需要考虑修正暂停时间。

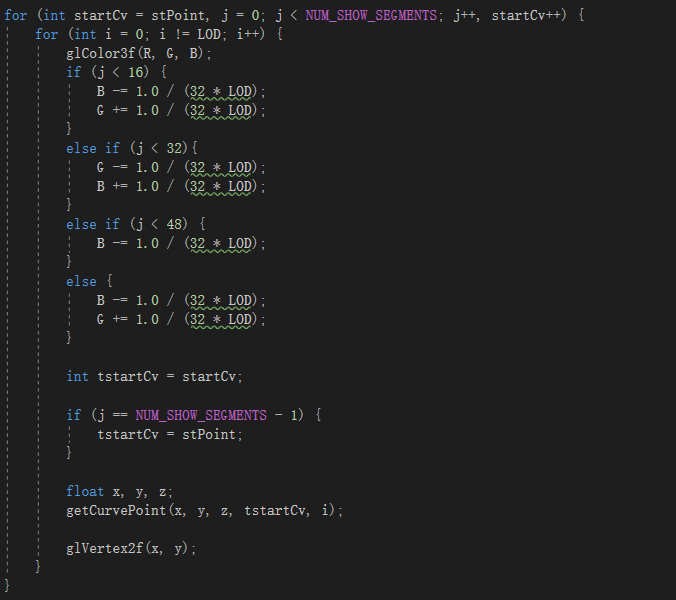
我尝试了两种修正方式，一种是通过输出处理时间肉眼找到处理时间的规律然后通过硬编码的方式将平均处理时间从暂停时间中减去。这样观测下来的暂停时间应为18ms。另一种方式是通过clock函数记录绘制每一帧的开始时间和结束时间（精确到毫秒），然后在下一帧的暂停时间为20-(前一帧的结束时间-前一帧的开始时间)如下图：



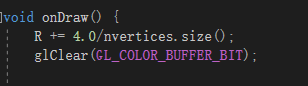
其中第二种方式达到的音画同步效果较好，但是仍然会在3min的音乐结束时产生1s左右的画面滞后现象。这原因主要是来源于clock的计时精度为1ms，小于毫秒级别的误差累积造成了最后的画面滞后。但仍然达到了不错的同步效果。

