



北京
2019

遨游“视”界 做你所想
Explore World, Do What You Want

OTT设备的多媒体系统架构设计与体验优化

峰米科技 张晖

出品: LiveVideoStack
—— 音视频技术社区 —— CSDN

关于峰米



北京
2019

遨游“视”界 做你所想
Explore World, Do What You Want

成立于2016年，是光峰光电和小米科技联合成立的小米生态链公司
主要产品为激光电视和微投





北京
2019

遨游“视”界 做你所想
Explore World, Do What You Want

1.OTT设备的特点

2.OTT平台的播放架构设计

3.OTT平台的播放体验优化

OTT设备的特点-OTT设备商的角色



北京
2019

遨游“视”界 做你所想
Explore World, Do What You Want



OTT设备商

Android Framework

@mlogic

MStar
semiconductor

OTT设备的特点-OTT设备工程师的一天



北京
2019

遨游“视”界 做你所想
Explore World, Do What You Want

中国移动 10:45 中国移动 12: 中国移动 14:00 中国移动 10:03 中国移动 11 中国移动 12:11 42%

< 微信(6) 用户体验 < 微信(3) 峰米多: < 微信(3) 峰米-yy芯片 < 微信(8) 峰米-yy芯片对 < 微信(8) 峰米多 < 微信(12) 用户体验中心



用户反馈, xxTV
动第一集时发生-

xxTV现在在我们
播放器

哥, 你们播放器播放
第一集有卡顿, 帮忙



这边RD给patch了, 我
release新so

yy芯片给release
OTA是什么时候

xxTV播放卡顿的问题解决了,
告诉用户下个OTA版本升级一
下



系统播放器



我这边复现一下

15:05

yy芯片的播放器你
们可见吗



哦, 复现到了。我



闭源了, 现在

17:15

那给



这个问题比较复杂
RD来解决

好, 辛苦尽快



两周之后



收到



OTT平台的多媒体架构设计-关键点



北京
2019

遨游“视”界 做你所想
Explore World, Do What You Want

需要能替代
Android系统
播放器

完全由我们
自己来维护

尽量减少芯片
厂商的依赖

升级不依赖rom

OTT平台的多媒体架构设计-需要的多媒体功能



北京
2019

遨游“视”界 做你所想
Explore World, Do What You Want

在线播放

- 支持常见流媒体协议：HLS，DASH，RTMP

本地播放

- 各种封装格式与编码格式，甚至蓝光格式
- 各种字幕格式，内置的、外挂的、多轨道的。字幕样式和时间偏移都可以修改
- 各种外接设备（功放、蓝牙音箱），支持多声道输出、RAW格式输出、根据输出设备不同、音频格式不同调节音画同步

