# 虚拟现实中的声觉计算

华炜 王锐 郑文庭

## 背景

- 在现实环境中，人类从外界获得的信息70%来源于视觉，而听觉约占15%～20%，是除了视觉以外的最重要的信息获取途径。
- 声音在虚拟现实系统中的作用，主要有以下几点：
  - 声音是用户和虚拟环境的另一种交互方法，人们可以通过语音与虚拟世界进行双向交流，如语音识别与语音合成等。
  - 数据驱动的声音能传递对象的属性信息。
  - 增强空间信息，借助于三维虚拟声音可以衬托视觉效果，使人们对虚拟体验的真实感增强，即使闭上眼睛，也知道声音来自那里。

## 声学基础

- 声音的基本特性
  - 声音的产生
    - 我们把正在发出声音的振动物体通常称为声源。
    - 物体振动或气流扰动而引起周围的空气或其它弹性介质发生波动的现象称为声波。
    - 声波所波及的空间范围称为声场。
  - 声音产生的三个条件
    - 存在声源并振动
    - 传播介质
    - 听觉感受

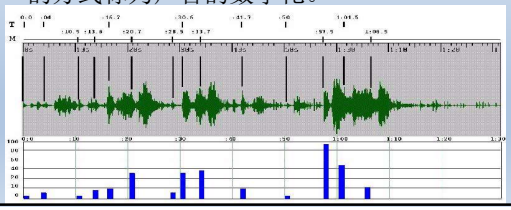


## 什么是声音

- 声音：50Hz-22,000Hz之间的压力波
  - 本质上而言是正弦波，具有波幅和频率等属性
- 声音的低频部分不仅仅被耳朵，也可被身体所感知
- 人感知到声音从某一地方发出
  - 不能精确地定位出音源

## 什么是数字声音？

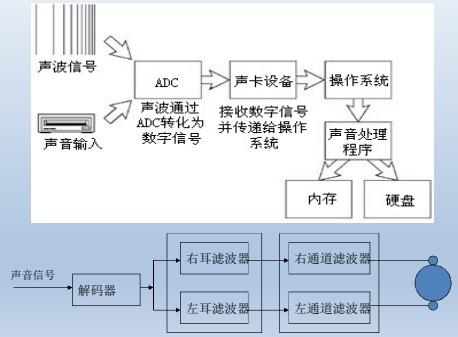
- 声音表现为波形, 可以记录、保存以及精确播放
- 采用数字技术将声音波形离散表示为数字的方式称为声音的数字化。



## 数字声音的编码存储

- 每秒钟CD品质的声音信号占据的空间是176KB，3分钟长度的歌曲容量是31MB
- 音频的压缩和解压缩：利用时空连贯性、查找表等技术。
  - 有损压缩：MPEG Layer 3 （MP3）：把部分并不需要的信息过滤掉，例如一些人耳听不到的高频率信号，或者一些无用的环境噪音。

## 声音的数字化处理





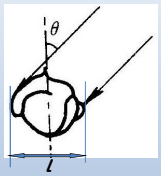


## 三维声音

- 考虑下列影响人感知声源位置的因素，把声音作三维定位，创造出三维的环境声响
  - 声源越远，音量越小
  - 从左边发出的声音将在左耳中产生较大的音量
  - 从左边发出的声音将首先到达左耳，左右耳朵相差1微秒左右
  - 从脑后传来的声音，与从前方传来的声音相比，有一定程度的减弱

## 立体声原理

- 声源平面定位
  - 时间差

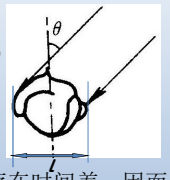



声音到达两侧耳壳处的时间差可近似为

$$\Delta t \approx \frac{l}{c}\sin\theta$$

## 立体声原理

- 声源平面定位
  - 时间差 $\Delta t \approx \frac{l}{c}\sin\theta$
  - 相位差




  - 由于传到两耳的声音存在时间差，因而也会产生相位差。对于频率为$f$的纯音，相位差与时间差有如下关系：$\Delta\varphi = 2\pi f \Delta t$

