# OpenGl应用

## 云信拉流端(二)

# 课程目标

学习通过底层操作系统服务(OpenSL ES学习)

手写实现音视频同步

# 什么是OpenGL?

- 音视频播放

Open Graphics Library

图形领域的工业标准，是一套跨编程语言、跨平台的、专业的图形编程(软件)接口。它用于二维、三维图像，是一个功能强大，调用方便的底层图形库。

与硬件无关。可以在不同的平台如Windows、Linux、Mac、Android、IOS之间进行移植。因此，支持OpenGL的软件具有很好的移植性，可以获得非常广泛的应用。

- OpenGL ES 1.0 和 1.1 ： Android 1.0和更高的版本支持这个API规范。
  OpenGL ES 2.0 ： Android 2.2(API 8)和更高的版本支持这个API规范。
  OpenGL ES 3.0 ： Android 4.3(API 18)和更高的版本支持这个API规范。
  OpenGL ES 3.1 : Android 5.0(API 21)和更高的版本支持这个API规范

# Android OpenGL ES

- GLSurfaceView

继承至SurfaceView，它内嵌的surface专门负责OpenGL渲染。

管理Surface与EGL

允许自定义渲染器(render)。

让渲染器在独立的线程里运作，和UI线程分离。

支持按需渲染(on-demand)和连续渲染(continuous)。

- OpenGL是一个跨平台的操作GPU的API，但OpenGL需要本地视窗系统进行交互，这就需要一个中间控制层， EGL就是连接OpenGL ES和本地窗口系统的接口，引入EGL就是为了屏蔽不同平台上的区别。

# OpenGL 专业名词

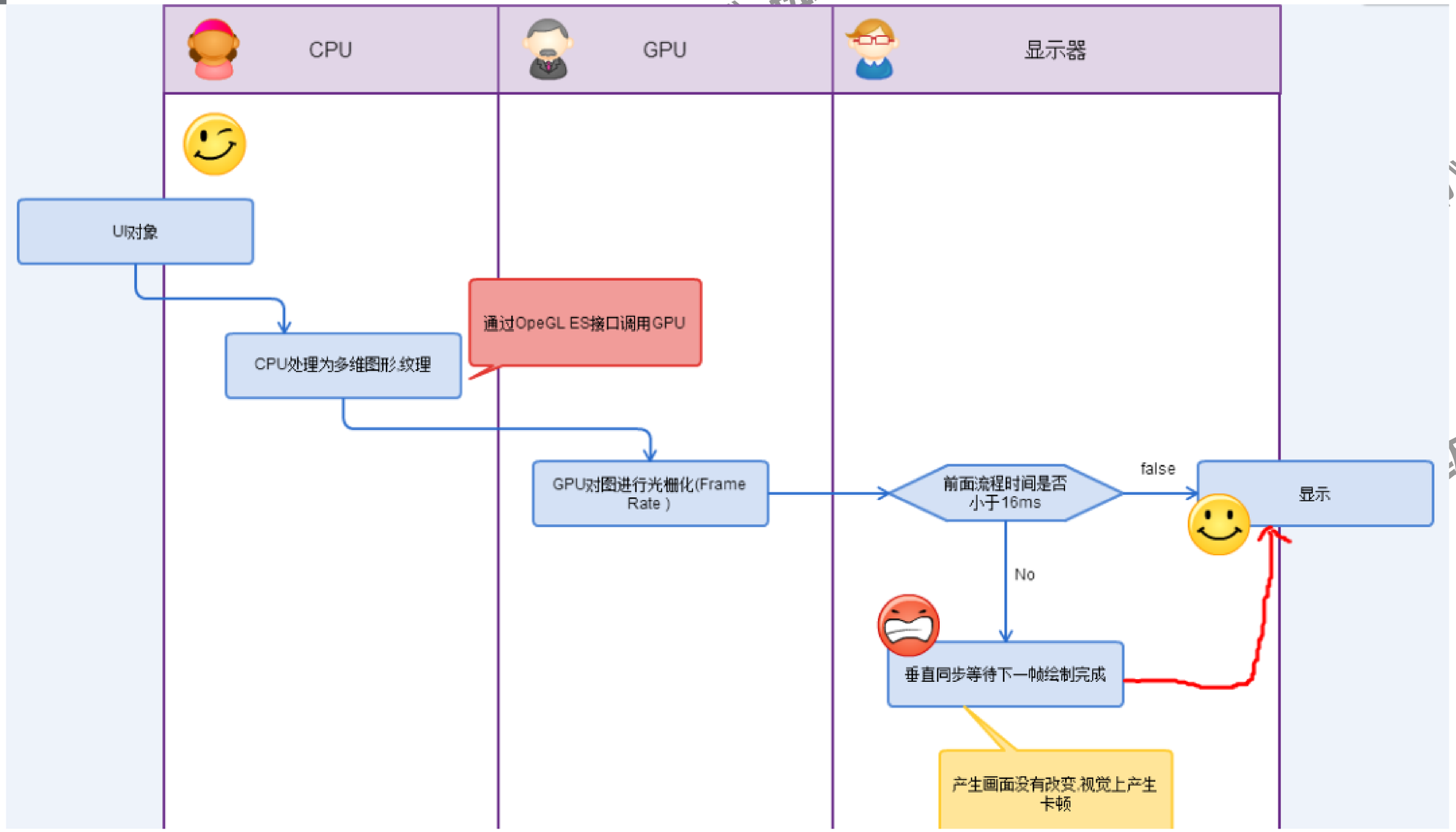
01 顶点着色器

02 片元着色器

03 坐标系

04 着色器语言

# Android渲染机制分析




—-> UI对象

—->CPU处理为纹理 纹理

—->通过OpeGL ES接口调用GPU

—->GPU对图进行光栅化

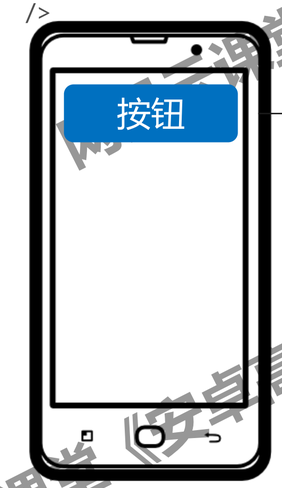
—->硬件时钟(Refresh Rate)

—->垂直同步

—->投射到屏幕

# Android绘制原理

```
<Button
    android:layout_width="match_parent"
android:layout_height="match_parent"
    android:text="按钮"
    />
```




LayoutInfalte r

Button对象

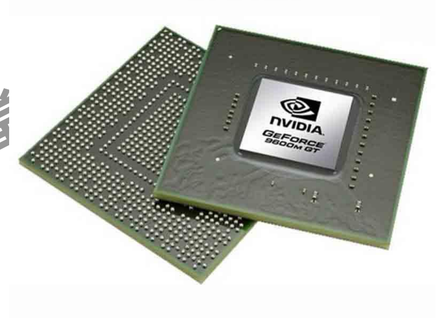
对象含有left top Right bottom,width height信息

CPU计算

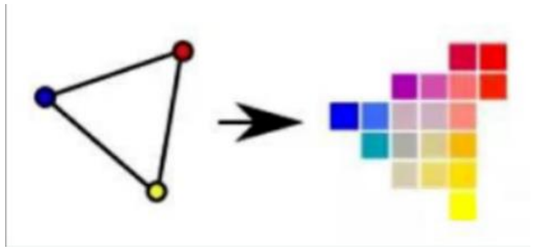
处理成多为的向量图形


将向量图像交给GPU

GPU负责填充



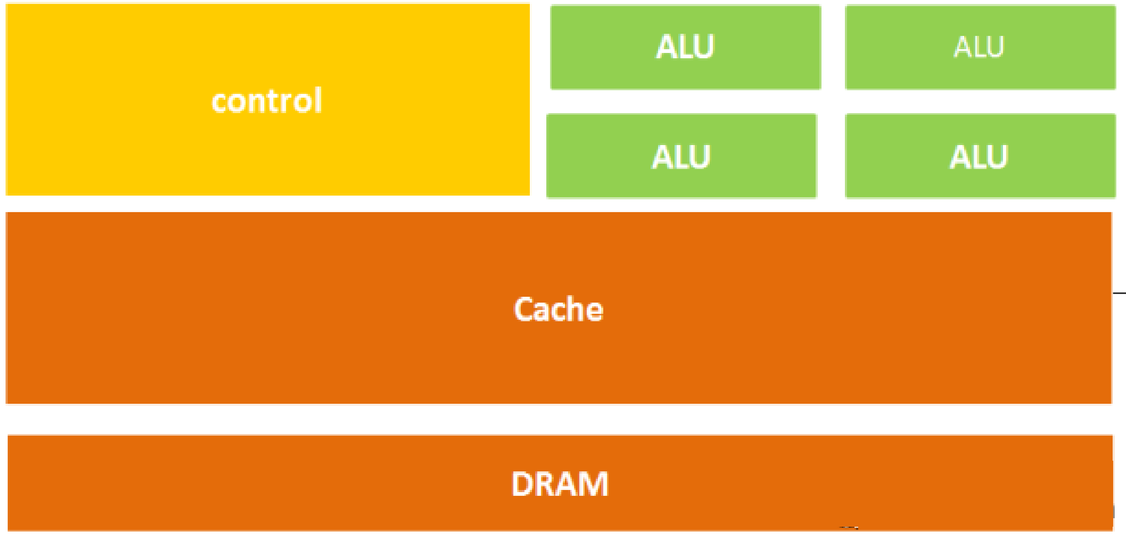

栅格化

定义: 栅格化是将响亮图形格式表示的图像转换成像素位图图像用以显示在显示器中

# GPU与CPU区别

- CPU( Central Processing Unit, 中央处理器)就是机器的“大脑”，也是布局谋略、发号施令、控制行动的“总司令官”
- CPU的结构主要包括运算器（ALU, Arithmetic and Logic Unit）、控制单元（CU, Control Unit）、寄存器（Register）、高速缓存器（Cache）和它们之间通讯的数据、控制及状态的总线。