* + 1. 线条颜色

除了达成音画同步之外，我还对可视化环带的颜色进行了调整，由于我每一帧显示64个点，我将所有点分为4组，每16个点为1组，对同一组中的点进行相同的颜色渐变操作：



整体色系为冷色系，由蓝绿色组合而成。除此之外，另外的红色作为进度标识色，在每一次调用绘制函数时会略微递增，直到在音乐结束时的红色份量达到1（最大）：

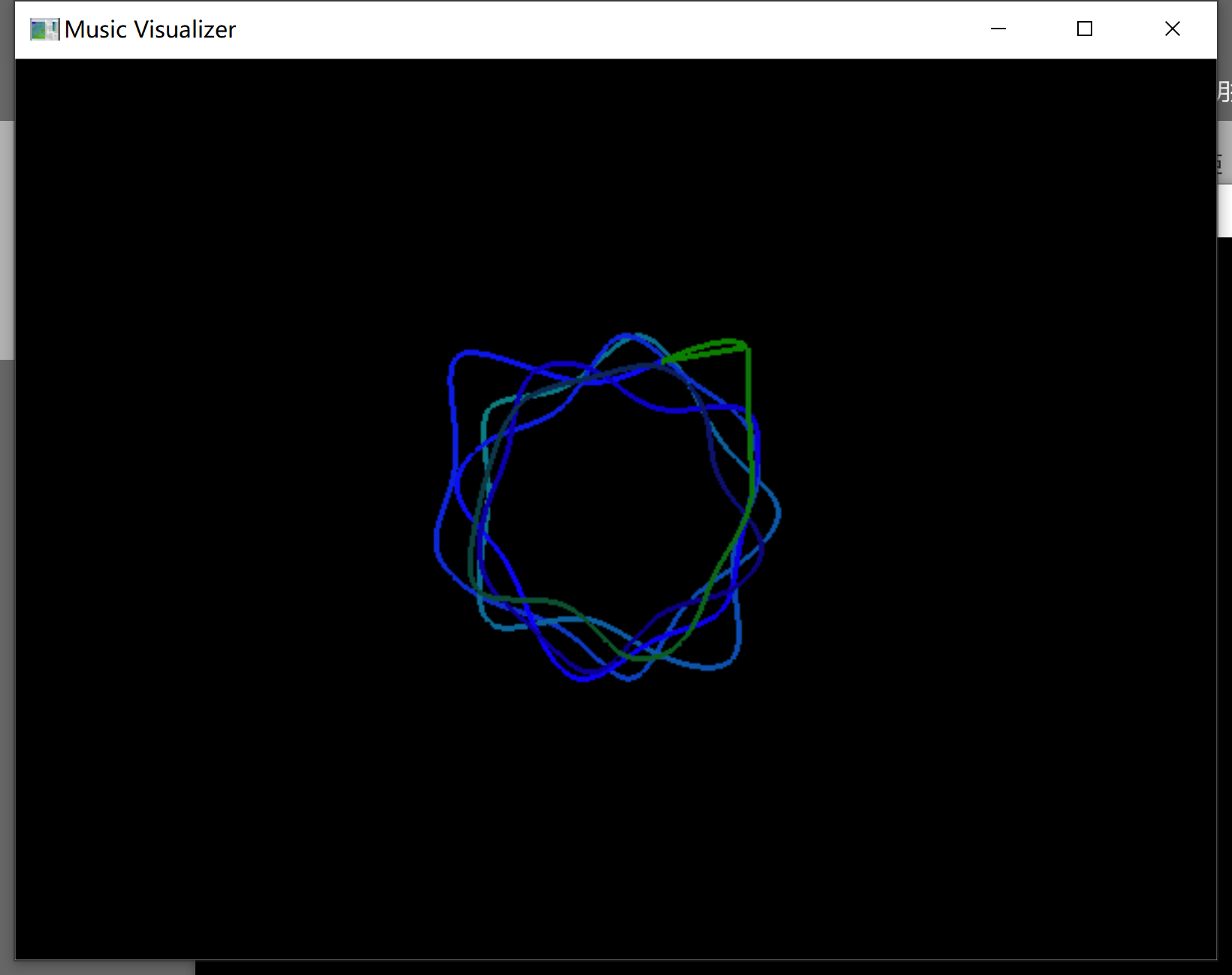


* 1. 差异比较

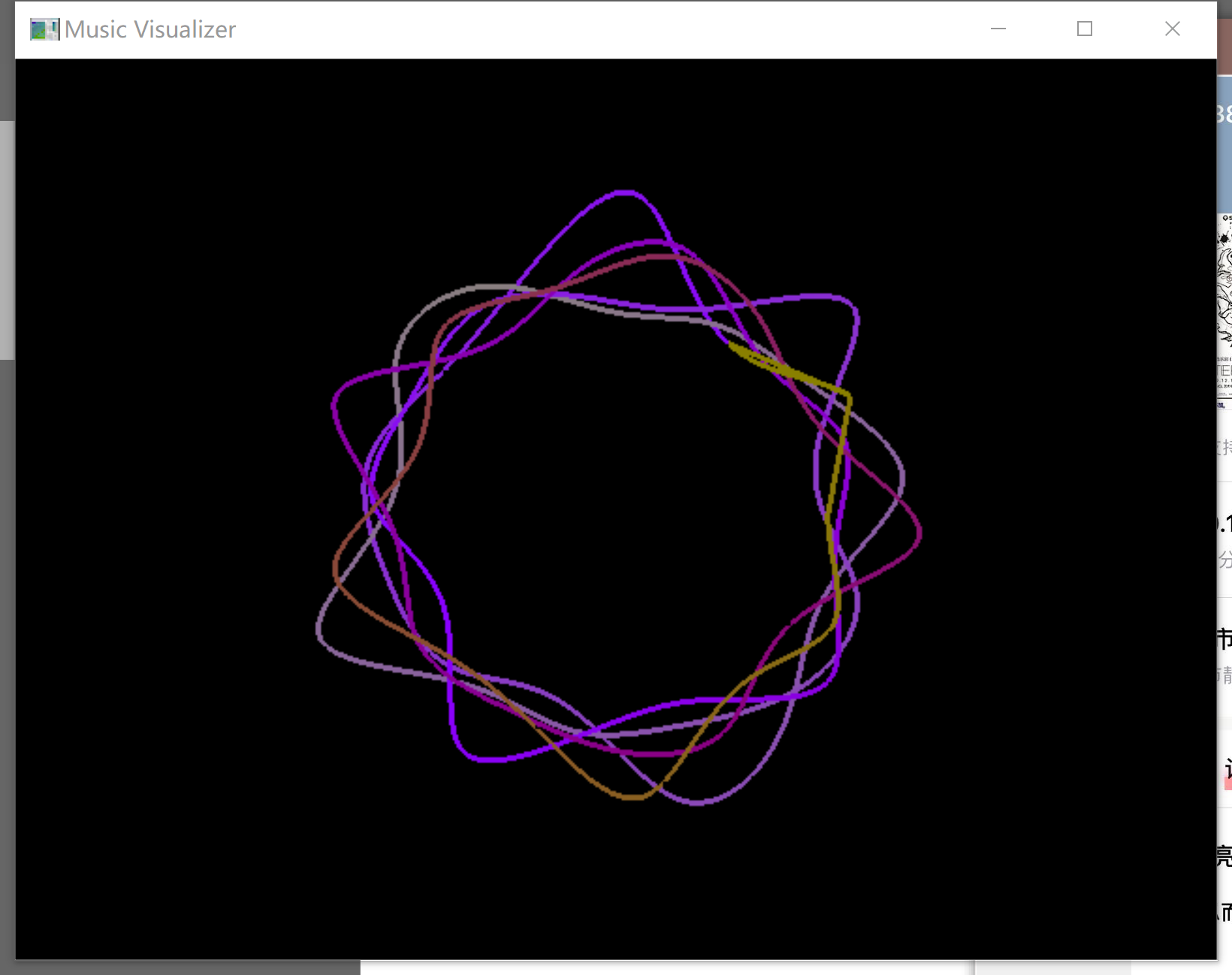
C++版本的优势主要在于所用的pcm格式采样频率高，因此显示出的样式更加平滑，此外由于数据整体一遍读入，可以方便的知道所显示帧所在的位置，因此红色的进度标识色更为准确。其劣势主要有，首先是输入格式。C++版本的音乐数值输入格式为pcm，而音乐播放格式为MP3，因此需要同时输入两个文件，并且需要提前使用ffmpeg工具将MP3转换为采样频率为16k的pcm格式。因此我将可视化展示的音乐通过硬编码的方式写入了代码中，无法通过参数指定需要可视化那个音乐。其次是，c++版本我虽然做了音画同步的优化，但是仍然会有累计误差造成的画面滞后现象。

Python版本的主要优势为使用python在许多数据处理上方便许多，其次是使用pyaudio可以很方便的达到音画同步的效果，最后是可以通过参数指定可视化的音乐。劣势在于计算速度低于c++，高强度的密集显示会造成音乐卡顿。Python版本的劣势在于采样频率没有c++版本那么高，取决于输入音频的采样频率，因此花纹的变化平滑度上逊色于c++版本。