将$c = \lambda f$ 及$\Delta t \approx \frac{l}{c}\sin\theta$ 代入上式，可得$\Delta\varphi = \frac{2\pi l}{\lambda}\sin\theta$

## 立体声原理

- 声源平面定位
  - 时间差
  - 相位差
  - 头部相关传递函数(Head-related Transfer Function)
    - 声级差
    - 音色差

## 人类听觉定位机制

- 混响时间差和压力差是根据声波到听者双耳的距离差和声强衰减差而抽象出的定位方法，但因为忽略了外耳和头部的作用，导致虚拟声源的混响时间差和压力差分别相等的位置构成一个圆而不是一个点，进而形成一个混淆锥，造成听觉定位混淆问题。
- 头部相关传递函数在虚拟声音合成中是常用的定位方法，它是通过放于耳鼓处的微型麦克风测出来自不同方位声音的脉冲响应。
  - 由于听者个体和声音的传播过程中所涉及因素的多样性，我们很难用一个统一的解析表达式来定义HRTF.
  - Wenzel和Kistler提出了非个人HRTF来建立三维声音环境。因为HRTF的数据量大，并有许多未测量方位的HRTF，为此，一些研究者对HRTF进行简化和逼近。

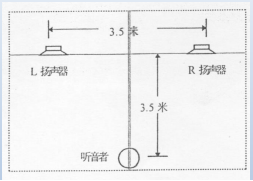
## 人类听觉定位机制

- 听觉定位是一个复杂的过程，除了以上几种定位方法外，还有视觉的辅助作用、反射定位、外耳的作用等。
  - 视觉系统与听觉系统不孤立。视觉系统可以帮助听觉系统矫正和验证声源的方位，训练听觉系统。
  - 现实生活中，达到耳道的声音有声源直接到达的，也有反射达到的。Douglas S. Brungart等人用声音的反射与声强的变化进行声音距离仿真。然而，较大的反射回声可能妨碍听觉定位，常常给人一个错觉。
  - 耳廓起非常重要的作用。当声波达到双耳时，首先接触耳廓，之后声音经耳廓不规则的反射后达到耳道，最后才能识别声音。耳廓的不规则形状，同时又因人而异，给模拟外耳进行声音定位带来很大的困难。由于采用耳机进行声音定位，所有屏蔽了耳廓的作用，产生了水平定位分辨率力较高，而上下和距离分辨率力较低的结果。听觉的混淆就是因为屏蔽了耳廓的作用，减少了声音信号对它的直接刺激。

## 人类听觉定位机制

- 每一种定位方式并不是孤立的，如Mark A. Ericson等人用HRTF和混响时间差来定位。然而，无论用什么方式，都存在问题，其原因是研究者并不完全理解人是怎样感知声音的，有待于听觉生理学和听觉心理学的进一步研究。

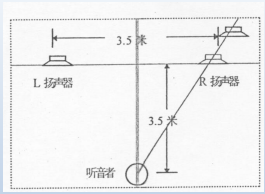
## 立体声定位




**1、**当左右扬声器发出相同强度的声音时，人听不见两只扬声器的存在，只感到声音是从两只扬声器的中点发出。这个点我们称之为“声像”。

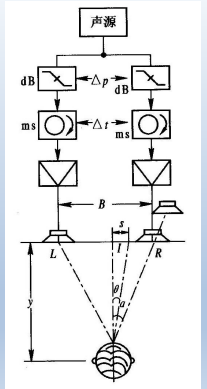
**2、**当增大其中一只扬声器的声压时，声像向这只扬声器的方向移动，当声压差大于15dB时，声像固定在这一方。

