短视频 SDK

高级功能和特效

产品文档

腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2022 腾讯云版权所有

本文档（含所有文字、数据、图片等内容）完整的著作权归腾讯云计算（北京）有限责任公司单独所有，未经腾讯云事先明确书面许可，任何主体不得以任何形式复制、修改、使用、抄袭、传播本文档全部或部分内容。前述行为构成对腾讯云著作权的侵犯，腾讯云将依法采取措施追究法律责任。

【商标声明】

腾讯云

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。未经腾讯云及有关权利人书面许可，任何主体不得以任何方式对前述商标进行使用、复制、修改、传播、抄录等行为，否则将构成对腾讯云及有关权利人商标权的侵犯，腾讯云将依法采取措施追究法律责任。

【服务声明】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100。

文档目录

版权所有：腾讯云计算（北京）有限责任公司

高级功能和特效
类抖音特效
类抖音特效(iOS)

最近更新时间：2021-07-23 19:44:22

滤镜特效

您可以为视频添加多种滤镜特效，我们目前支持11种滤镜特效，每种滤镜您也可以设置视频作用的起始时间和结束时间。如果同一个时间点设置了多种滤镜特效，SDK 会应用最后一种滤镜特效作为当前的滤镜特效。

设置特效的方法为：

```
- (void) startEffect:(TXEffectType)type startTime:(float)startTime;
- (void) stopEffect:(TXEffectType)type endTime:(float)endTime;
//特效的类型（type 参数），在常量 TXEffectType 中有定义：
typedef NS_ENUM(NSInteger,TXEffectType)
{
TXEffectType_ROCK_LIGHT, //动感光波
TXEffectType_DARK_DRAEM, //暗黑幻境
TXEffectType_SOUL_OUT, //灵魂出窍
TXEffectType_SCREEN_SPLIT,//视频分裂
TXEffectType_WIN_SHADOW, //百叶窗
TXEffectType_GHOST_SHADOW,//鬼影
TXEffectType_PHANTOM, //幻影
TXEffectType_GHOST, //幽灵
TXEffectType_LIGHTNING, //闪电
TXEffectType_MIRROR, //镜像
TXEffectType_ILLUSION, //幻觉
};
- (void) deleteLastEffect;
- (void) deleteAllEffect;
```

调用 deleteLastEffect() 删除最后一次设置的滤镜特效。
调用 deleteAllEffect() 删除所有设置的滤镜特效。

版权所有：腾讯云计算（北京）有限责任公司

Demo 示例：

在1 – 2s之间应用第一种滤镜特效；在3 – 4s之间应用第2种滤镜特效；删除3 – 4s设置的滤镜特效。

```
//在1-2s之间应用第一种滤镜特效
[_ugcEdit startEffect:TXEffectType_SOUL_OUT startTime:1.0];
[_ugcEdit stopEffect:TXEffectType_SOUL_OUT startTime:2.0)];
//在3-4s之间应用第2种滤镜特效
[_ugcEdit startEffect:TXEffectType_SPLIT_SCREEN startTime:3.0];
[_ugcEdit stopEffect:TXEffectType_SPLIT_SCREEN startTime:4.0];
//删除3-4s设置的滤镜特效
[_ugcEdit deleteLastEffect];
```

慢/快动作

您可以进行多段视频的慢速/快速播放，设置慢速/快速播放的方法为：

```
- (void) setSpeedList:(NSArray *)speedList;

//TXSpeed 的参数如下：
@interface TXSpeed: NSObject
@property (nonatomic, assign) CGFloat startTime; //加速播放起始时间(s)
@property (nonatomic, assign) CGFloat endTime; //加速播放结束时间(s)
@property (nonatomic, assign) TXSpeedLevel speedLevel; //加速级别
@end

// 目前支持变速速度的几种级别，在常量 TXSpeedLevel 中有定义：
typedef NS_ENUM(NSInteger, TXSpeedLevel) {
SPEED_LEVEL_SLOWEST, //极慢速
SPEED_LEVEL_SLOW, //慢速
SPEED_LEVEL_NOMAL, //正常速
SPEED_LEVEL_FAST, //快速
SPEED_LEVEL_FASTEST, //极快速
};
```

Demo 示例：

```
// SDK 拥有支持多段变速的功能。 此 Demo 仅展示一段慢速播放
TXSpeed *speed =[[TXSpeed alloc] init];
```

```
speed.startTime = 1.0;
speed.endTime = 3.0;
speed.speedLevel = SPEED_LEVEL_SLOW;
[_ugcEdit setSpeedList:@[speed]];
```