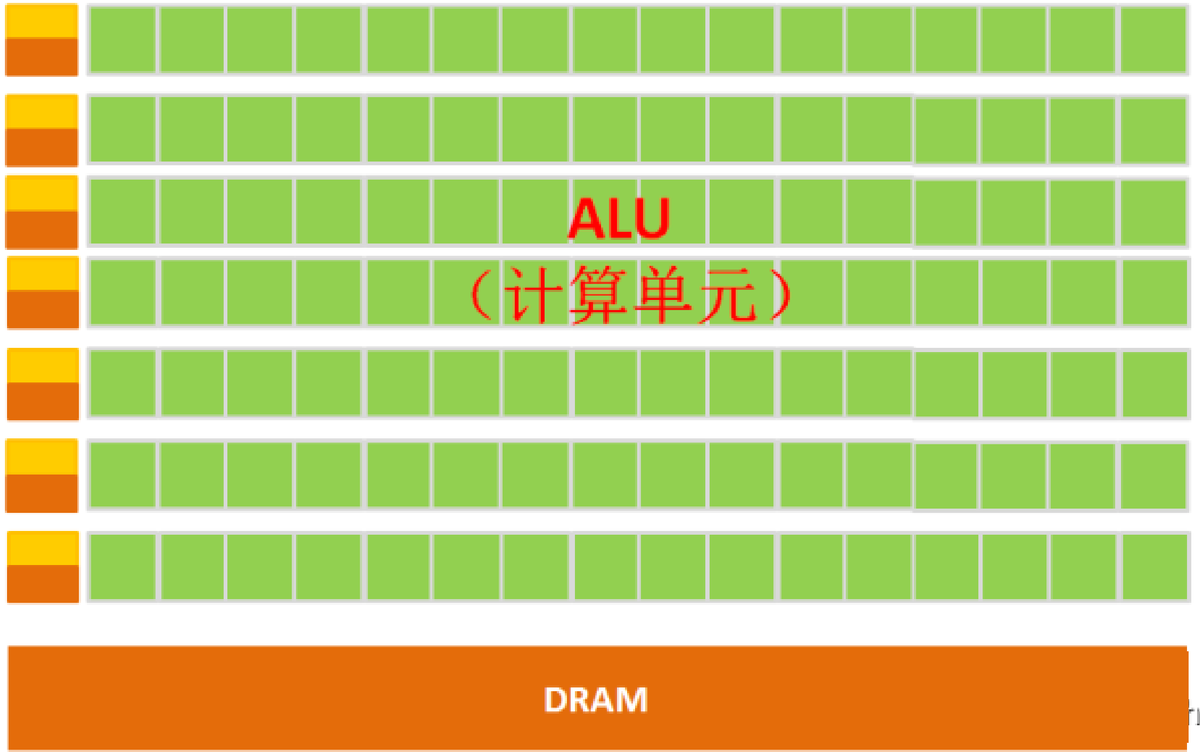
计算单元主要执行算术运算、移位和转换

存储单元主要用于保存运算中产生的数据以及指令等；

控制单元则对指令译码，并且发出为完成每条指令所要执行的各个操作的控制信号。

# GPU

- GPU全称为Graphics Processing Unit，中文为图形处理器，就如它的名字一样，GPU最初是用在个人电脑、工作站、游戏机和一些移动设备（如平板电脑、智能手机等）上运行绘图运算工作的微处理器。




GPU无法单独工作，必须由CPU进行控制调用才能工作。CPU可单独作用，处理复杂的逻辑运算和不同的数据类型，但当需要大量的处理类型统一的数据时，则可调用GPU进行并行计算