15



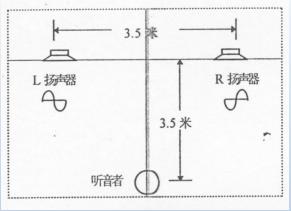

**3、**若将其中一只扬声器后移，但保证两扬声器的声音到达人耳时声压相等。人也分别不出两个声音，但由于时间差，声像移向近的一方；超过3毫秒，声像固定在近的一方。此时的声音仍只有一个；但有加厚的现象；若继续移远，加厚现象越来越显著，当两扬声器声音的时间差达50毫秒以上时，人就可以听到先后的两个声音，这时称为“回声现象”。

16




在双声道信号间引进强度差或时间差，可以人为地改变单个声像在扬声器基线上的位置。

17




4 当两扬声器相位相反，而且有一定的强度差时，声像会离开两扬声器的连线，出现在扬声器之外，甚至可能出现在听者的身后，这种现象称为界外立体声，它是一种特殊的立体声.

18

## 声音在游戏和VR中的重要性

- 可以根据声音收集我们周围物体的信息
- 在游戏中，除了视觉以外，最重要的信息获取手段就是声音
- 甚至比视觉更为集中、有效
- 通常可以指导视觉

## 目标1：营造沉浸感

- 空间的暗示使得玩家能判断自己的方位
  - 距离
  - 方向感
  - 当音源和听者移动时的更新
  - 连续性是关键!
- 在实体的世界中
  - 玩家不能听到大墙背后的声音

## 玩家的空间定位

空间的合成感





玩家自身的位置


大的石头房子


小的金属房子

玩家所在空间的材质


小的木质房子

## 目标2: 营造一种美学意境

- 能传递不同的心情和意境
  - 沉重的、压迫感的和狭窄的能造成热的空间感
  - 回音能造成开阔、冷的空间感
  - 其他的环境相关的声音操作

Demo:仙剑奇侠音乐

## 计算三维声音的声波跟踪法

- 根据声波在空间中的传输方式，从声源出发，发射声波并依据场景的三维几何属性进行描述，计算声波在空间中反射、折射等不同的传播途径与衰减程度，最终获得人耳所感知的各个方向的声音。
- 需要建立一个类似于图形绘制的几何引擎。
- 真正的三维几何声波传播计算耗费资源，因此游戏场景中用于声音计算的几何通常非常简单。在音效重要性高的游戏中（例如第一人称射击游戏），三维音效场景几何则要复杂一些。

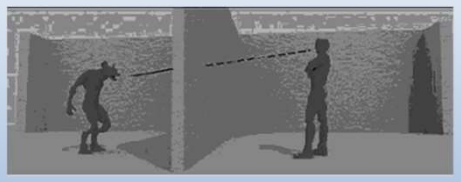
## 计算三维声音的声波跟踪法（续）

- 根据场景几何（线、三角形和四边形）的尺寸、类型和分布来计算它们对声音的影响。
- 构成声频几何的基本元素是声频多边形，它们的属性包括位置、大小、形状和材质属性。它的形状、位置与音源紧密相关，听众能够感觉到每个独立的声音是否被反射、穿越或环绕多边形发射，而材质属性则决定传输的声音被吸收或反射的比例。
- 在游戏引擎中，声频几何的位置和形状可以在游戏装载时从三维场景的几何表示转换而得。全局反射或者封闭的值可以通过参数进行设置。

## 遮挡特效

- 遮挡：声源和人耳不处在同一区间，因此声音的直接和间接传播都不存在。
- 遮挡效果原理上可以通过调低音量来实现，更加实际的方法是采用低通滤波（low-pass filter）算法。
- 在大部分情况下，可以采用将音源定位在不可见的障碍物后面，声波的传播通路被遮挡住，低通滤波的尺度则依据物体几何的参数（厚度）和材质决定。由于音源和和人耳之间没有直接的通道，也不存在音源的回波效果。