倒放

您可以将视频画面倒序播放，设置倒放的方法：

```
- (void) setReverse:(BOOL)isReverse;
```

Demo 示例：

```
[_ugcEdit setReverse:YES];
```

重复视频片段

您可以设置重复播放一段视频画面，声音不会重复播放。
设置重复片段方法：

```
- (void) setRepeatPlay:(NSArray *)repeatList;

//TXRepeat 的参数如下：
@interface TXRepeat: NSObject
@property (nonatomic, assign) CGFloat startTime; //重复播放起始时间(s)
@property (nonatomic, assign) CGFloat endTime; //重复播放结束时间(s)
@property (nonatomic, assign) int repeatTimes; //重复播放次数
@end
```

Demo 示例：

```
TXRepeat *repeat = [[TXRepeat alloc] init];
repeat.startTime = 1.0;
repeat.endTime = 3.0;
repeat.repeatTimes = 3; //重复次数
[_ugcEdit setRepeatPlay:@[repeat]];
```

版权所有：腾讯云计算（北京）有限责任公司

类抖音特效(Android)

最近更新时间：2021-01-27 15:33:16

滤镜特效

您可以为视频添加多种滤镜特效，我们目前支持11种滤镜特效，每种滤镜您也可以设置视频作用的起始时间和结束时间。如果同一个时间点设置了多种滤镜特效，SDK 会应用最后一种滤镜特效作为当前的滤镜特效。

设置滤镜特效：

```
/**
* 设置滤镜特效开始时间
* @param type 滤镜特效类型
* @param startTime 滤镜特效开始时间（ms）
*/
public void startEffect(int type, long startTime);
/**
* 设置滤镜特效结束时间
* @param type 滤镜特效类型
* @param endTime 滤镜特效结束时间（ms）
*/
public void stopEffect(int type, long endTime);
```

参数说明：@param type：滤镜特效的类型，在常量 TXVideoEditConstants 中有定义：

```
public static final int TXEffectType_SOUL_OUT = 0; //滤镜效果1
public static final int TXEffectType_SPLIT_SCREEN = 1; //滤镜效果2
public static final int TXEffectType_DARK_DRAEM = 2; //滤镜效果3
public static final int TXEffectType_ROCK_LIGHT = 3; //滤镜效果4
public static final int TXEffectType_WIN_SHADDOW = 4; //滤镜效果5
public static final int TXEffectType_GHOST_SHADDOW = 5; //滤镜效果6
public static final int TXEffectType_PHANTOM_SHADDOW = 6; //滤镜效果7
public static final int TXEffectType_GHOST = 7; //滤镜效果8
public static final int TXEffectType_LIGHTNING = 8; //滤镜效果9
public static final int TXEffectType_MIRROR = 9; //滤镜效果10
public static final int TXEffectType_ILLUSION = 10; //滤镜效果11
```

版权所有：腾讯云计算（北京）有限责任公司

删除最后一个设置的滤镜特效：

```
public void deleteLastEffect();
```

删除所有设置的滤镜特效：

```
public void deleteAllEffect();
```

完整示例如下：
在1 – 2s之间应用第一种滤镜特效；在3 – 4s之间应用第2种滤镜特效；删除3 – 4s设置的滤镜特效。

```
//在1-2s之间应用第一种滤镜特效
mTXVideoEditer.startEffect(TXVideoEditConstants.TXEffectType_SOUL_OUT, 1000);
mTXVideoEditer.stopEffect(TXVideoEditConstants.TXEffectType_SOUL_OUT, 2000);
//在3-4s之间应用第2种滤镜特效
mTXVideoEditer.startEffect(TXVideoEditConstants.TXEffectType_SPLIT_SCREEN, 3000);
mTXVideoEditer.stopEffect(TXVideoEditConstants.TXEffectType_SPLIT_SCREEN, 4000);
//删除3-4s设置的滤镜特效
mTXVideoEditer.deleteLastEffect();
```

慢/快动作

您可以进行多段视频的慢速/快速播放，设置慢速/快速播放的方法为：

```
public void setSpeedList(List speedList);

//TXSpeed 的参数如下：
public final static class TXSpeed {
public int speedLevel; // 变速级别
public long startTime; // 开始时间
public long endTime; // 结束时间
}

// 目前支持变速速度的几种级别，在常量 TXVideoEditConstants 中有定义：
SPEED_LEVEL_SLOWEST - 极慢
SPEED_LEVEL_SLOW - 慢
SPEED_LEVEL_NORMAL - 正常
```