其他

- 无线投屏
- 音效：Dolby，DTS
- 图像：HDMI、PQ

OTT平台的多媒体架构设计-开源播放框架对比



北京
2019

遨游“视”界 做你所想
Explore World, Do What You Want

ExoPlayer

完全基于java实现
开发集成迅速

流媒体协议支持完善

本地播放支持很差

纯java实现
存在性能问题

ijkplayer

学习成本低
上手快

流媒体协议支持完善

本地播放支持偏少

架构略混乱
社区不活跃

VLC

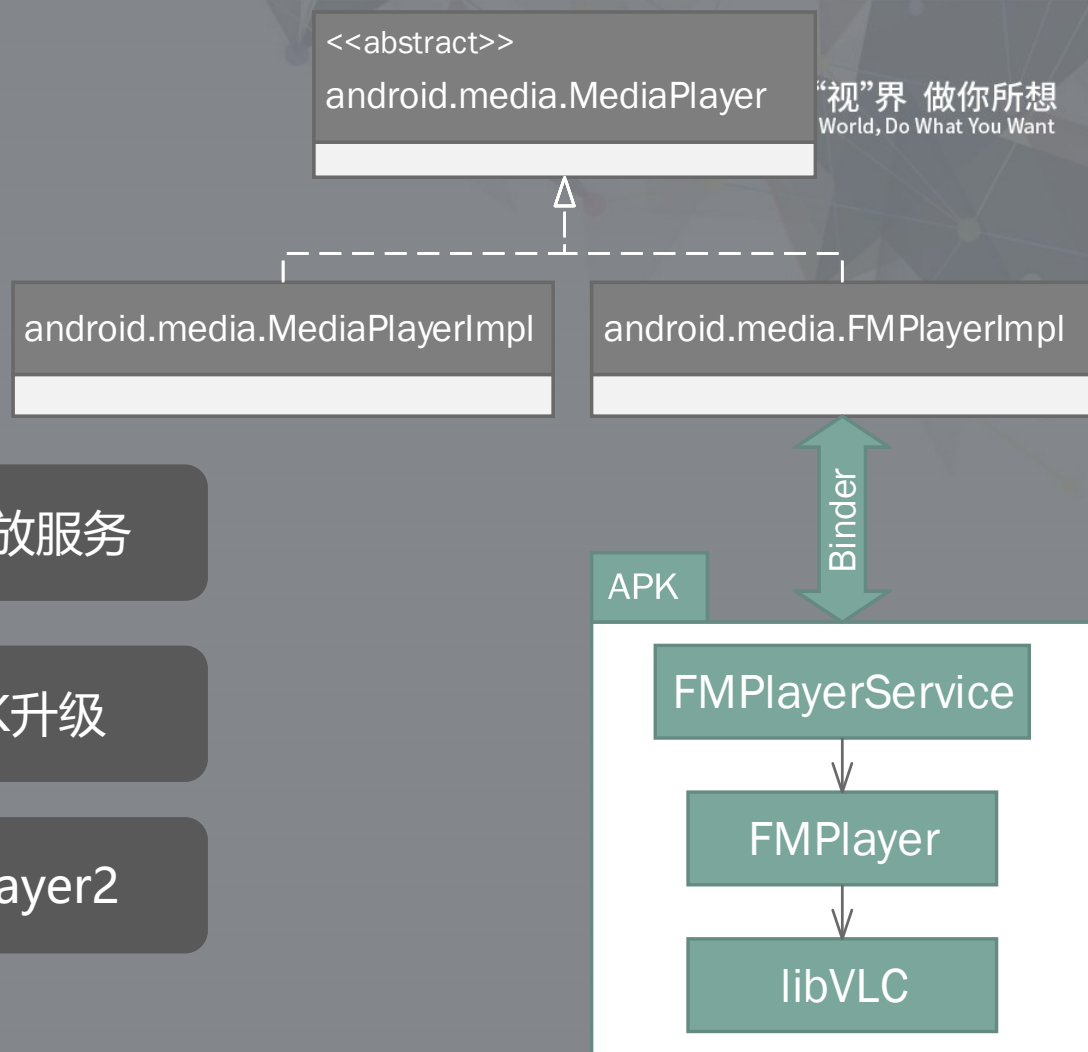
跨平台
社区活跃

本地播放支持
非常完善

学习成本高，上手慢

有自己的一套demux可
靠性待检验

OTT平台的多媒体架构设计



封装播放功能接口

系统级播放服务

切换原生播放实现

独立APK升级

内容商SDK插件化

MediaPlayer2



北京
2019

遨游“视”界 做你所想
Explore World, Do What You Want

OTT平台的播放体验优化

1. 多媒体功能自动化测试框架
2. 播放器色觉辅助功能开发
3. 从零开始的AV同步之旅



Netflix Test Studio

The screenshot displays the Netflix Test Studio web interface. The top navigation bar includes 'Test Plan', 'Testing' (active), 'My Devices', 'History', 'Results', 'Batch History', and 'Admin'. Below this, a sub-navigation bar shows 'Cloud NRD 4.2 Test Cases', 'NRDP Reference Applications: Netflix_refapp_4.2.0 (4.2.0)', 'Warning Logging', 'Production Content', and 'Test Overrides'. The main content area is titled 'BasicSmokeTest' and shows its version (1) and date (2015-06-04 18:29:02). The test description states: 'Runs a very basic playback test with activation, playback and pause. This is an engineering test that can be used to help develop your NRD. This test can be added to your test plan upon request. Runs a very basic playback test with activation, playback and pause. Tags: DevTest,nrdp:4.0,nrdp:4.1,nrdp:4.2'. A progress indicator shows the test is 'RUNNING' with a green segment on a scale from 0 to 7. A box labeled 'Receive real time test execution updates.' points to this indicator. Below the description, a table lists the test steps: Step 1 (Select Device UI 'NTS Client Test'), Step 2 (Start the Netflix Application on Device), and Step 3 (Activate the selected device account using email activation). A box labeled 'Send RPC commands to control test execution flow' points to the step table. To the right, a box labeled 'Notifications about TestExecutor state transitions' points to the 'Status' column. At the bottom, a 'Batch Tests' section shows 'Cancel' and 'Pause' buttons. A box labeled 'BasicSmokeTest Initializing the device test' points to the 'Status' column. Another box labeled 'BasicSmokeTest Starting the test' points to the 'Status' column. The footer contains copyright information for Netflix, Inc. (© 2015 Netflix, Inc.) and a disclaimer about the use of the content.

NETFLIX Netflix Test Studio Remote Testing from Any Location

Test Plan Testing My Devices History Results Batch History Admin

Cloud NRD 4.2 Test Cases NRDP Reference Applications: Netflix_refapp_4.2.0 (4.2.0) Warning Logging Production Content Test Overrides

Test Cases BasicSmokeTest

BasicSmokeTest

Version: 1 Date: 2015-06-04 18:29:02

Runs a very basic playback test with activation, playback and pause.
This is an engineering test that can be used to help develop your NRD. This test can be added to your test plan upon request.
Runs a very basic playback test with activation, playback and pause.
Tags: DevTest,nrdp:4.0,nrdp:4.1,nrdp:4.2

Receive real time test execution updates.

RUNNING

ESN: NFDEV-MGX-000002092
Log id: 45808706

Status

Duration: 00:00:01
Time: 00:00:04

Step 1 Select Device UI 'NTS Client Test'.
Step 2 Start the Netflix Application on Device.
Step 3 Activate the selected device account using email activation.