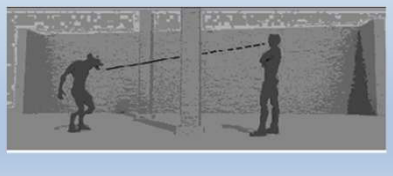
## 遮挡（Occlusion)特效（续）



声波的遮挡效果，直接和间接的传播通道都不存在

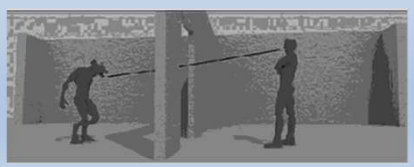
## 障碍特效

- 声源和人耳之间存在遮挡物，但声源和人耳在同一区间中，因此反射将被人耳感知。



## 排斥（Obstruction）特效

- 声源和听众在不同的房间，但它们有直接的接触，直接的声音可以传到听众，但反射的声音会发生失真（依据材料的厚度，形状和属性）



## 三维声音中的几个影响因素

- Doppler 效应
  - 快速移动的物体朝向听者，声音将被压缩，因此，具有更高的频率
  - 快速移远的物体，声音将被扩充，因此，具有更低的频率

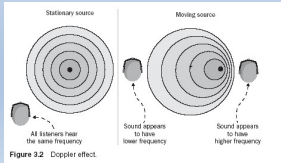


Figure 3.2 Doppler effect.

## 三维声音中的几个影响因素

- Distance Factor:实际场景中的尺寸与meters的比例。如英尺：0.3048
- Rolloff Factor:决定声音消失的快慢。0表示没有衰减。类似于绘制中的雾。(demo:carrevupdown.wav)
- 最小距离:

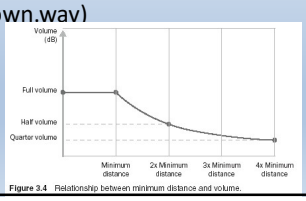


Figure 3.4 Relationship between minimum distance and volume.

## 三维声音中的几个影响因素

- Ambient Sounds:泛音
- Sound Cones:音锥，模拟有向声音（demo）

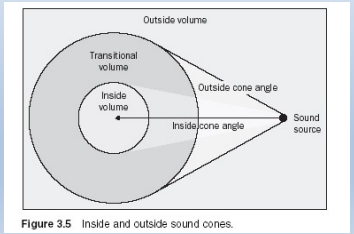


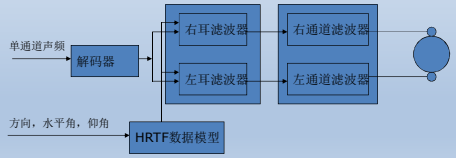
Figure 3.5 Inside and outside sound cones.

## 游戏中的音频编程

- 更好地贴切游戏中声音的意义
  - 声音做为载体在空间传播
  - 玩家和游戏智能利用声音做出gameplay决定
- 如果声音处理不好
  - 玩家将忽视声音的存在
  - 对于有遮挡物的室内场景，将给玩家错误的暗示

## 虚拟三维声音的合成

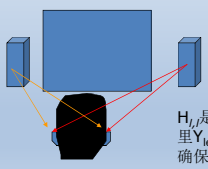
- 利用HRTF和立体声耳机合成的虚拟声源。虚拟声音的合成可以通过输入信号和一对HRTF的卷积来实现。




## 虚拟三维声音的合成

- 近年来出现了比较便宜的新一代PC三维声卡。使用DSP芯片处理立体声或5.1格式的声音，并且通过卷积输出真实的三维声音。

PC机喇叭装在监视器两侧，与监视器方向一致，面向用户。知道了用户头部的相对位置（面向显示器），就可以从查找表中检索得到HRTF。这样，只要用户保持处于最佳位置区域中，就有可能创建出在用户周围有许多扬声器的假象。



与耳机不同，基于喇叭的三维声音系统不能分离出每只耳朵听到的声音。

$$\begin{bmatrix} Y_{left} \\ Y_{right} \end{bmatrix} = \begin{bmatrix} H_{l,l} & H_{l,r} \\ H_{r,l} & H_{r,r} \end{bmatrix} \begin{bmatrix} S_{left} \\ S_{right} \end{bmatrix}$$