SPEED_LEVEL_FAST - 快
SPEED_LEVEL_FASTEST - 极快

完整示例如下：

```
List<txvideoeditconstants.txspeed> list = new ArrayList<>();
TXVideoEditConstants.TXSpeed speed1 = new TXVideoEditConstants.TXSpeed();
speed1.startTime = 0;
speed1.endTime = 1000;
speed1.speedLevel = TXVideoEditConstants.SPEED_LEVEL_SLOW; // 慢速
list.add(speed1);

TXVideoEditConstants.TXSpeed speed2 = new TXVideoEditConstants.TXSpeed();
speed2.startTime = 1000;
speed2.endTime = 2000;
speed2.speedLevel = TXVideoEditConstants.SPEED_LEVEL_SLOWEST; // 极慢速
list.add(speed2);

TXVideoEditConstants.TXSpeed speed3 = new TXVideoEditConstants.TXSpeed();
speed3.startTime = 2000;
speed3.endTime = 3000;
speed3.speedLevel = TXVideoEditConstants.SPEED_LEVEL_SLOW; //慢速
list.add(speed3);

mTXVideoEditer.setSpeedList(list);
```

倒放

您可以将视频画面倒序播放。通过调用 **setReverse(true)** 开启倒序播放，调用 **setReverse(false)** 停止倒序播放。

> ⚠ 注意：
> **setTXVideoReverseListener()** 老版本（SDK 4.5以前）首次监听需要手动调用，新版本不需要调用即可生效。

Demo 示例：

```
mTXVideoEditer.setTXVideoReverseListener(mTxVideoReverseListener);
mTXVideoEditer.setReverse(true);
```

重复视频片段

您可以设置重复播放一段视频画面，声音不会重复播放。目前 Android 只支持设置一段画面重复，重复三次。如需取消之前设置的重复片段，调用 setRepeatPlay(null) 即可。

设置重复片段方法：

```
public void setRepeatPlay(List repeatList);

//TXRepeat 的参数如下：
public final static class TXRepeat {
public long startTime; //重复播放起始时间(ms)
public long endTime; //重复播放结束时间(ms)
public int repeatTimes; //重复播放次数
}
```

Demo 示例：

```
long currentPts = mVideoProgressController.getCurrentTimeMs();

List repeatList = new ArrayList<>();
TXVideoEditConstants.TXRepeat repeat = new TXVideoEditConstants.TXRepeat();
repeat.startTime = currentPts;
repeat.endTime = currentPts + DEAULT_DURATION_MS;
repeat.repeatTimes = 3; //目前只支持重复三次
repeatList.add(repeat); //目前只支持重复一段时间
mTXVideoEditer.setRepeatPlay(repeatList);
```

版权所有：腾讯云计算（北京）有限责任公司

动效变脸
大眼瘦脸和挂件(iOS)

最近更新时间：2021-12-10 11:23:38

目前，仅原移动直播企业版 SDK，短视频企业版 SDK 以及短视频企业版 Pro SDK 支持高级美颜特效，需要购买原 移动直播企业版 License、短视频企业版 License 或短视频企业版 Pro License 后，使用对应的功能。

功能说明

大眼、瘦脸、动效贴纸、绿幕等特效功能，是基于优图实验室的 AI 识别技术和天天 P 图的美妆技术为基础开发的特权功能，腾讯云小视频团队通过跟优图和 P 图团队合作，将这些特效深度整合到 RTMP SDK 的图像处理流程中，以实现更好的视频特效。

接入流程

单击此处 申请企业版本 License。

版本下载

在 SDK 下载 页面下方下载企业版 SDK 压缩包，压缩包有加密（解压密码和 Licence 在接入流程步骤获取），成功解压后得到一个 Demo 和 SDK 文件，特效资源存放在 SDK/Resource 下。

Xcode 工程设置

具体操作请参见 工程配置。

步骤1：添加 Framework

企业版需要额外链接以下 Framework：

- AssetsLibrary.framwork
- CoreMedia.framework
- Accelerate.framework
- Metal.framework

步骤2：添加链接参数

1. 在工程 **Build Setting** > **Other Link Flags** 里，增加 -ObjC 选项。
2. 如果使用了 AI 抠背功能，需要把 **Product** > **Edit Scheme** > **Run** > **Options** > **Metal API Validation** 设置为 Disabled。

步骤3：添加动效资源

请检查是否添加动效资源：将 SDK/Resource 下的文件以 groups 的形式添加到工程中，文件列表如下：

- AECore.metallib
- FilterEngine.bundle
- RPNSegmenter.bundle
- YTHandDetector.bundle
- detector.bundle
- e1
- o1
- poseest.bundle
- u1
- ufa.bundle
- v1