Send RPC commands to control test execution flow

Notifications about TestExecutor state transitions

BasicSmokeTest Initializing the device test
BasicSmokeTest Starting the test

Batch Tests

Create New Batch Run

Cancel Pause

© 2015 Netflix, Inc.
All content herein is protected by U.S. copyright and other applicable intellectual property laws and may not be copied without the express permission of Netflix, Inc., which reserves all rights.
Reuse of any of this content for any purpose without the permission of Netflix, Inc. is strictly prohibited.
Build Date: 2015-05-13 10:14:07

体验优化-色觉辅助-色盲/色弱的成因



北京
2019

遨游“视”界 做你所想
Explore World, Do What You Want

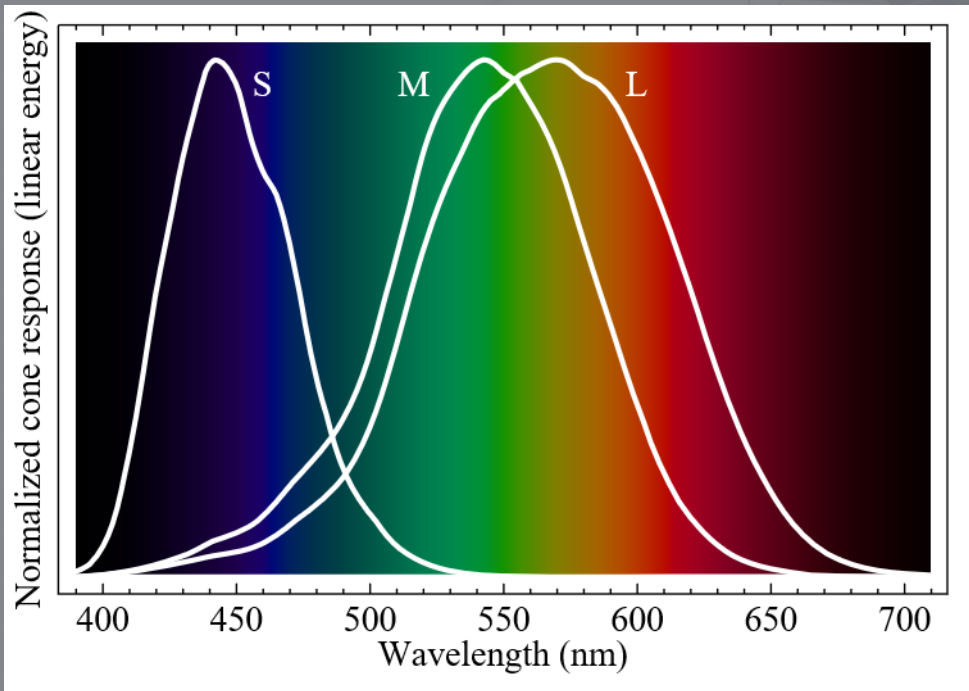
在人眼中，有3种对不同的光谱敏感的视锥细胞，造就了三色视觉。

视锥细胞按照它们的光谱敏感度峰值波长的顺序被标记为：短（S）、中（M）、和长（L）的视锥细胞类型。

三种锥细胞的缺失或功能缺陷导致色盲与色弱！

约6%人口为色弱，约2%人口色盲

视锥类型	范围	峰值波长
S	400–500 nm	420–440 nm
M	450–630 nm	534–555 nm
L	500–700 nm	564–580 nm