→

对图形进行绘制 可以理解成对像素点的赋值

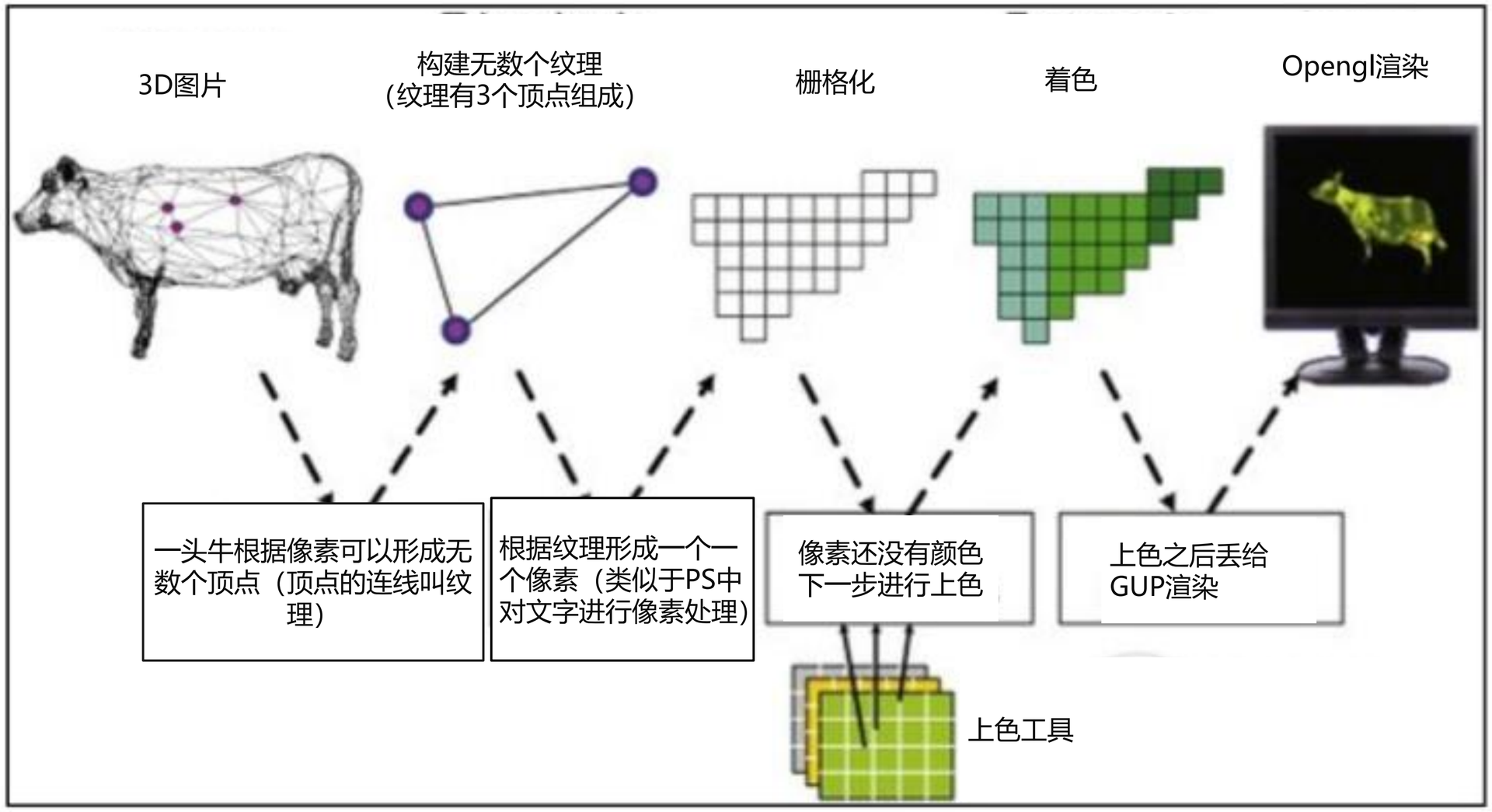
比如我要将初始颜色黑色0x000000变成红色 0xff0000

通过0x000000& 0xff0000即可，同样像素的颜色绘制通过运算得到的

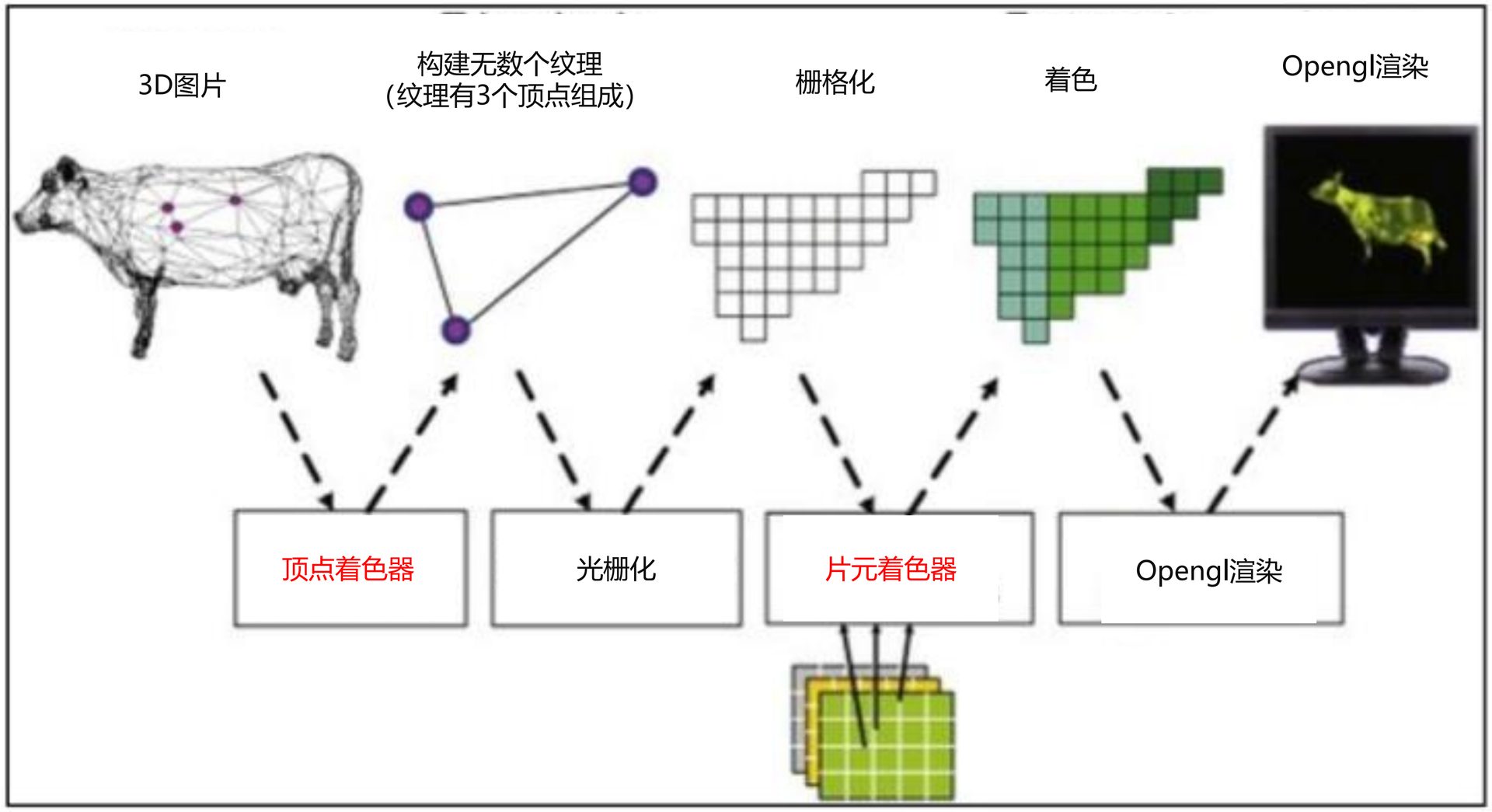
而GPU中又有很多的运算器ALU 他天生适合渲染图形

# 为什么会出现着色器





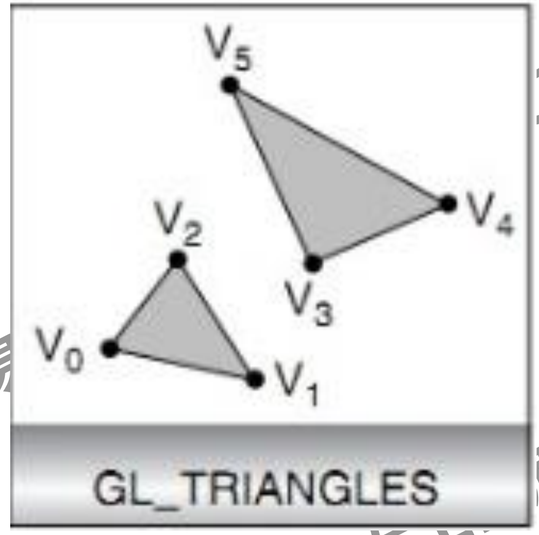
# 为什么会出现着色器



# 绘制模式

- GL_TRIANGLES

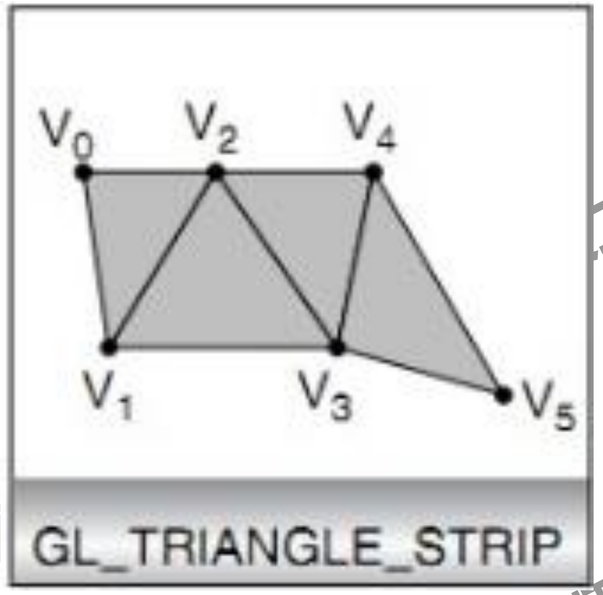
GL_TRIANGLES是以每三个顶点绘制一个三角形。第一个三角形使用顶点v0,v1,v2,第二个使用v3,v4,v5,以此类推。如果顶点的个数n不是3的倍数，那么最后的1个或者2个顶点会被忽略。



# 绘制模式

- GL_TRIANGLE_STRIP

构建当前三角形的顶点的连接顺序依赖于要和前面已经出现过的2个顶点组成三角形的当前顶点的序号的奇偶性（如果从0开始）：




如果当前顶点是奇数：

组成三角形的顶点排列顺序：T = [n-1 n-2 n].

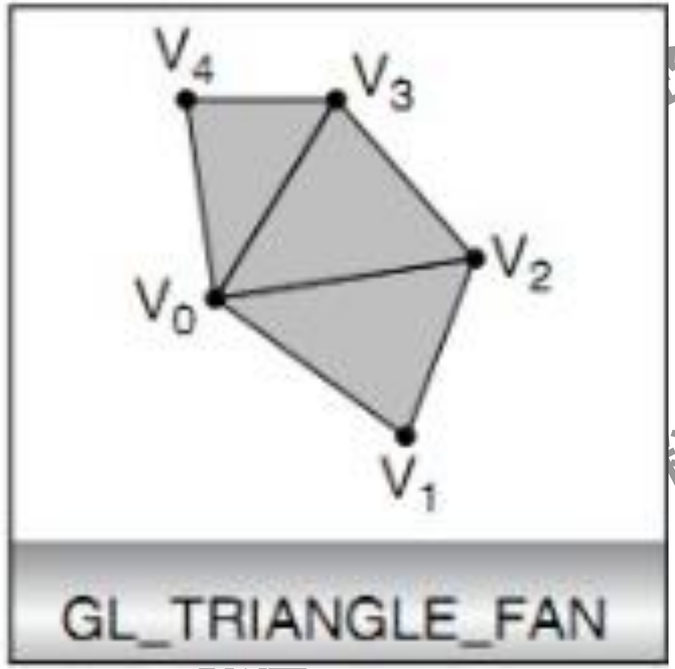
如果当前顶点是偶数：

组成三角形的顶点排列顺序：T = [n-2 n-21 n].

# 绘制模式

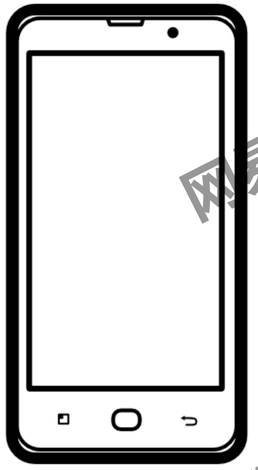
- GL_TRIANGLE_STRIP

GL_TRIANGLE_FAN绘制各三角形形成一个扇形序列，以v0为起始点，（v0，v1，v2）、（v0，v2，v3）、（v0，v3，v4）。



# 为什么会出现着色器

- 优势



# OpenSl ES 播放的优势

- 优势

（1）C 语言接口，需要在 NDK 下开发，能更好地集成在 native 应用中

（2）运行于 native 层，播放 速度极快，延时低  对于音视频同步中以音频为准的最为适合不过

（3）减少java层频繁的反射调用，如果通过AudioTrack播放需要将解码后得pcm数据反射java 增加开销

# 如何学习底层直接调用Android服务

- 地址 https://github.com/googlesamples/android-ndk

Android NDK samples with Android Studio http://developer.android.com/ndk

1,289 commits | 5 branches | 0 releases | 48 contributors | Apache-2.0

Branch: master ▾ | New pull request | Find File | Clone or download ▾

ggfan added more instruction to build native-midi with studio (#630) — Latest commit 86ceeb2 on 15 Mar