步骤4：导入 Licence 文件

企业版需要 Licence 验证通过后，相应功能才能生效。您可以向我们的商务人员申请一个免费30天的调试用 Licence。
得到 Licence 后，您需要将其命名为 **YTFaceSDK.licence**，然后如上图所示添加到工程。

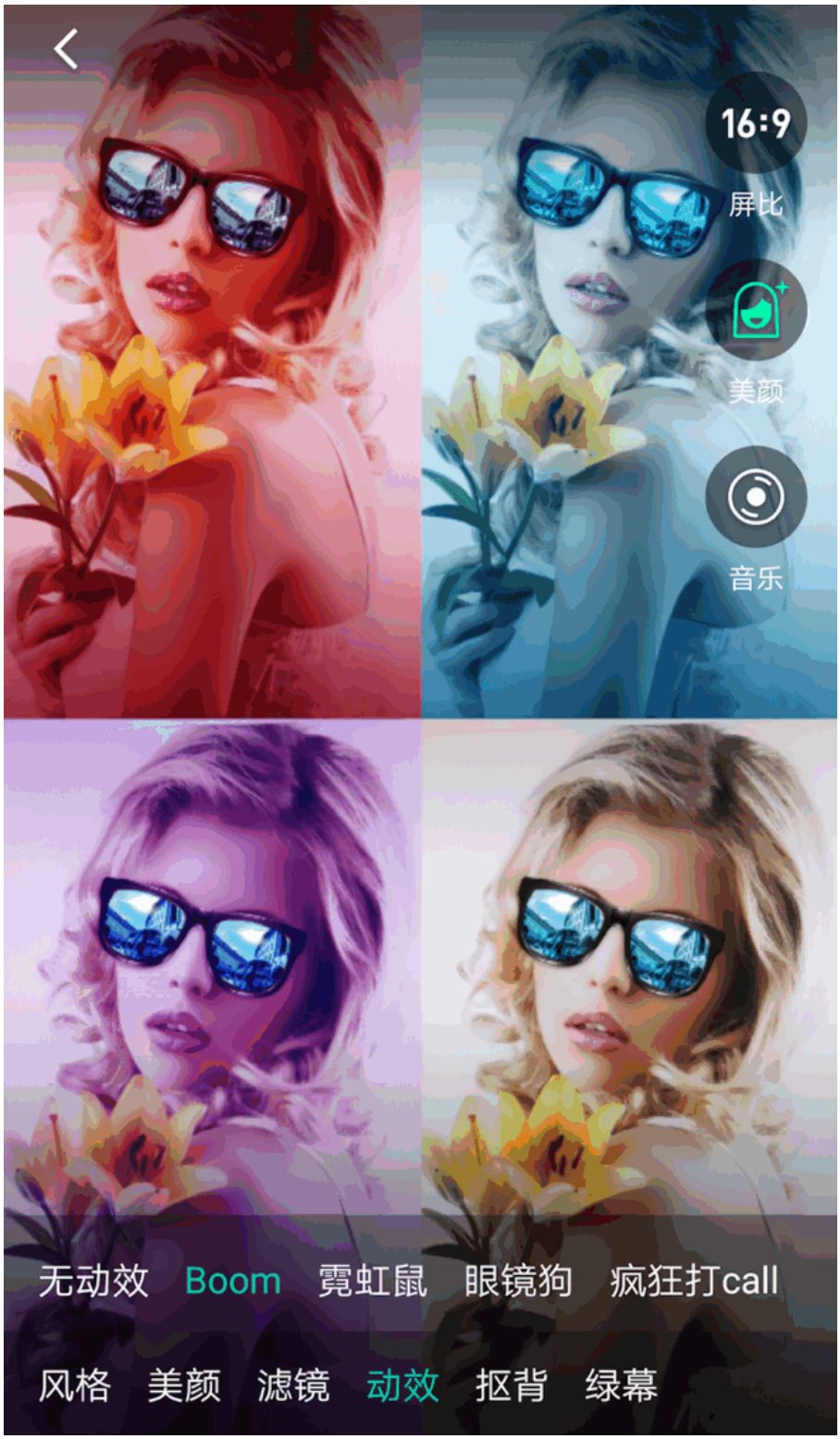
> 说明：
>
> - 每个 Licence 都有绑定具体的 Bundle Identifier，修改 app 的 Bundle Identifier 会导致验证失败。
> - YTFaceSDK.licence 的文件名固定，不可修改。
> - iOS 和 Android 不需要重复申请 licence，一个 Licence 可以同时授权一个 iOS 的 bundleid 和一个 Android 的 packageName。