体验优化-色觉辅助-色盲/色弱看到的画面



北京
2019

遨游“视”界 做你所想
Explore World, Do What You Want



正常视觉



红绿色盲/色弱

体验优化-色觉辅助-色盲/色弱看到的画面



北京
2019

遨游“视”界 做你所想
Explore World, Do What You Want



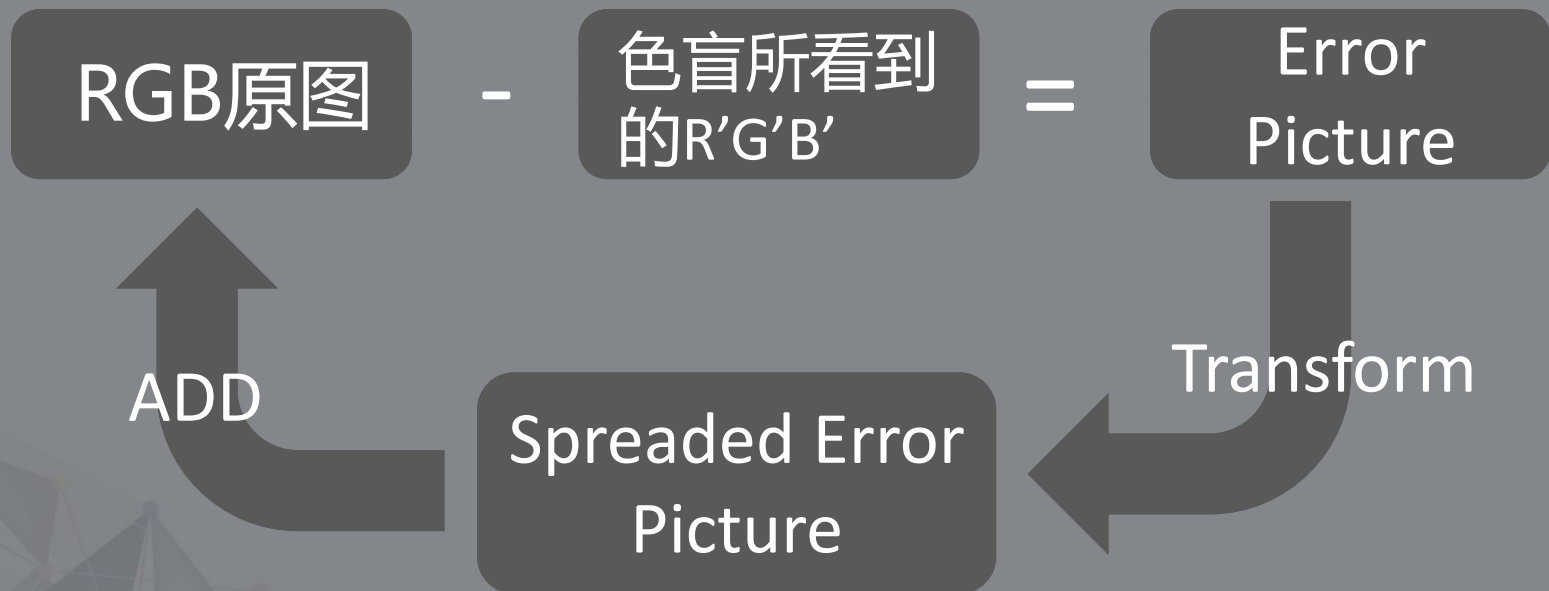
正常视觉



红绿色盲/色弱

首先明确一点：色盲/色弱是生理原因导致的，所以我们不可能让一个红色盲患者重新看到红色。

我们能做的色觉辅助是：恢复出色觉障碍患者看不到的图像信息。
如分辨球赛中的两队队员，股票k线图的涨跌等。

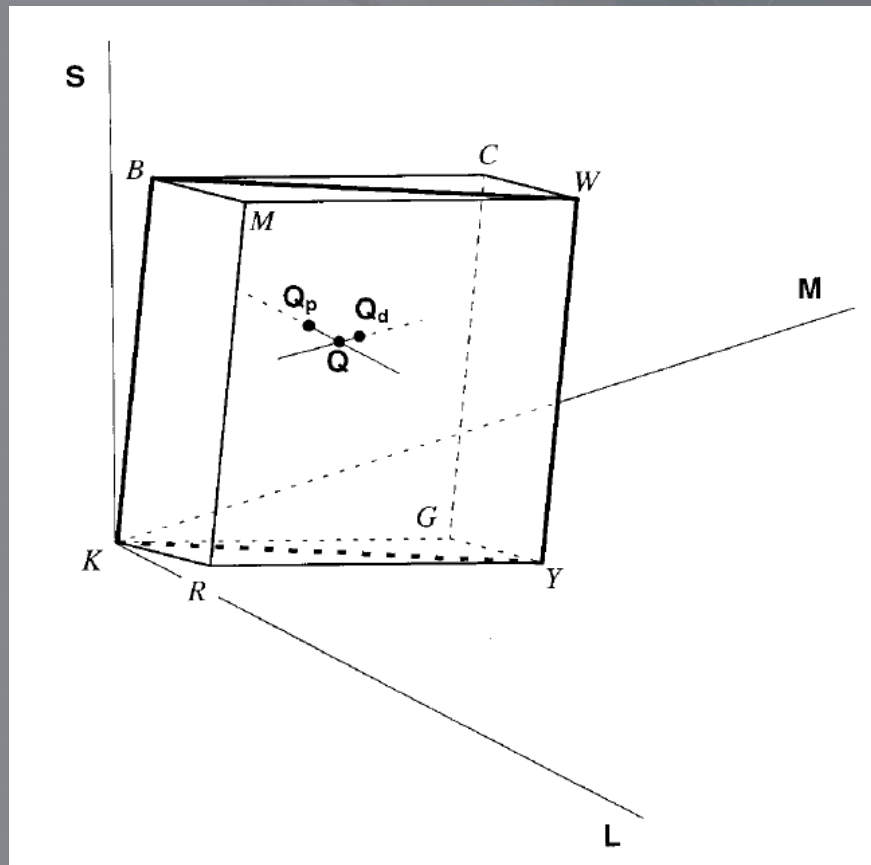


把色盲看不到的信息映射到他们能看到的色域中去

正常视觉能看到的颜色在一个三维空间中
色觉障碍人群看到的颜色则在二维平面上

对于某一颜色Q
红色盲看到的颜色相当于Q点沿L方向投射到对应的平面上，即图中Qp点
绿色盲看到的颜色相当于Q点沿M方向投射到对应的平面上，即图中Qd点。

现在我们来求Q和Q'之间的关系
即可得到色盲模拟的转换矩阵



最后得到的RGB到R'G'B'之间的转换矩阵如下

$$\text{对于红色盲, } T' = \begin{pmatrix} 0.0685 & 0.9315 & 0.0000 \\ 0.0685 & 0.9315 & 0.0000 \\ 0.0136 & -0.0136 & 1.0000 \end{pmatrix}$$

$$\text{对于绿色盲, } T' = \begin{pmatrix} 0.4156 & 0.5844 & 0.0000 \\ 0.4156 & 0.5844 & 0.0000 \\ -0.0424 & 0.0424 & 1.0000 \end{pmatrix}$$

$$\text{对于蓝色盲, } T' = \begin{pmatrix} 1.0000 & -0.0233 & 0.0233 \\ 0.0000 & 1.0003 & -0.0003 \\ 0.0000 & 1.0003 & -0.0003 \end{pmatrix}$$