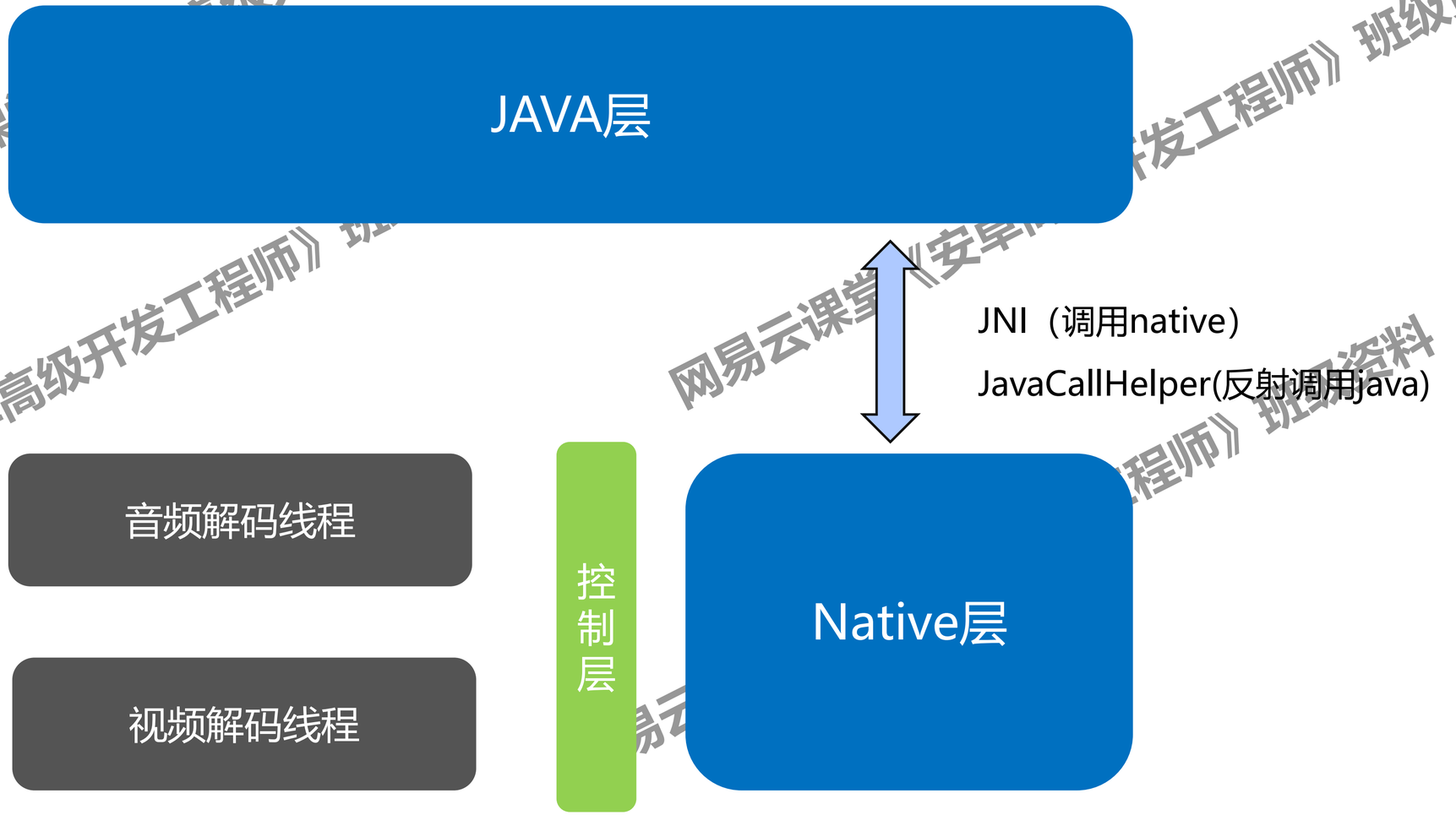
| | | |
|---|---|---|
| .ci_tools | CI script update: no need to install with constraintLayout 1.0.2+ (#618) | 4 months ago |
| audio-echo | Update build script to version 3.3.1 (#622) | 4 months ago |
| bitmap-plasma | Update build script to version 3.3.1 (#622) | 4 months ago |
| builder | Update build script to version 3.3.1 (#622) | 4 months ago |
| camera | Update build script to version 3.3.1 (#622) | 4 months ago |
| display-p3 | Update build script to version 3.3.1 (#622) | 4 months ago |
| endless-tunnel | Update build script to version 3.3.1 (#622) | 4 months ago |
| gles3jni | Update build script to version 3.3.1 (#622) | 4 months ago |
| hello-cdep | Update build script to version 3.3.1 (#622) | 4 months ago |
| hello-gl2 | Update build script to version 3.3.1 (#622) | 4 months ago |
| hello-jni | Update build script to version 3.3.1 (#622) | 4 months ago |
| hello-jniCallback | Update build script to version 3.3.1 (#622) | 4 months ago |
| hello-libs | Update build script to version 3.3.1 (#622) | 4 months ago |
| hello-neon | Update build script to version 3.3.1 (#622) | 4 months ago |
| kotlin-app | Update build script to version 3.3.1 (#622) | 4 months ago |
| native-activity | Update build script to version 3.3.1 (#622) | 4 months ago |
| native-audio | Update build script to version 3.3.1 (#622) | 4 months ago |
| native-codec | Update build script to version 3.3.1 (#622) | 4 months ago |
| native-media | Update build script to version 3.3.1 (#622) | 4 months ago |
| native-midi | added more instruction to build native-midi with studio (#630) | 3 months ago |
| native-plasma | Update build script to version 3.3.1 (#622) | 4 months ago |
| nn_sample | Update build script to version 3.3.1 (#622) | 4 months ago |
| other-builds | Update build script to version 3.3.1 (#622) | 4 months ago |
| san-angeles | Update build script to version 3.3.1 (#622) | 4 months ago |
| sensor-graph | Update build script to version 3.3.1 (#622) | 4 months ago |

- Camera 底层操作相机
- bitmap-plasma 底层直接渲染Bitmap
- native-activity native层操作Activity
- native-audio native层播放音频
- native-media native层控制多媒体
- native-plasma Bitmap中细胞质效果
- nn_sample Android中网络库
- sensor-graph native层获取地图经纬度
- Webp native层加载和渲染webp

# 使用OpenSL ES流程图


创建音频引擎　设置混音器　创建播放器（录音器）　设置缓冲队列和回调函数　设置播放状态　启动回调函数

# 云信课堂项目框架

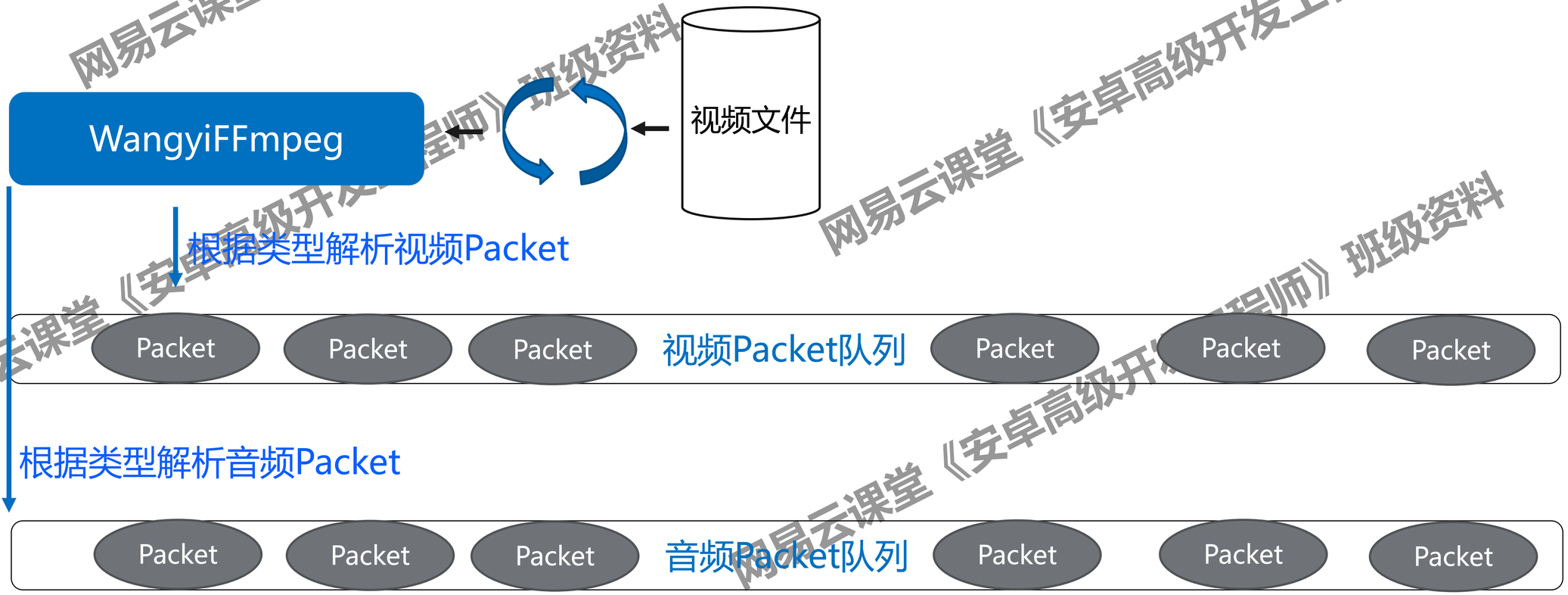


# 控制层

作用:　　初始化FFmpeg参数设置

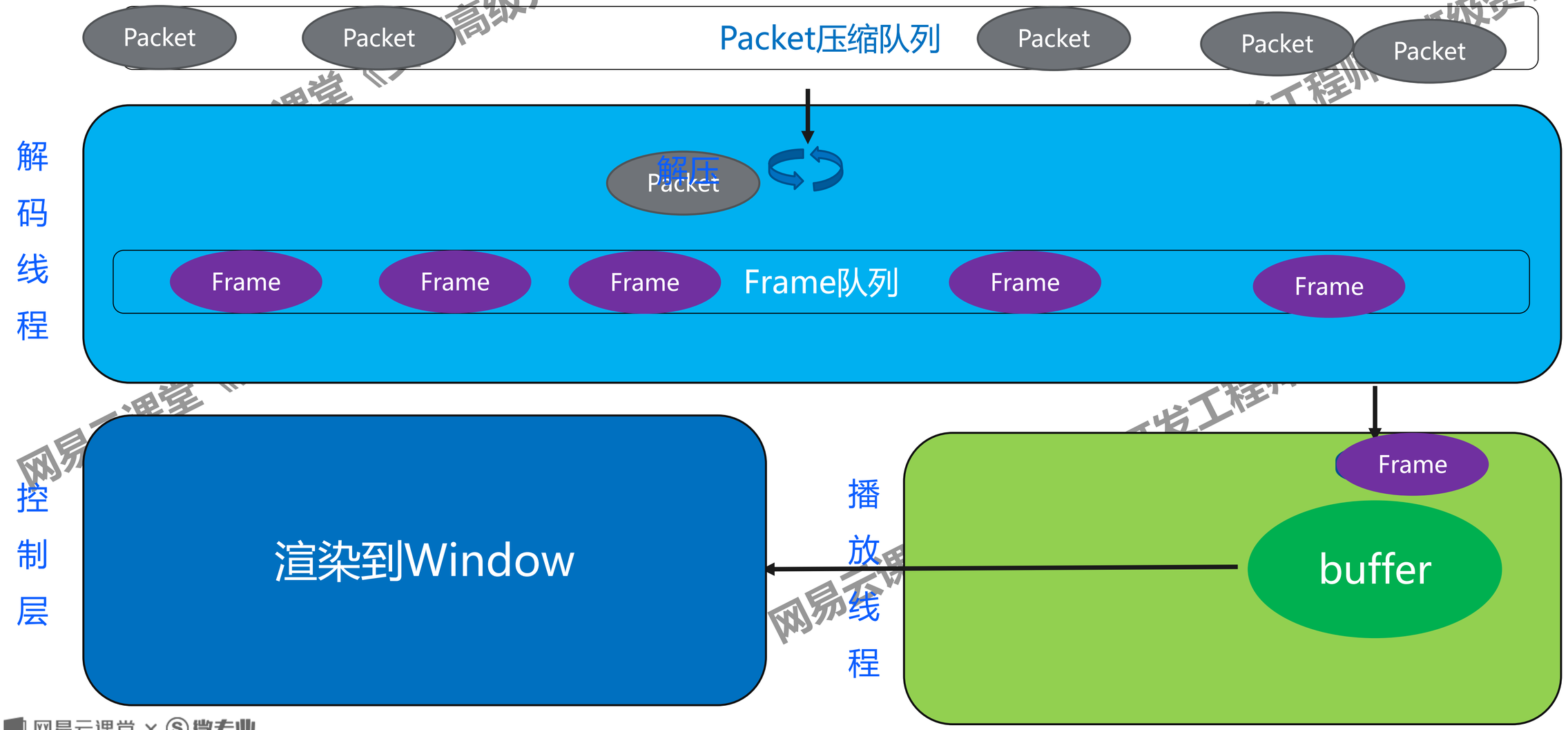
控制进度播放(开始播放,暂定，调节播放)