从9.4版本开始，SDK 支持二合一的 Licence，这种方式不再需要 YTFaceSDK.licence，在从商务人员处获取到 Licence 对应的 key 和 url 后，设置方式和标准版 Licence 设置方式相同。

功能调用

动效贴纸

示例



一个动效模板是一个目录，里面包含很多资源文件。每个动效因为复杂度不同，目录个数和文件大小也不尽相同。

短视频中的示例代码是从后台下载动效资源，再统一解压到 Resource 目录。您可以在短视频代码中找到动效资源和动效缩略图的下载地址，格式如下：

- https://st1.xiangji.qq.com/yunmaterials/{**动效名**}.zip
- https://st1.xiangji.qq.com/yunmaterials/{**动效名**}.png

> **说明：**
> **建议客户将动效资源放在自己的服务器上，以防短视频变动造成不必要的影响。**

当解压完成后，即可通过以下接口开启动效效果：

```
/**
* 选择动效
*
* @param tmplName: 动效名称
* @param tmplDir: 动效所在目录
*/
- (void)selectMotionTmpl:(NSString *)tmplName inDir:(NSString *)tmplDir;
```

AI 抠背

示例

需要下载 AI 抠背的资源，接口跟动效接口相同：

```
/**
* 选择抠背动效
*
* @param tmplName: 动效名称
* @param tmplDir: 动效所在目录
```

```
*/
- (void)selectMotionTmpl:(NSString *)tmplName inDir:(NSString *)tmplDir;
```