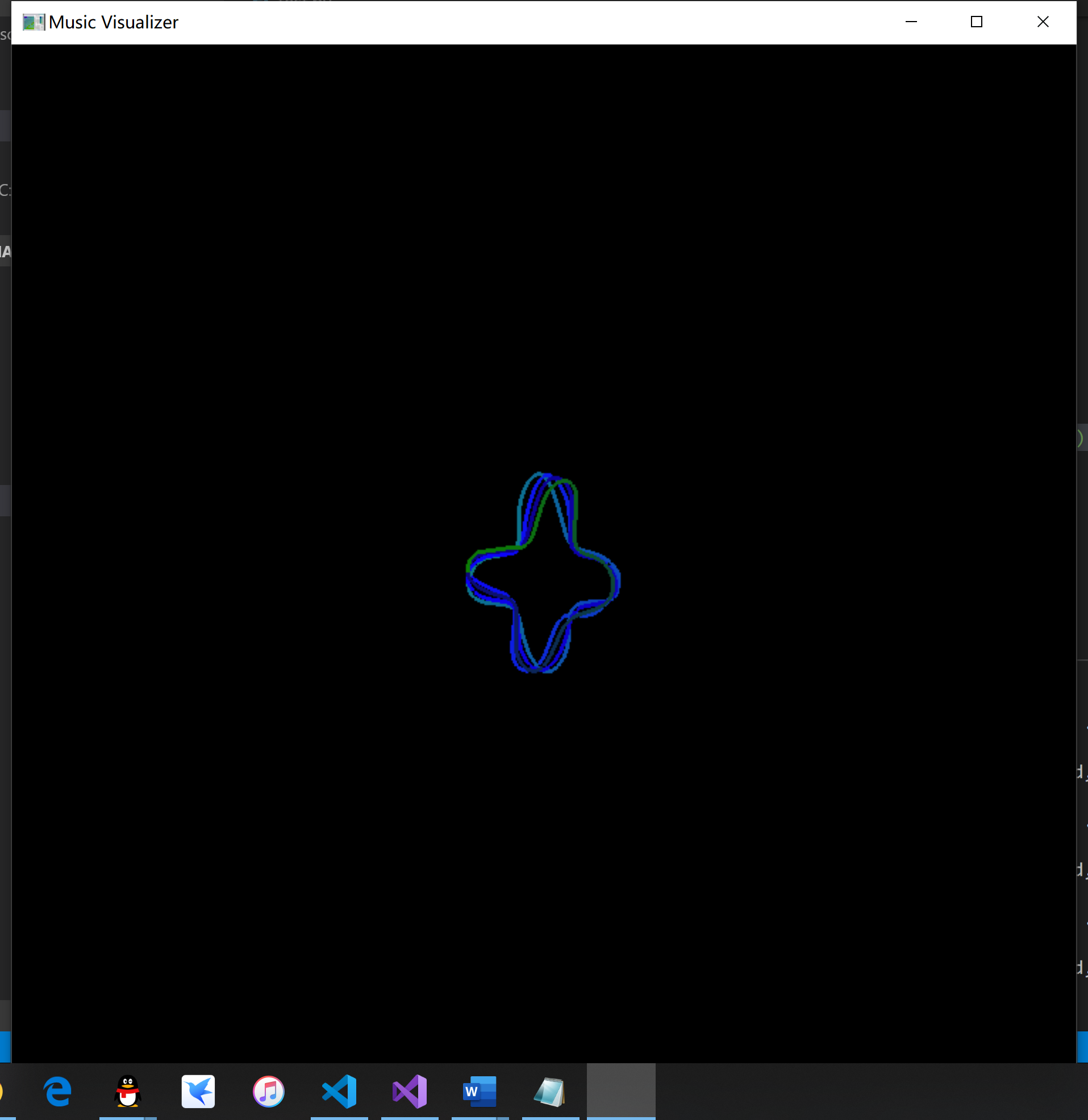
1. 效果展示
   1. C++版本
      1. 开始



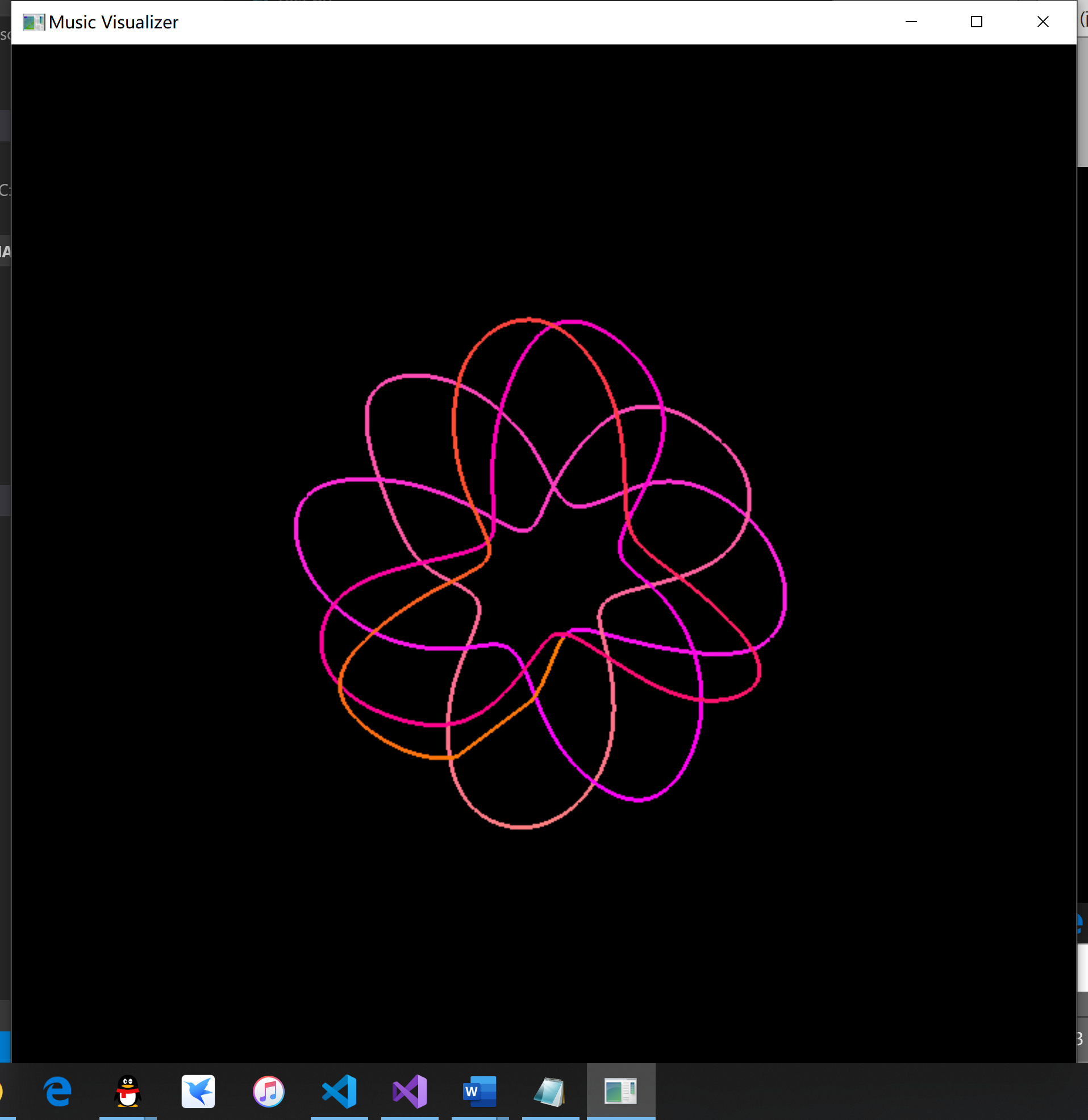
* + 1. 结束



* 1. Python版本
     1. 开始