从视频文件(视频流)解析Packet



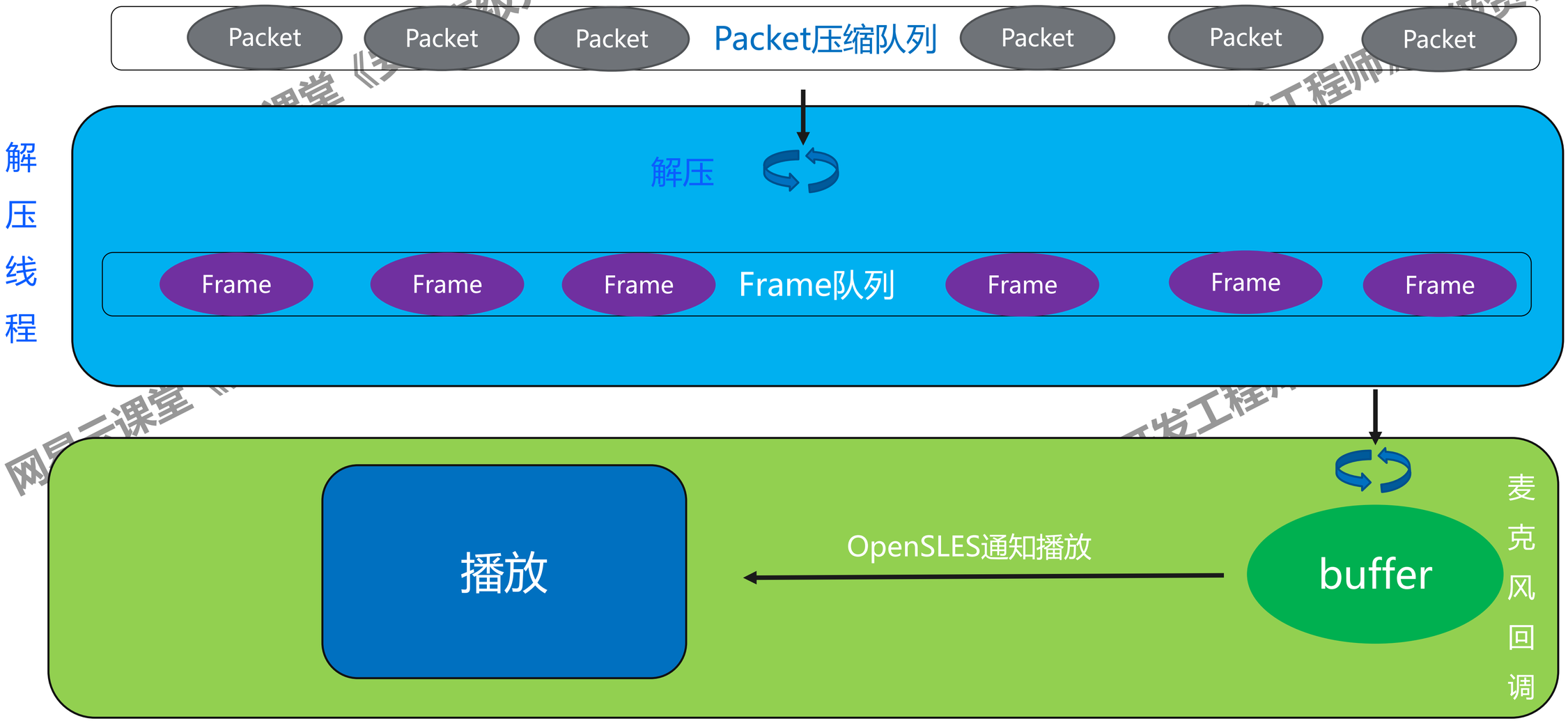
# 视频层

作用: 解码Packet 转换成Frame



# 音频层

作用: 解码Packet 转换成Frame



# 音视频同步

问题：音视频不同步

原因：由于FFmpeg是靠CPU来进行解码，音视频解码器都是异步进行解码。

# 音视频同步解决方案

- 音视为准

以音频为准，视频与音频同步，音频只管播放，视频有超前和延后。如果超前放慢速度，如果延后 减少视频播放等待时间，如果延后很严重，队列积累时间过长，这直接丢弃，进入最新解码帧

- 视频为准

视频画面每次循环不变，音频根据视频来 延迟或等待

- 自定义时间为准

根据开始加载视频时间为0时间，后面的音频帧和视频帧依赖自定义时间进行同步

# AVRational

```
typedef struct AVRational{
    int num; ///< numerator
    int den; ///< denominator
} AVRational;
```

AVRational这个结构标识一个分数

1 num为分数

2 den为分母。

# AVRational

- 意思

DTS，Decoding Time Stamp，解码时间戳，告诉解码器packet的解码顺序。

PTS，Presentation Time Stamp，显示时间戳，指示从packet中解码出来的数据的显示顺序。

音频中二者是相同的，但是视频由于B帧（双向预测）的存在，会造成解码顺序与显示顺序并不相同，也就是视频中DTS与PTS不一定相同。

# AVRational time_base;

- 含义是时间 刻度

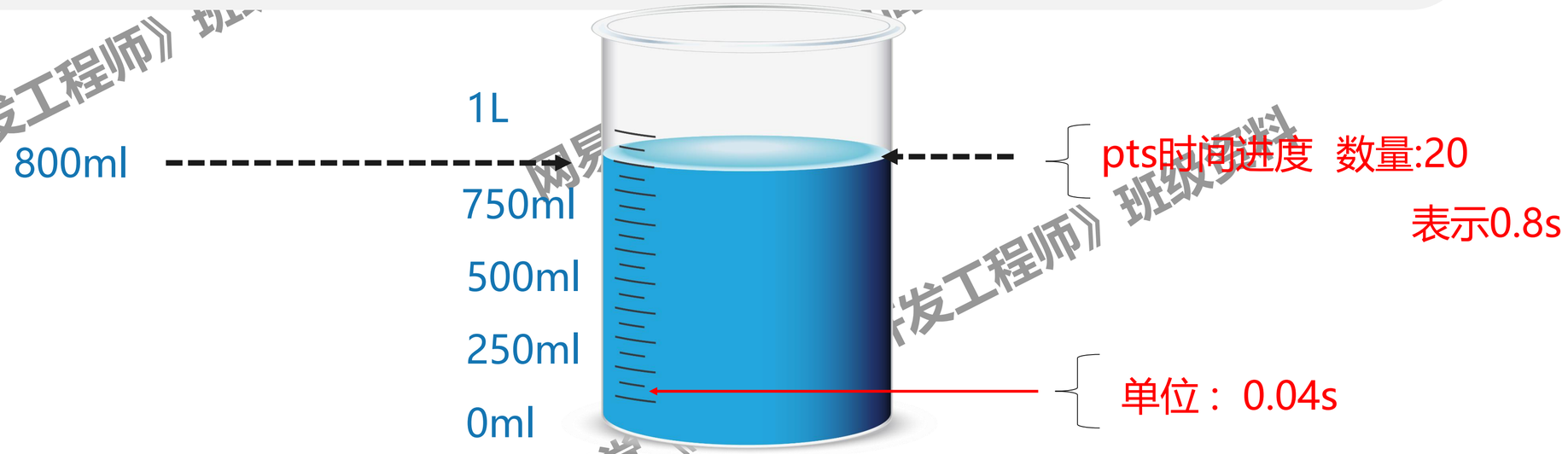
```
AVStream->AVCodecContex->time_base     {1,100}
```

音频中二者是相同的，但是视频由于B帧（双向预测）的存在，会造成解码顺序与显示顺序并不相同，也就是视频中DTS与PTS不一定相同。

# 什么是单位

单位是人为定义的，如什么是1ml　你可以自己定义1ml的实际大小，国际公认是4摄氏度水的10cm*10cm*10cm的容器体积为1L。这是标准，地球人都认可这种标准