高级美颜接口（大眼、瘦脸等）

您可以通过 TXUGCRecord 的 getBeautyManager 方法获取 TXBeautyManager 对象来进行设置各项美颜参数，其方法如下。

```
/**
* 设置美颜（磨皮）算法
*
* SDK 内部集成了两套风格不同的磨皮算法，一套我们取名叫“光滑”，适用于美女秀场，效果比较明显。
* 另一套我们取名“自然”，磨皮算法更多地保留了面部细节，主观感受上会更加自然。
*
* @param beautyStyle 美颜风格，光滑或者自然，光滑风格磨皮更加明显，适合娱乐场景。
*/
- (void)setBeautyStyle:(TXBeautyStyle)beautyStyle;

/**
* 设置美颜级别
* @param level 美颜级别，取值范围0 - 9； 0表示关闭，1 - 9值越大，效果越明显。
*/
- (void)setBeautyLevel:(float)level;

/**
* 设置美白级别
*
* @param level 美白级别，取值范围0 - 9；0表示关闭，1 - 9值越大，效果越明显。
*/
- (void)setWhitenessLevel:(float)level;

/**
* 设置红润级别
*
* @param level 红润级别，取值范围0 - 9；0表示关闭，1 - 9值越大，效果越明显。
*/
- (void)setRuddyLevel:(float)level;

/**
```

```
* 设置大眼级别（企业版有效，其它版本设置此参数无效）
*
* @param level 大眼级别，取值范围0 - 9；0表示关闭，1 - 9值越大，效果越明显。
*/
- (void)setEyeScaleLevel:(float)level;

/**
* 设置瘦脸级别（企业版有效，其它版本设置此参数无效）
*
* @param level 瘦脸级别，取值范围0 - 9；0表示关闭，1 - 9值越大，效果越明显。
*/
- (void)setFaceSlimLevel:(float)level;

/**
* 设置 V 脸级别（企业版有效，其它版本设置此参数无效）
*
* @param level V 脸级别，取值范围0 - 9；0表示关闭，1 - 9值越大，效果越明显。
*/
- (void)setFaceVLevel:(float)level;

/**
* 设置下巴拉伸或收缩（企业版有效，其它版本设置此参数无效）
*
* @param level 下巴拉伸或收缩级别，取值范围-9 - 9；0 表示关闭，小于0表示收缩，大于0表示拉
伸。
*/
- (void)setChinLevel:(float)level;
/**
* 设置短脸级别（企业版有效，其它版本设置此参数无效）
*
* @param level 短脸级别，取值范围0 - 9；0表示关闭，1 - 9值越大，效果越明显。
*/
- (void)setFaceShortLevel:(float)level;

/**
* 设置瘦鼻级别（企业版有效，其它版本设置此参数无效）
*
* @param level 瘦鼻级别，取值范围0 - 9；0表示关闭，1 - 9值越大，效果越明显。
*/
- (void)setNoseSlimLevel:(float)level;
```

```
/**
* 设置亮眼（企业版有效，其它版本设置此参数无效）
*
* @param level 亮眼级别，取值范围0 - 9；0表示关闭，1 - 9值越大，效果越明显。
*/
- (void)setEyeLightenLevel:(float)level;

/**
* 设置白牙（企业版有效，其它版本设置此参数无效）
*
* @param level 白牙级别，取值范围0 - 9；0表示关闭，1 - 9值越大，效果越明显。
*/
- (void)setToothWhitenLevel:(float)level;

/**
* 设置祛皱（企业版有效，其它版本设置此参数无效）
*
* @param level 祛皱级别，取值范围0 - 9；0表示关闭，1 - 9值越大，效果越明显。
*/
- (void)setWrinkleRemoveLevel:(float)level;

/**
* 设置祛眼袋（企业版有效，其它版本设置此参数无效）
*
* @param level 祛眼袋级别，取值范围0 - 9；0表示关闭，1 - 9值越大，效果越明显。
*/
- (void)setPounchRemoveLevel:(float)level;

/**
* 设置法令纹（企业版有效，其它版本设置此参数无效）
*
* @param level 法令纹级别，取值范围0 - 9；0表示关闭，1 - 9值越大，效果越明显。
*/
- (void)setSmileLinesRemoveLevel:(float)level;

/**
* 设置发际线（企业版有效，其它版本设置此参数无效）
*
* @param level 发际线级别，取值范围-9 - 9；0表示关闭，小于0表示抬高，大于0表示降低。
```

```
*/
- (void)setForeheadLevel:(float)level;

/**
* 设置眼距（企业版有效，其它版本设置此参数无效）
*
* @param level 眼距级别，取值范围-9 - 9；0表示关闭，小于0表示拉伸，大于0表示收缩。
*/
- (void)setEyeDistanceLevel:(float)level;

/**
* 设置眼角（企业版有效，其它版本设置此参数无效）
*
* @param level 眼角级别，取值范围-9 - 9；0表示关闭，小于0表示降低，大于0表示抬高。
*/
- (void)setEyeAngleLevel:(float)level;

/**
* 设置嘴型（企业版有效，其它版本设置此参数无效）
*
* @param level 嘴型级别，取值范围-9 - 9；0表示关闭，小于0表示拉伸，大于0表示收缩。
*/
- (void)setMouthShapeLevel:(float)level;

/**
* 设置鼻翼（企业版有效，其它版本设置此参数无效）
*
* @param level 鼻翼级别，取值范围-9 - 9；0表示关闭，小于0表示拉伸，大于0表示收缩。
*/
- (void)setNoseWingLevel:(float)level;

/**
* 设置鼻子位置（企业版有效，其它版本设置此参数无效）
* @param level 鼻子位置级别，取值范围-9 - 9；0表示关闭，小于0表示抬高，大于0表示降低。
*/
- (void)setNosePositionLevel:(float)level;

/**
* 设置嘴唇厚度（企业版有效，其它版本设置此参数无效）
* @param level 嘴唇厚度级别，取值范围-9 - 9；0表示关闭，小于0表示拉伸，大于0表示收缩。
```

```
*/
- (void)setLipsThicknessLevel:(float)level;

/**
* 设置脸型（企业版有效，其它版本设置此参数无效）
* @param level 美型级别，取值范围0 - 9；0表示关闭，1 - 9值越大，效果越明显。
*/
- (void)setFaceBeautyLevel:(float)level;
```

绿幕功能

使用绿幕需要先准备一个用于播放的 MP4 文件，通过调用以下接口即可开启绿幕效果。

```
/**
* 设置绿幕文件
*
* @param file: 绿幕文件路径。支持 mp4; nil 关闭绿幕
*/
-(void)setGreenScreenFile:(NSURL *)file;
```

问题排查

1. 工程编译不过?

检查 AssetsLibrary.framwork、CoreMedia.framework、Accelerate.framework、Metal.framework 依赖库是否已经添加。

2. 工程运行过程中 crash?

1. 检查工程是否配置了 -ObjC。
2. 检查 Metal API Validation 是否设置成了 Disabled。

3. 工程特效不生效?