$H_{j,k}$是左喇叭与左耳之间的HRTF，以此类推。这里$Y_{left}$和$Y_{right}$是已知的，需要计算喇叭和输出以确保消除了干扰。

## 虚拟三维声音的合成

- 许多公司已经开发出了能处理6声道数字声音的三维声卡，并用两个喇叭播放出来。这些声卡都具有提供简化三维声音的能力，在视频游戏中得到了成功的应用。
- 产生三维声音所需的所有计算并不都是有声卡硬件完成的，有一部分需要由CPU来做，显示帧刷新率有一定的负面影响，降低约10%－30%。

## 常见的声音底层驱动

- OpenAL
  - OpenAL（Open Audio Library）是自由软件界的跨平台音效API。它设计给多通道三维位置音效的特效表现。其 API 风格模仿自OpenGL。
- DirectSound
  - DirectSound是DirectXAudio的一个较底层的部件，提供了丰富的接口函数，实现.wav格式的波形声音数据的播放控制

## DirectSound编程概述

- 基本流程是：
  - 建立一个IDirectSound接口（实际用于控制声音设备)
  - 建立一个对应于声音设备的IDirectSoundBuffer接口，然后调用函数载入声音文件。对该声音文件的控制，播放等都通过IDirectSoundBuffer在一个缓冲区（buffer）中进行
  - 每个声音文件都要有一个缓冲区

## DirectSound Devices

- DirectSound Devices 代表逻辑上的声音的控制设备，同时用于管理创建缓冲区
- 一般通过一个IDirectSound8 接口操作DirectSound Devices
- 函数DirectSoundCreate8
  - 创建并初始化一个IDirectSound8 的接口，如：

```
LPDIRECTSOUND8 lpds;//指向IDirectSound8的指针
HRESULT hr = DirectSoundCreate8(NULL, &lpds, NULL)); //调用后lpds即指向一个IDirectSound8接口
```

  - 在默认设备上创建一个DirectSound device

## SoundBuffer

- SoundBuffer即为一个声音缓冲区，用于操作特定的声音
- 创建SoundBuffer
  - 函数CreateBasicBuffer 返回一个指向IDirectSoundBuffer 接口的指针
  - 通过控制DSBUFFERDESC 的不同位可以改变Buffer的属性
- 声音的播放和停止：
  - IDirectSoundBuffer8::Play
  - IDirectSoundBuffer8::Stop

## 缓冲区的控制(一）

- 通过缓冲区控制声音
- 声音频率的控制：
  - IDirectSoundBuffer8::GetFrequency
  - IDirectSoundBuffer8::SetFrequency
- 声音音量的控制可以使用
  - IDirectSoundBuffer8::GetVolume
  - IDirectSoundBuffer8::SetVolume
- 声音播放位置的设定
  - IDirectSoundBuffer8::GetPan
  - IDirectSoundBuffer8::SetPan

3D声音不可设置pan

## 缓冲区的控制（二）

- IDirectSoundBuffer8::GetStatus
  - 缓冲区是在播放还是处于停止状态
- IDirectSoundBuffer8::GetFormat
  IDirectSoundBuffer8::SetFormat
  - 设定声音缓冲区的格式。对于不同的声音类型，有不同的结构体,以下代码设定wav声音的格式：

```
WAVEFORMATEX wfx;
wfx.wFormatTag = (WORD) WAVE_FORMAT_PCM;
pDSBPrimary->SetFormat(&wfx)
```

## 使用优先级

- 用于设定程序可以在多大程度上使用设备分三级：
  - DSSCL_NORMAL：不可写缓冲区,不可设定缓冲区格式；
  - DSSCL_PRIORITY：对设备有高优先级，可设定缓冲区格式，但是不可写；
  - DSSCL_WRITEPRIMARY：最高优先级，可写。

## 3D 缓冲区

- 在获得IDirectSoundBuffer8 接口lpDs3dBuffer之后，可以用下列方法获得3D缓冲区

```
LPDIRECTSOUND3DBUFFER8 lpDs3dBuffer;
HRESULT hr = lpDsbSecondary->QueryInterface(IID_IDirectSound3DBuffer8, (LPVOID *)&lpDs3dBuffer);
```