对于红色盲, 相当于把RGB空间的色沿L轴投射到R=G的色面
对于绿色盲, 相当于把RGB空间的色沿M轴投射到R=G的色面
对于蓝色盲, 相当于把RGB空间的色沿S轴投射到B=G的色面

核心问题：如何将色盲人群看不到的颜色信息映射到他们能看到的色域中去

法一：

直接在RGB空间中操作，把色盲感知不到的颜色全都转换为其他的颜色

以红色盲为例，得到error picture后，乘上如下的Transform矩阵

$\begin{pmatrix} 0 & 0 & 0 \\ 0.7 & 1 & 0 \\ 0.7 & 0 & 1 \end{pmatrix}$ ，相当于把红色分别分散到绿色和蓝色通道中

法二：

在LAB色彩空间中操作，LAB中的L代表亮度通道，A代表红绿通道，B代表蓝黄通道

以红色盲为例，得到error picture后，先转换为LAB色彩空间，再乘上如下的Transform矩阵

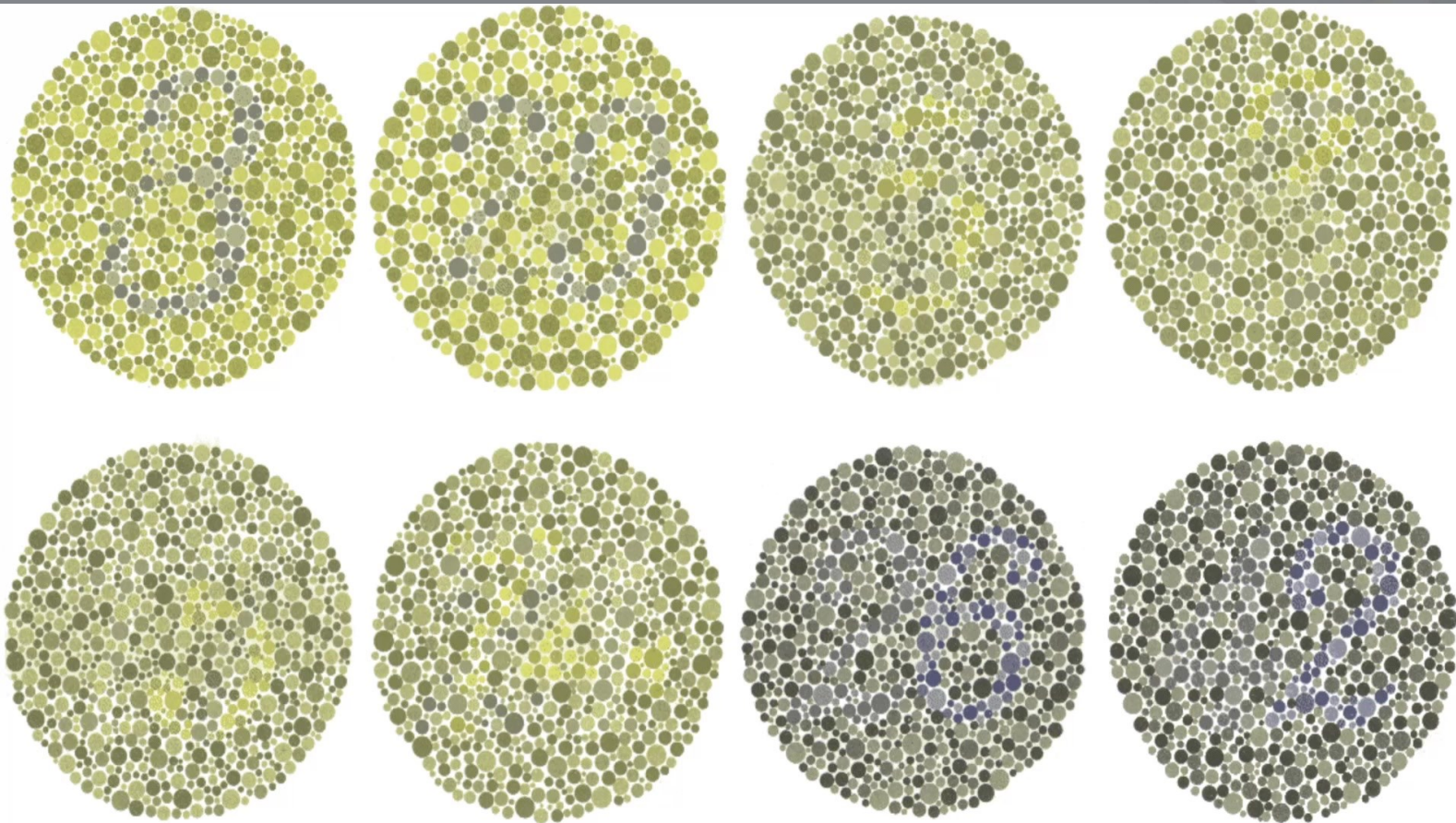
$\begin{pmatrix} 1 & 0.5 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix}$ ，相当于把红绿通道色分散到亮度通道和蓝黄通道中