1. 检查是否调用了 +[TXUGCBase setLicenceURL:key:] 方法，以及参数是否正确。
2. 调用 TXUGCBase的 getLicenseInfo 方法，带有动效的 Licence 会包含 pituLicense 字段。
3. 检查 pitu 资源是否添加正确，尤其要注意 handdetect、handtrack、res18_3M 三个文件要以 folder refrence 形式添加，最简单的方法就是比对自己工程添加的动效文件是否和我们 demo 添加的完全一致。
4. SDK 会把 Licence 下载到沙盒的 Documents 目录当中, 可以在开发过程中使用 Xcode 菜单中 Window > Devices and Simulators 工具导出并使用 查询工具（单击下载）查看有效期。

查询工具 是一个 xcode 工程，目前仅支持在 mac 上使用， 后续会开放其他查询方式。

大眼瘦脸和挂件(Android)

最近更新时间：2021-12-10 11:24:04

目前，仅原移动直播企业版 SDK，短视频企业版 SDK 以及短视频企业版 Pro SDK 支持高级美颜特效，需要购买原 移动直播企业版 License、短视频企业版 License 或短视频企业版 Pro License 后，使用对应的功能。

功能说明

大眼、瘦脸、动效贴纸、绿幕等特效功能，是基于优图实验室的 AI 识别技术和天天 P 图的美妆技术为基础开发的特权功能，腾讯云小视频团队通过跟优图和 P 图团队合作，将这些特效深度整合到 RTMP SDK 的图像处理流程中，以实现更好的视频特效。

接入流程

单击此处 申请企业版本 License。

版本下载

在 SDK 开发包 页面下方下载商用版本 SDK 压缩包，压缩包有加密（解压密码和 Licence 文件 可以跟我们的商务同学获取），成功解压后在 SDK 目录下得到一个 aar 和 zip，分别对应两种集成方式。

工程设置

具体操作请参见 工程配置。

添加 SDK

使用 aar 方式集成

直接替换工程中的非企业版的 aar，并在 app 目录下的 build.gradle 中修改对应的名称即可。

使用 jar 包方式集成

1. 需要解压 zip，把 libs 下的 so 拷贝到您的 jni 加载路径下。其中跟动效有关的 so 如下：

so		
libYTCommon.so	libnnpack.so	libpitu_device.so
libpitu_tools.so	libWXVoice.so	libgameplay.so
libCameraFaceJNI.so	libYTFaceTrackPro.so	libimage_filter_gpu.so

so		
libimage_filter_common.so	libpitu_voice.so	libvoicechanger_shared.so
libParticleSystem.so	libYTHandDetector.so	libGestureDetectJni.so
libsegmentern.so		

2. 把解压后的 assets 文件夹下的所有资源拷贝到工程的 assets 目录下，包括 asset 根目录下的文件和 camera 文件夹下的文件。

导入 License 文件

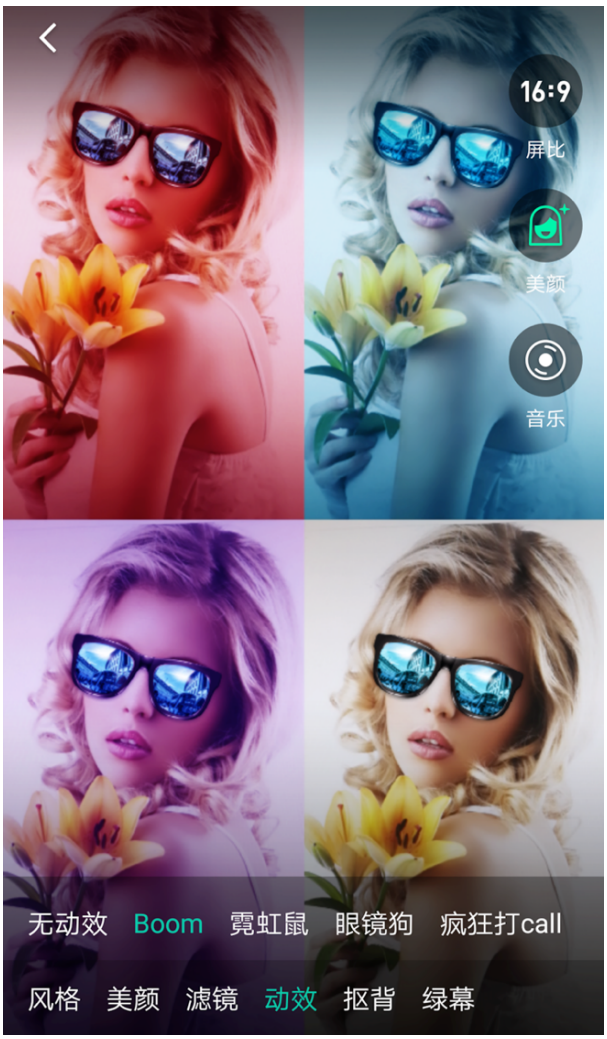
商用版需要 License 验证通过后，相应功能才能生效。您可以向我们的商务同学申请一个免费30天的调试用 License。
得到 Licence 后，您需要将其命名为 **YTFaceSDK.licence**，放到工程的 assets 目录下。