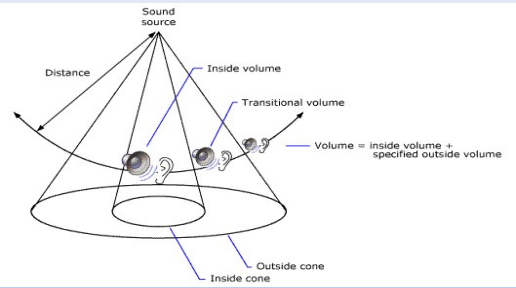
- 3D声音的控制中频率，位置等都要用三位坐标计算
- 设置声音的效果，如回声，如下列代码所示

```
DSEFFECTDESC dsEffect; //effect结构体
dsEffect.guidDSFXClass = GUID_DSFX_STANDARD_ECHO; //回声
pDSBuffer->SetFX(1, &dsEffect, dwResults)
```

## Sound Cones （声锥）

- 对于定向的声音，每个3D buffer逻辑上都有一个sound cone（声锥），此时把声音视为点声源。
- 声锥分内外两层，内层声锥中的声音按照一般方法计算，即不考虑声锥的存在。
- 外层声锥中的声音按照程序中设定的因子减弱，即可以由开发者自由控制
- 在两层中间，声音按照角度的增大减弱，即是渐变的
- 一般使用时如果没有显式指定，声音会是不定向（全向）的，此时内，外层声锥的角度是360度，即声锥不会起作用。

## Sound Cone图解




## DxUTSound文件提供的实用类

- 微软提供了几个实用类来简化编程，这些类在DxUTSound中定义
- CSoundManager 封装了前述IDirectSound8接口及相应的操作，如提供了初始化该接口的函数initialize，以及加载文件到CSound类的函数

```
Create( CSound** pSound，LPWSTR strWaveFileName,…);
```

- CSound 封装了前述IDirectSoundBuffer类对象和相应的操作，可以通过控制这个类来控制buffer，如播放函数play(),停止函数stop()等。使用Get3DBufferInterface函数得到对应的3D buffer
- CStreamingSound 封装了用于把一个大wav文件load到一个buffer中的操作。
- CWaveFile 封装了对wav文件读写的操作，并且可以获取文件的大小和格式

## 播放3D声音的完整流程（一）

- 调用CWaveFile::Open( LPWSTR strFileName…)打开一个wav文件，strFileName为文件名。打开后可以调用GetFormat()察看格式。
- 调用CSoundManager::Create( CSound** ppSound, LPWSTR strWaveFileName,…)把一个名为strWaveFileName的文件load到ppsound封装的buffer里去
- CSound::Play函数可以播放CSound所封装的buffer，CSound::Stop用于停止
- Play中可以使用DSBPLAY_LOOPING设定循环播放声音

## 播放3D声音的完整流程（二）

- CSound::Get3DBufferInterface获取3D buffer
- GetAllParameters和SetAllParameters设置3D缓冲区的参数；
- 代码参考：

```
CSoundManager*    g_pSoundManager      = NULL;
CSound*           g_pSound             = NULL;
LPDIRECTSOUND3DBUFFER  g_pDS3DBuffer   = NULL;
```

## 播放3D声音的完整流程（三）

```
hr = g_pSoundManager->Initialize( hDlg, DSSCL_PRIORITY );
//初始化并设定cooperate level
hr = g_pSoundManager->Create( &g_pSound, strFileName,
DSBCAPS_CTRL3D, guid3DAlgorithm );
//把strFileName中的内容load到g_pSound里
hr = g_pSound->Get3DBufferInterface( 0, &g_pDS3DBuffer )
// 得到3D buffer
g_dsBufferParams.dwMode = DS3DMODE_HEADRELATIVE;
g_pDS3DBuffer->SetAllParameters( &g_dsBufferParams,
DS3D_IMMEDIATE );
//设定新参数
hr = g_pSound->Play( 0, dwFlags )
// 播放音乐，dwFlags==DSBPLAY_LOOPING就可以循环播放声音
```