体验优化-色觉辅助-优化效果



北京
2019

遨游“视”界 做你所想
Explore World, Do What You Want



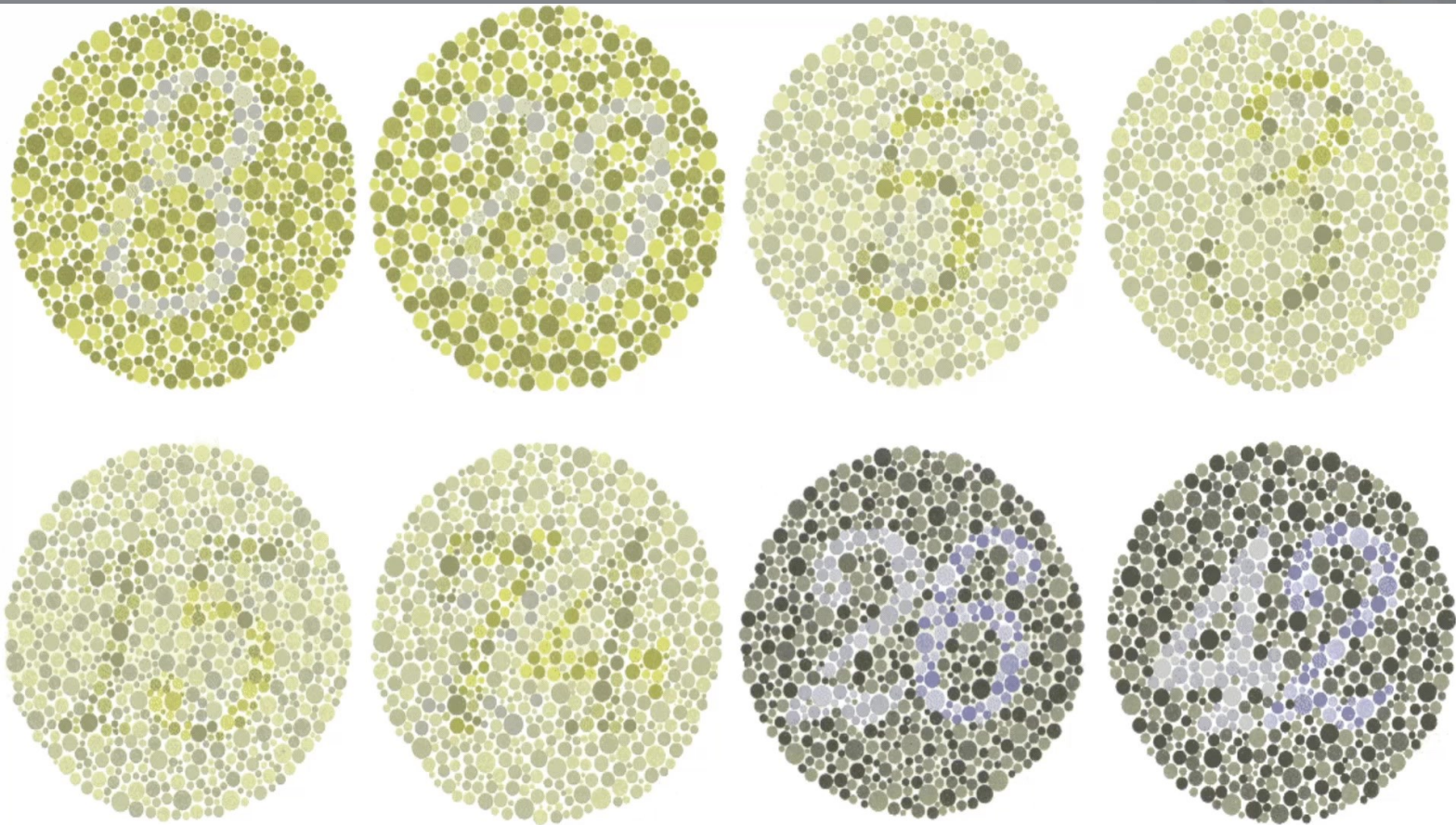
红色盲

体验优化-色觉辅助-优化效果



北京
2019

遨游“视”界 做你所想
Explore World, Do What You Want



红色盲
优化后

体验优化-色觉辅助-优化效果



北京
2019

遨游“视”界 做你所想
Explore World, Do What You Want



红绿色盲



红绿色盲优化后



1.在Android framework中实现
frameworks/native/services/surfaceflinger/Effects/Daltonizer.cpp

2.在播放器应用层加入后处理功能
如MediaCodec + GLSurfaceView的方式



北京
2019

遨游“视”界 做你所想
Explore World, Do What You Want

体验优化-音画同步

1. 测试方法与同步标准
2. Android的音画同步策略
3. 从零开始的AV同步之旅

体验优化-音画同步-测试方法



北京
2019

遨游“视”界 做你所想
Explore World, Do What You Want





- 1.通过主观评价实验来统计出一个合理的区间范围
- 2.参考杜比\ATSC等权威机构给出的区间范围
- 3.不同的输出设备需要给不同的区间范围

speaker输出时的音视频同步区间是 $[-60, +30]$ ms

蓝牙音箱输出时的音视频同步区间是 $[-160, +60]$ ms

功放设备输出时的音视频同步区间是 $[-140, +40]$ ms

负值代表音频落后于视频，正值代表音频领先于视频。



视频方面，我们关注同步逻辑对视频码流的pts做了哪些调整。

- 1、利用pts和系统时间计算预计送显时间（视频帧应该在这个时间点显示）
- 2、利用vsync对预计送显时间进行调整
- 3、计算实际送显时间与当前系统时间之间的时间差，决定是要丢帧还是送显