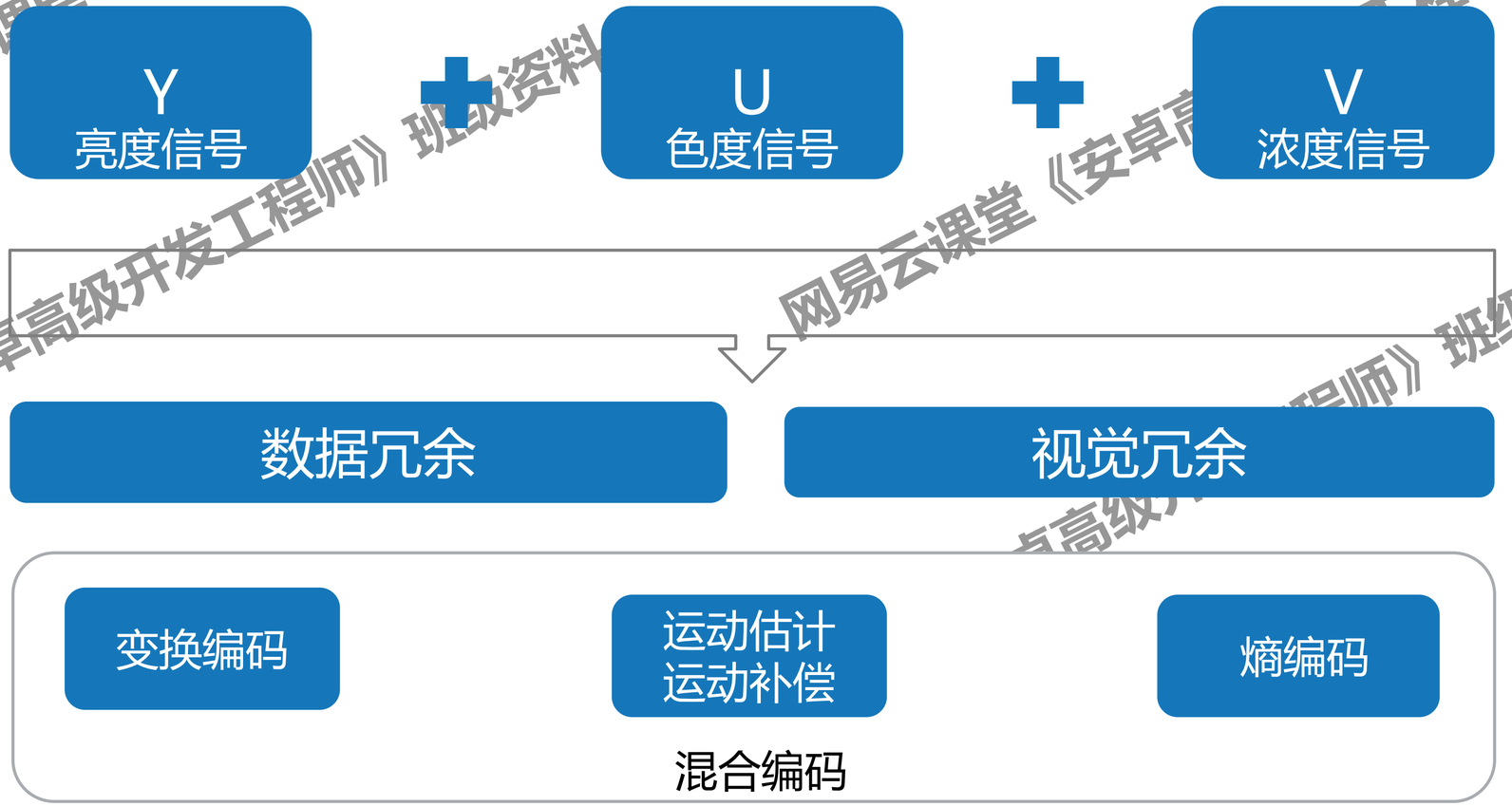
可是在音视频中 这种 参考标准会发生变化，用户可以自定义标准。标准可以用time_base表示




实际上time_base的意思就是时间的刻度：如（1,25），那么时间刻度就是1/25 每一份是0.04s

# 视频编码原理

YUV信号采集之后通过数据冗余和视觉冗余这两个基本条件，再通过混合编码算法、交换编码、运动估计和运动补偿、熵编码三种结合来进行压缩编码。



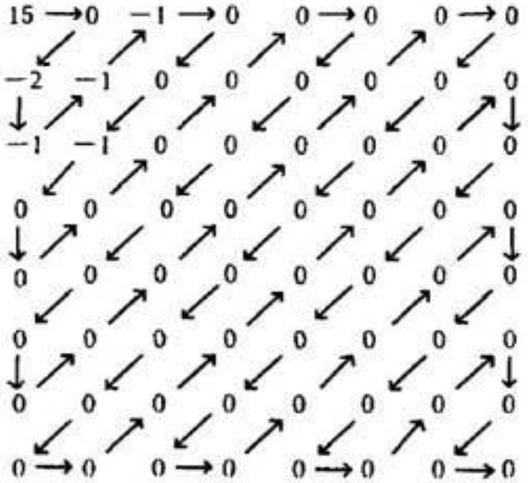
# 变换编码

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 139 | 144 | 149 | 153 | 155 | 155 | 155 | 155 |
| 144 | 151 | 153 | 156 | 159 | 156 | 156 | 156 |
| 150 | 155 | 160 | 163 | 158 | 156 | 156 | 156 |
| 159 | 161 | 162 | 160 | 160 | 159 | 159 | 159 |
| 159 | 160 | 161 | 162 | 162 | 155 | 155 | 155 |
| 161 | 161 | 161 | 161 | 160 | 157 | 157 | 157 |
| 162 | 162 | 161 | 163 | 162 | 157 | 157 | 157 |
| 162 | 162 | 161 | 161 | 163 | 158 | 158 | 158 |

8x8图像块

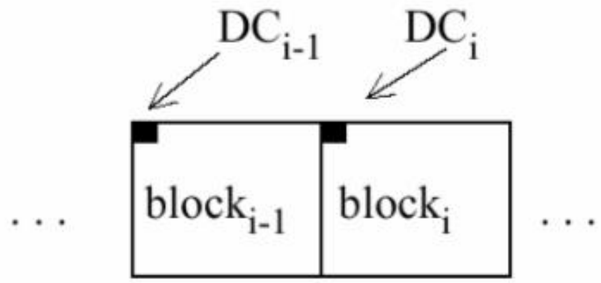
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 236 | -1.0 | -12 | -5.2 | 2.1 | -1.7 | -2.7 | -1.3 |
| -22 | -18 | -6.2 | -3.2 | -2.9 | -0.1 | 0.4 | -1.2 |
| -11 | 9.3 | -1.6 | 1.5 | 0.2 | -0.9 | -0.6 | -0.1 |
| -7.1 | -1.9 | 0.2 | 1.5 | 1.6 | -0.1 | 0 | 0.3 |
| -0.6 | -0.8 | 1.5 | 1.6 | -0.1 | -0.7 | 0.6 | 1.3 |
| -1.8 | -0.2 | 1.6 | -0.1 | -0.8 | 1.5 | 1 | -1 |
| -1.3 | -0.4 | -0.3 | -1.5 | 0.5 | 1.7 | 1.1 | -0.8 |
| -2.6 | 1.6 | -3.8 | -1.8 | 1.9 | 1.2 | -0.6 | -0.4 |