## 常见的声音引擎

- BASS
- MikMod
- FMOD
- ModPlug
- Miles Sound System
- EAX

## 数字音乐创作-MIDI

- MIDI - Musical Instrument Digital Interface（乐器数字界面）
  - 存储的不是声音信号，而是各种乐器的发音命令，播放时系统根据这些命令合成乐曲。
  - midi文件的优点是非常小，可以满足长时间音乐的需要。
- 用于控制音乐合成器
  - 合成细节由合成器决定
- MIDI数据
  - 仅仅是一个事件的列表，描述了一个声音卡或其他播放设备要产生某种声音的特定的步骤
  - 每一个描述乐器演播的动作的字都赋给一个特定的二进制代码。要奏响一个音符，你要发出一个“音符开”（Note On）消息，然后对该音符赋以一个“速度”，用以判断该音符能奏多响。
  - 其他控制包括选择哪件乐器演奏、混合和平移声音以及控制电子乐器等。

## 数字音乐创作- MIDI

- MIDI协议收录了常用的16类乐器，其中第16类是音效，如电话铃、鸟叫、海浪等。每类各8种音色，一共有128种音色。另外还收录了一组打击乐音色，并规定了每件打击乐器在键盘上的键位。这份标准MIDI（General MIDI）音色排序表简称GM音色表。
- GM音色表统一了数字乐器的标准，但128种音色毕竟太少，所以又有了GS和XG标准的音色表，它们的音色增加了很多。现在高档的合成器或音源中都有成百上千种音色，而且音质越来越好。

## 音频处理软件

| 软件 | 特点 |
| --- | --- |
| GoldWave | 方便上手，单轨操作 |
| Cool Edit Pro | 插件多，支持多轨操作，易出错 |
| Adobe Audition | 专业音频编辑和混合环境<br>严格，不易出错 |
| Cakewalk | 多种音色，作曲，伴奏 |
| 唱吧 | 使用简单、与社交结合 |

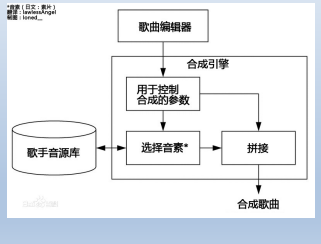
55

## VOCALOID

- VOCALOID是由YAMAHA集团发行的歌声合成器技术以及基于此项技术的应用程序，由剑持秀纪（Kenmochi Hideki）率领研究小组于西班牙的庞培法布拉大学开发。最初该项目并未有商业目的，YAMAHA集团帮助其商业化并最终变为了产品“VOCALOID”。该产品可以令用户通过输入歌词和旋律的方式让软件生成唱词，配合加载伴奏数据来完成整首音乐制作，在制作过程中无需任何新的歌手提供声音资料。
- 2000年3月开始开发Vocaloid。VOCALOID4于2014年11月20日在日本发布。

## VOCALOID系统架构

- 歌曲编辑器（Score Editor）
- 音源库（Singer Library）
- 合成引擎（Synthesis Engine）




## VOCALOID参数介绍

- 音速（VEL、Velocity）
- 力度（强弱法）（DYN、Dynamics）
- 呼吸声（BRE、Breathiness）
- 明亮感（BRI、Brightness）
- 透明感（CLE、Clearness）
- 口的开合度（OPE、Opening）
- 性别参数（GEN、Gender Factor）
- 滑音位置（POR、Portamento Timing）
- 音高折曲（PIT、Pitch Bend）
- 音高折曲敏感度（PBS、Pitch Bend Sensitivity）

## utau歌声合成软件

- UTAU是一款由饴屋/菖蒲氏开发的免费的歌声合成软件，2010年1月份(v0.2.60版之后)改为共享软件。
- UTAU的正式名称为“歌声合成工具UTAU”，是在人力VOCALOID下诞生的产物。

## 其他的进一步话题

- 音乐心理学
- 语音识别技术
- 商业化/产业化
  - –音乐网站/版权
  - –卡拉OK
- ……