音频方面，我们关注同步逻辑中是如何获取“Audio当前播放的时间”的。

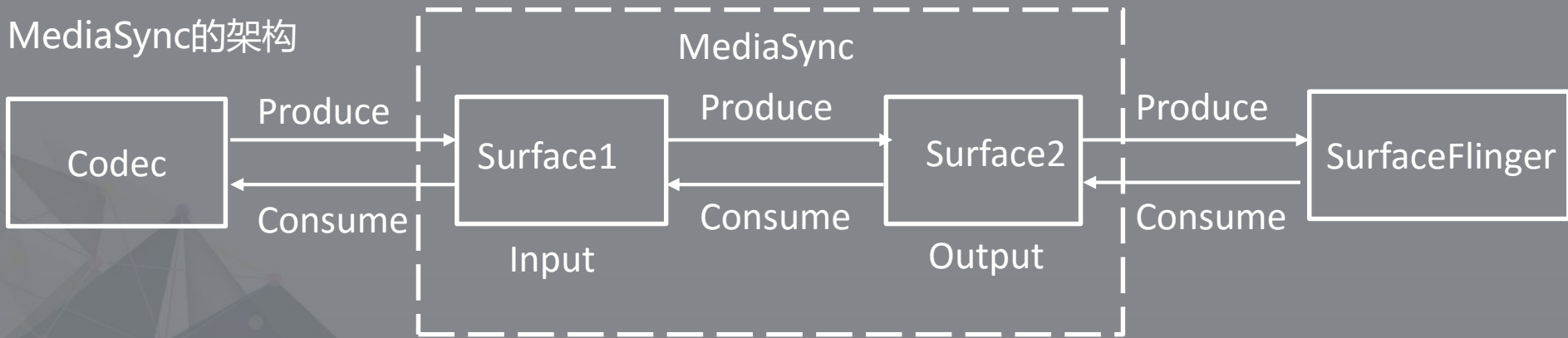
- 1、在设备支持的前提下，优先使用AudioTrack的getTimeStamp方法获取
- 2、其次选择AudioTrack的getPlayheadPositionUs，此时需要做平滑以及考虑后端延迟