图像块经过DCT变换后的系数



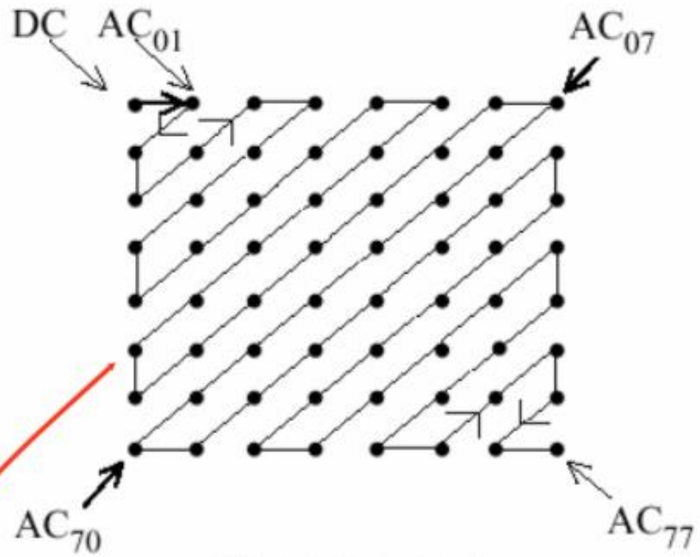

量化后的DCT系数

# 熵编码




Differential DC encoding

Huffman coded




Zigzag sequence

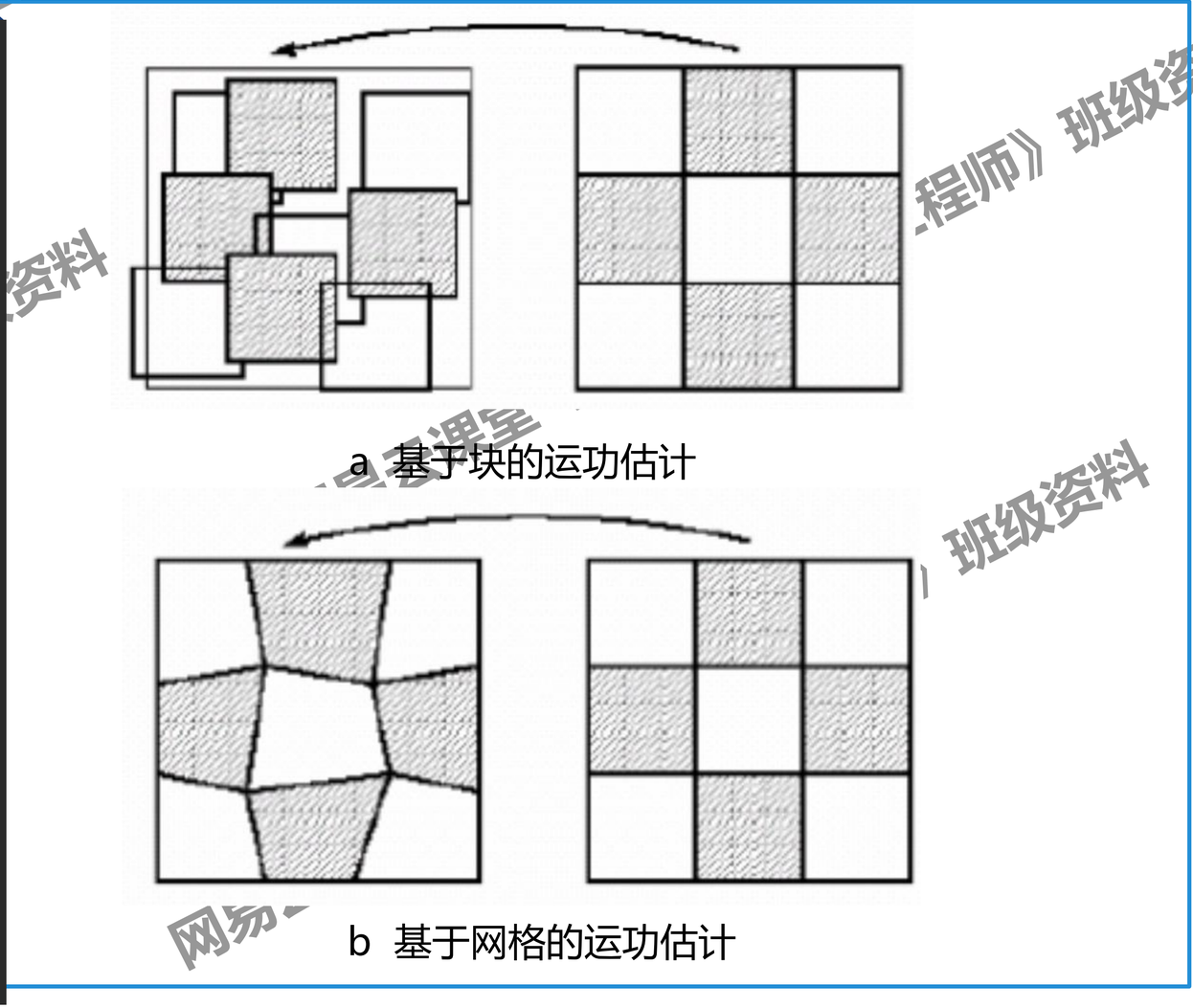
Runlength coding
then huffman or
arithmetic coding