* + 1. 结束

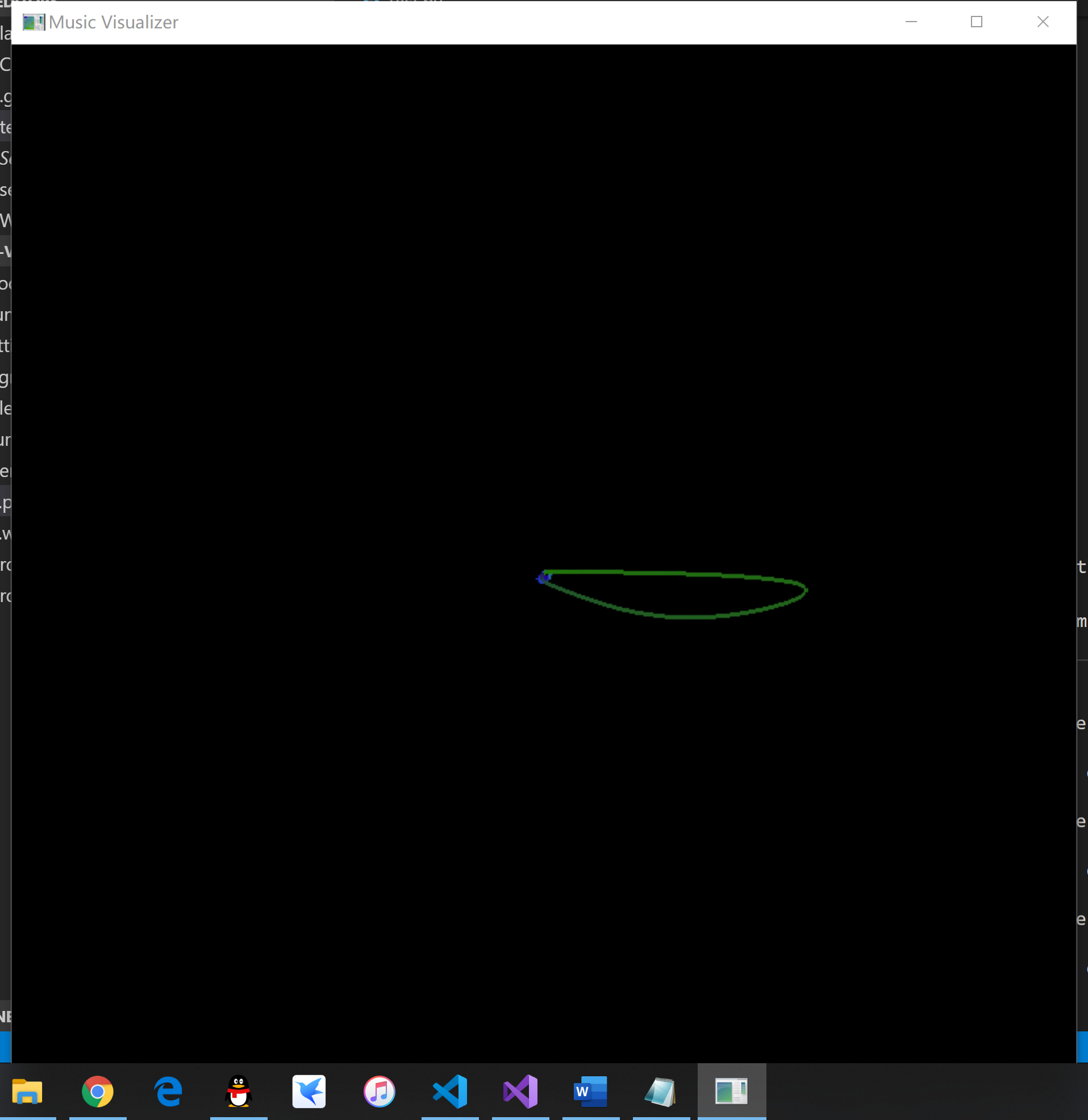


1. 讨论与思考
   1. 如何改进c++版本的音画同步问题？

现在主要的改进方法为使用精度更高的计时函数，然后取整记录误差，在误差满1ms的时候补偿回去。

* 1. 为什么不用快速傅里叶变换？

我尝试过使用快速傅里叶变换以后的频域数据进行可视化，但是频域数据的变化比较小，可视化出来的效果如下：



右边这个椭圆形一直会出现频幕上，可视化的视觉效果没有直接使用时域采样数据好，因此我最后选择使用时域的数据。

1. 实验感想

通过这次实验，我更深刻的理解了B样条生成算法，同时学到了很多音乐相关的采样、可视化的相关知识。