MediaSync是Android M新加入的API，可以帮助应用实现视音频的同步播放

普通的MediaCodec架构



MediaSync的架构

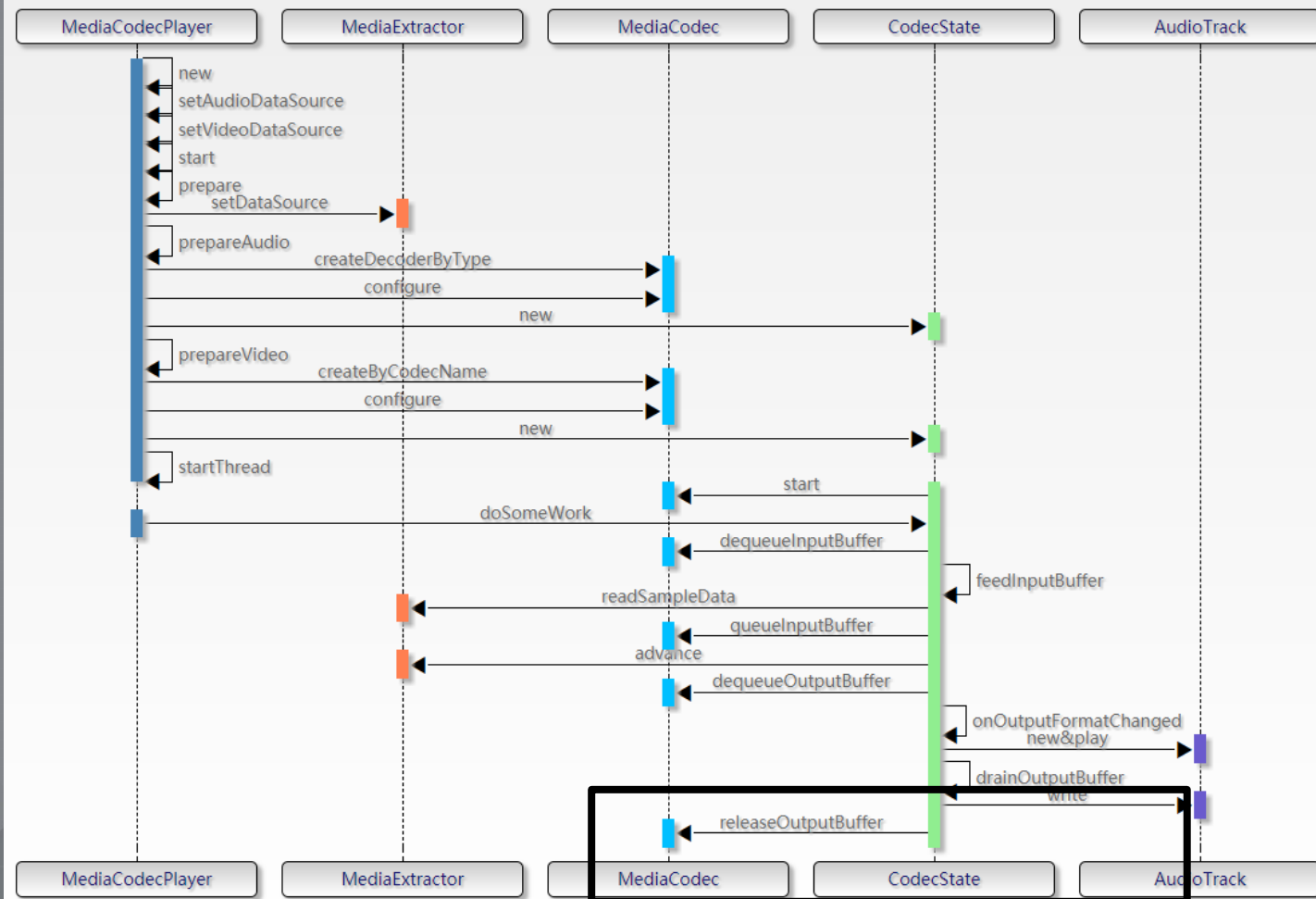




核心问题：

1. 究竟哪种方法可以达到最好的avsync结果？
2. 哪些逻辑是必要的？
3. 如果我们想用Android原生API从零开始写一个av同步的播放器，都需要做哪些工作？

播放器	1st	2nd	3rd	Avg (ms)
ExoPlayer	-26	-18	-31	-25
MediaSync	+23	+32	+33	+29
ijkplayer	-77	-63	-75	-71



Doing av sync in drainOutputBuffer

”界 做你所想
d, Do What You Want



用video的pts与audio的播放时间进行对比，然后直接调用releaseOutputBuffer方法进行显示

Audio部分

利用AudioTrack的getPlaybackHeadPosition方法返回audio的播放时间

```
int numFramesPlayed = mAudioTrack.getPlaybackHeadPosition();  
return (numFramesPlayed * 1000000L) / mSampleRate;
```

Video部分

调用的是没有timestamp参数的releaseOutputBuffer方法

```
mCodec.releaseOutputBuffer(index, render);
```

测试结果：-173ms 惨不忍睹

改造audio播放时间的获取，加上latency

```
public long getAudioTimeUs() {  
    int numFramesPlayed = mAudioTrack.getPlaybackHeadPosition();  
    if (getLatencyMethod != null) {  
        try {  
            latencyUs = (Integer) getLatencyMethod.invoke(mAudioTrack, (Object[]) null) * 1000L / 2;  
            latencyUs = Math.max(latencyUs, 0);  
        } catch (Exception e){  
            getLatencyMethod = null;  
        }  
    }  
    return (numFramesPlayed * 1000000L) / mSampleRate - latencyUs;  
}
```

测试结果：-128ms



用上getTimeStamp方法

```
public long getAudioTimeUs() {  
    long systemClockUs = System.nanoTime() / 1000;  
    int numFramesPlayed = mAudioTrack.getPlaybackHeadPosition();  
    if (systemClockUs - lastTimestampSampleTimeUs >= MIN_SAMPLE_INTERVAL_US) {  
        audioTimestampSet = mAudioTrack.getTimestamp(audioTimestamp);  
        if (getLatencyMethod != null) {  
            ...  
        }  
        lastTimestampSampleTimeUs = systemClockUs;  
    }  
}
```

测试结果：-87.5ms



基于vsync调整送显时间，使用带时间戳参数的releaseOutputBuffer方法

```
public long getRealTimeUsForMediaTime(long mediaTimeUs) {  
    ...  
    long unadjustedFrameReleaseTimeNs = System.nanoTime() + (earlyUs * 1000);  
    long adjustedReleaseTimeNs = frameReleaseTimeHelper.adjustReleaseTime(  
        mediaTimeUs, unadjustedFrameReleaseTimeNs);  
    return adjustedReleaseTimeNs / 1000;  
}  
  
mCodec.releaseOutputBuffer(index, realTimeUs*1000);
```

测试结果：-76ms



提前两倍vsync时间调用releaseOutputBuffer

```
long lateUs = System.nanoTime()/1000 - realTimeUs;
//如果video比预期release时间来的早了2*vsyncduration时间以上，
//则跳过并且进入下次循环，否则予以显示
long twiceVsyncDurationUs = 2 * mMediaTimeProvider.getVsyncDurationNs()/1000;
if (lateUs < -twiceVsyncDurationUs) {
    // too early;
    return false;
} else {
    mCodec.releaseOutputBuffer(index, realTimeUs*1000);
}
```

测试结果：-66ms



播放器	测试结果
v1.0 最简单朴素的同步逻辑	-173ms
v2.0 考虑latency	-128ms
v3.0 使用getTimestamp	-87.5ms
v4.0 基于vsync调整releasetime,并且使用带时间戳参数的releaseOutputBuffer接口	-76ms
v5.0 提前两倍vsync duration时间调用releaseOutputBuffer接口	-66ms
v6.0 其他小优化	-56ms
ExoPlayer	-25ms
ijkplayer	-71ms
MediaSync	+29ms



北京
2019

遨游“视”界 做你所想
Explore World, Do What You Want

总结与展望

1. 真正的“最后一公里”
2. 玩家越多，机会越多
3. 融合新技术



北京
2019

遨游“视”界 做你所想
Explore World, Do What You Want

Thank you



出品: LiveVideoStack CSDN
—— 音视频技术社区 ——