出处：视频压缩编码和音频压缩编码的基本原理

# 运动估计和运动补偿

## 运动估计示例

基于块的运动估计
基于网格的运动估计



a 基于块的运动估计

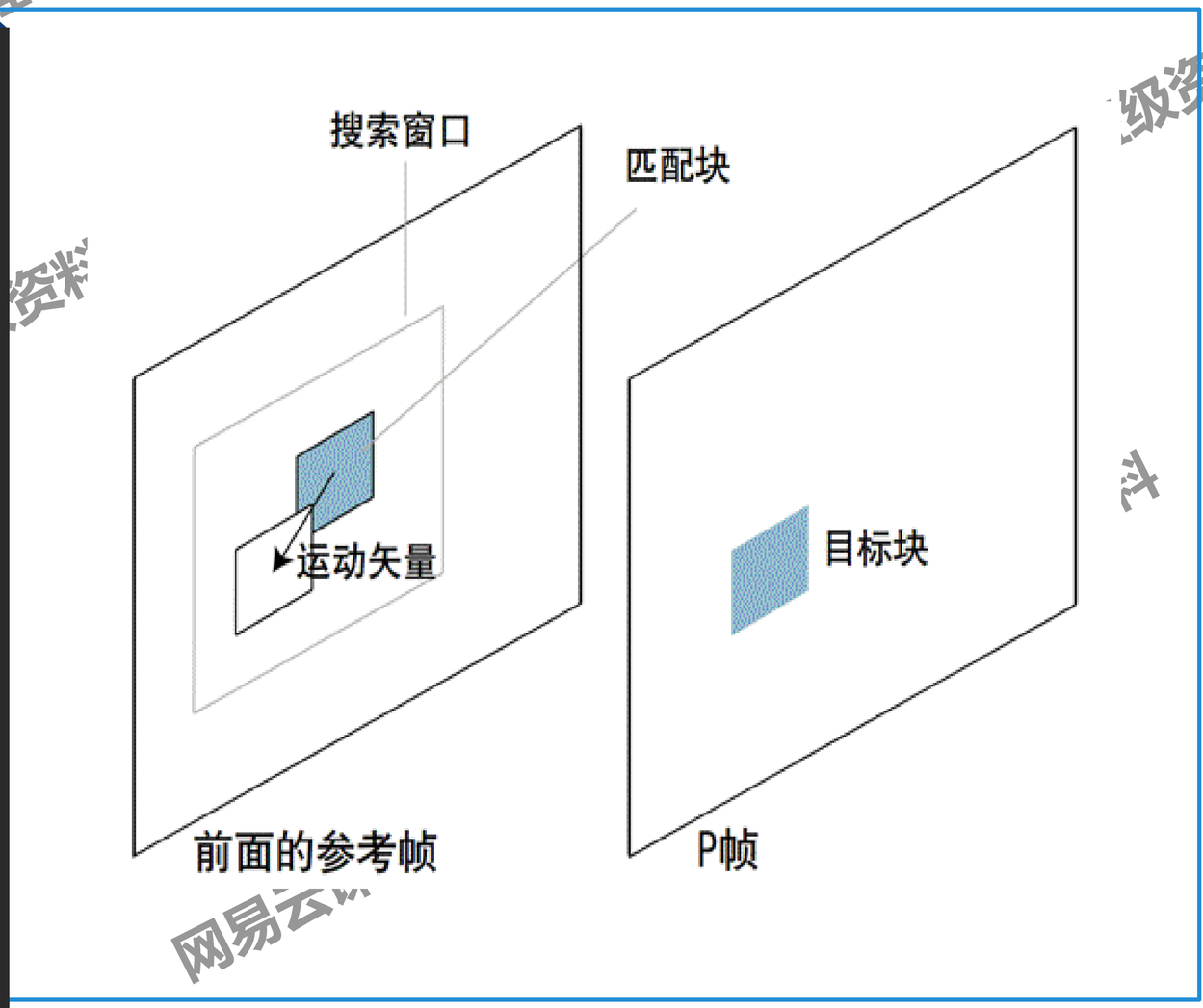
b 基于网格的运动估计

# 运动估计和运动补偿

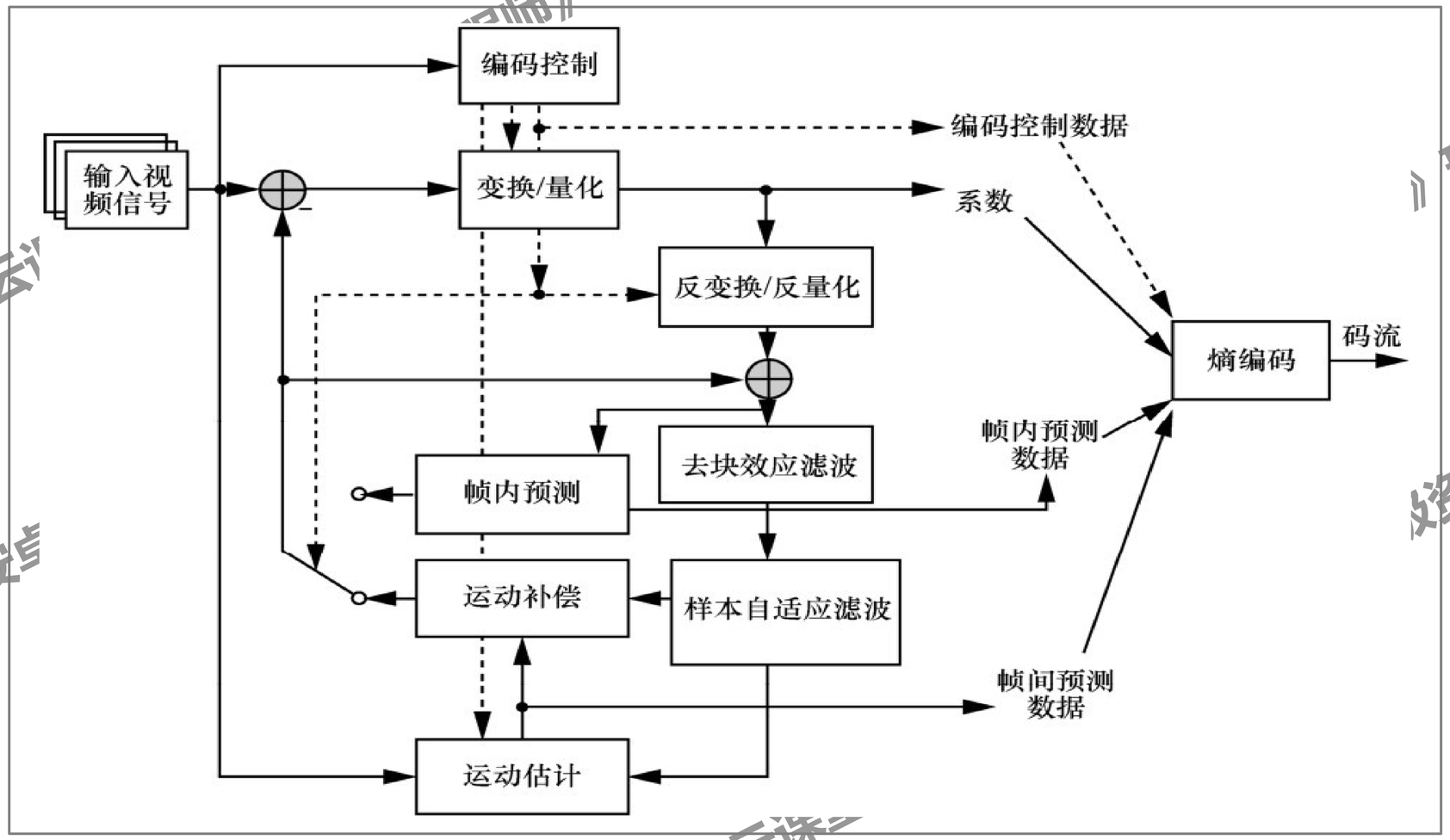
## 运动补偿示例

参考帧

P帧



# 混合编码




混合编码模型，该模型普遍应用于MPEG1，MPEG2，H.264等标准中。

# 音频编码原理

# 什么是掩蔽效应

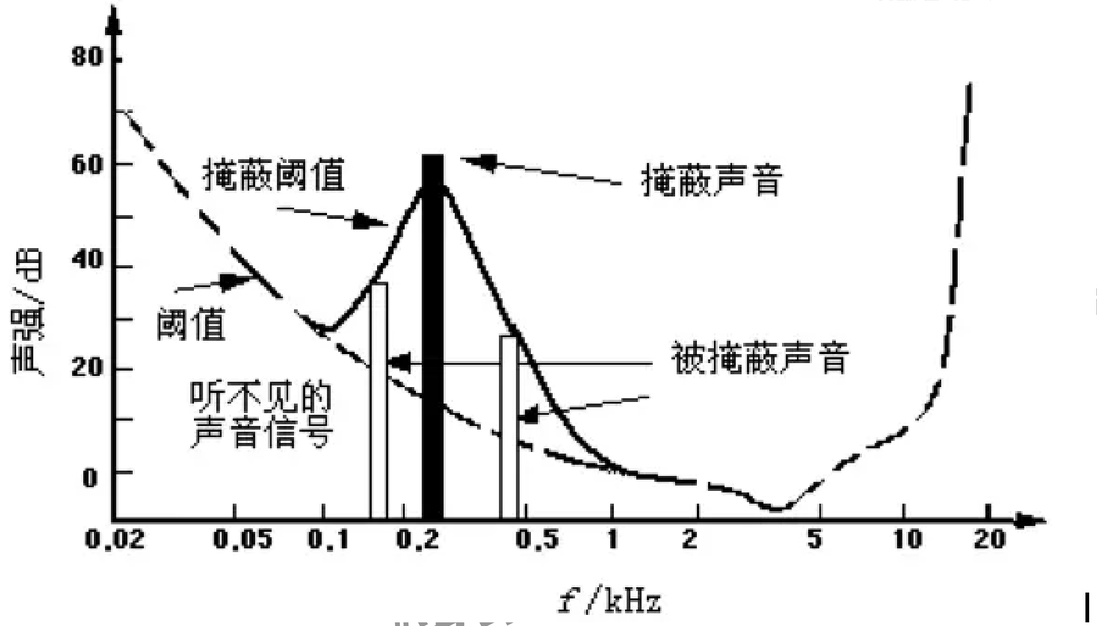
人耳对一个声音的听觉灵敏度因为另一个声音的存在而降低的现象，我们称为掩蔽效应：

1.频率相近的两个纯音间易发生掩蔽效应，且较强声音易掩蔽较弱的声音

2.低频纯音可以有效掩蔽高频纯音，反之作用小。

# 频谱掩蔽效应

2:发生在掩蔽声与被掩蔽声同时作用时
1:强音会掩蔽其频率周围的弱音



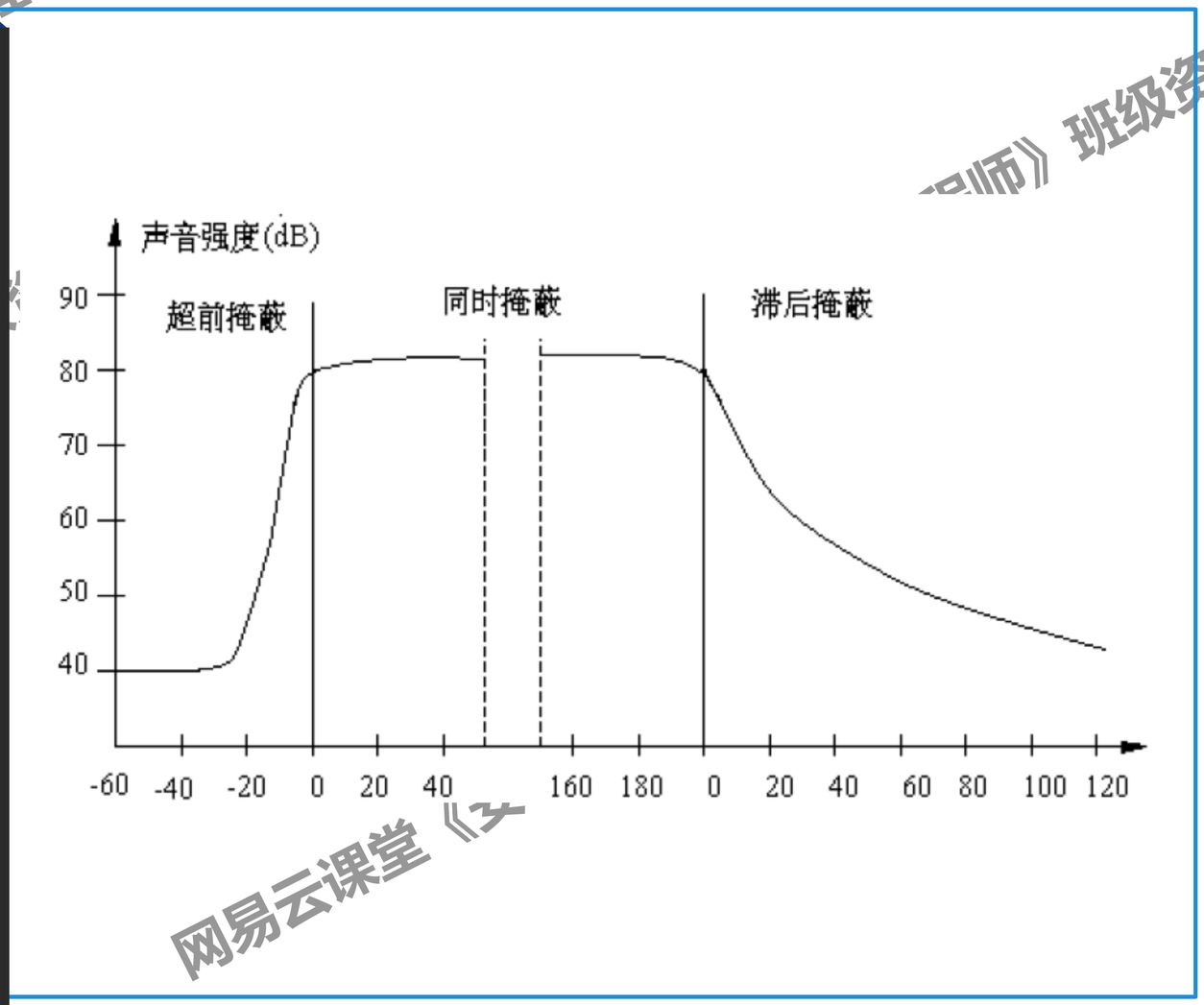
# 时域掩蔽效应

发生在掩蔽声与被掩蔽声不同时出现的时候

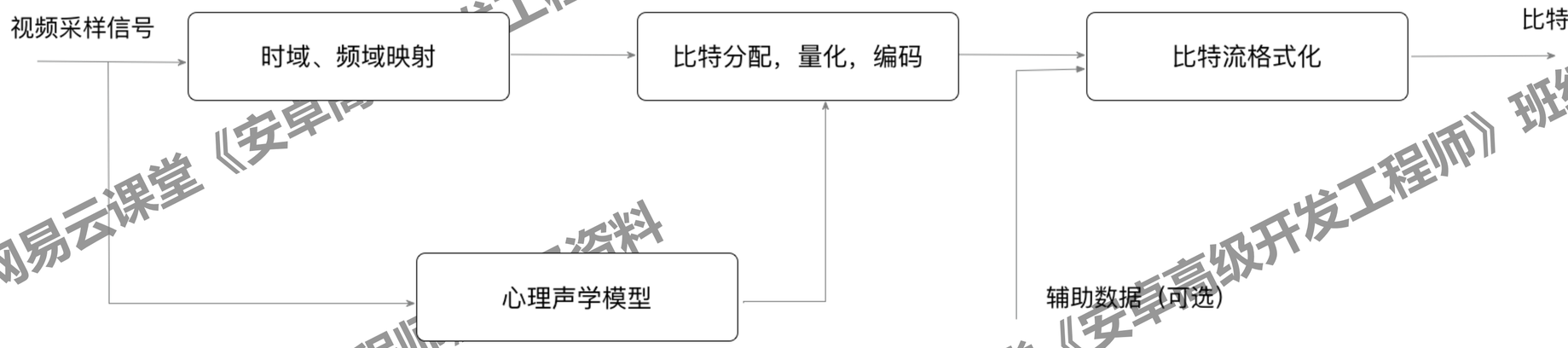
1:前掩蔽：是指人耳在听到强信号之前的短暂时间内，已经存在的弱信号会被掩蔽而听不到。

2:同时掩蔽：是指当强信号与弱信号同时存在时，弱信号会被强信号所掩蔽而听不到。

3:后掩蔽：是指当强信号消失后，需经过较长的一段时间才能重新听见弱信号。



# 音频压缩编码




数字音频编码系统模型

对每一个音频声道中的音频采样信号，首先都要将它们映射到频域中，这种时域到频域的映射可通过子带滤波器实现。每个声道中的音频采样块首先要根据心理声学模型来计算掩蔽门限值，然后由计算出的掩蔽门限值决定从公共比特池中分配给该声道的不同频率域中多少比特数，接着进行量化以及编码工作，最后将控制参数及辅助数据加入数据之中，产生编码后的数据流。

# 谢谢观看