> ⚠ 注意：
>
> - 每个 License 都有绑定具体的 package name，修改 App 的 package name 会导致验证失败。
> - YTFaceSDK.licence 的文件名固定，不可修改、且必须放在 assets 目录下。
> - iOS 和 Android 不需要重复申请 Licence，一个 License 可以同时授权一个 iOS 的 BundleID 和一个 Android 的 PackageName。

从9.4版本开始，SDK 支持二合一的 License，这种方式不再需要 YTFaceSDK.licence，在从商务同学处获取到 License 对应的 Key 和 URL 后，设置方式和标准版 License 设置方式相同。

功能调用

动效功能



一个动效模板是一个目录，里面包含很多资源文件。每个动效因为复杂度不同，目录个数以和文件大小也不尽相同。

Demo 中的示例代码是从后台下载动效资源，再统一解压到 sdcard。您可以在 Demo 代码中找到动效资源的下载地址，格式如下：

```
http://dldir1.qq.com/hudongzhibo/AISpecial/Android/156/(动效name).zip
```

> **说明：**
> 建议您将动效资源放在自己的服务器上，以防 Demo 变动造成不必要的影响。

当解压完成后，即可通过以下接口开启动效效果。

```
/**
* setMotionTmpl 设置动效贴图文件位置
* @param tmplPath
```

```
*/
public void setMotionTmpl(String tmplPath);
```

AI 抠背

需要下载 AI 抠背的资源，接口跟动效接口相同。

```
/**
* setMotionTmpl 设置动效贴图文件位置
* @param tmplPath
*/
public void setMotionTmpl(String tmplPath);
```

高级美颜接口（大眼、瘦脸等）

```
// 大眼效果 0~9
mTXCameraRecord.getBeautyManager().setEyeScaleLevel(eyeLevel);
// 瘦脸效果 0~9
mTXCameraRecord.getBeautyManager().setFaceSlimLevel(faceSlimLevel);
// V脸效果 0~9
mTXCameraRecord.getBeautyManager().setFaceVLevel(faceVLevel);
// 下巴拉伸或收缩效果 0~9
mTXCameraRecord.getBeautyManager().setChinLevel(chinSlimLevel);
// 缩脸效果 0~9
mTXCameraRecord.getBeautyManager().setFaceShortLevel(faceShortLevel);
// 瘦鼻效果 0~9
mTXCameraRecord.getBeautyManager().setNoseSlimLevel(noseScaleLevel);
// 亮眼效果 0~9
mTXCameraRecord.getBeautyManager().setEyeLightenLevel(eyeLightenLevel);
// 白牙效果 0~9
mTXCameraRecord.getBeautyManager().setToothWhitenLevel(toothWhitenLevel);
// 祛皱效果 0~9
mTXCameraRecord.getBeautyManager().setWrinkleRemoveLevel(wrinkleRemoveLevel);
// 祛眼袋效果 0~9
mTXCameraRecord.getBeautyManager().setPounchRemoveLevel(pounchRemoveLevel);
// 祛法令纹效果 0~9
mTXCameraRecord.getBeautyManager().setSmileLinesRemoveLevel(smileLinesRemoveLevel);
// 调整发际线 0~9
mTXCameraRecord.getBeautyManager().setForeheadLevel(foreheadLevel);
// 调整眼间距 0~9
mTXCameraRecord.getBeautyManager().setEyeDistanceLevel(eyeDistanceLevel);
// 调整眼角 0~9
mTXCameraRecord.getBeautyManager().setEyeAngleLevel(eyeAngleLevel);
// 调整嘴形 0~9
mTXCameraRecord.getBeautyManager().setMouthShapeLevel(mouthShapeLevel);
// 调整鼻翼 0~9
mTXCameraRecord.getBeautyManager().setNoseWingLevel(noseWingLevel);
// 调整鼻子位置 0~9
mTXCameraRecord.getBeautyManager().setNosePositionLevel(nosePositionLevel);
// 调整嘴唇厚度 0~9
mTXCameraRecord.getBeautyManager().setLipsThicknessLevel(lipsThicknessLevel);
```

版权所有：腾讯云计算（北京）有限责任公司

```
// 调整脸型 0~9
mTXCameraRecord.getBeautyManager().setFaceBeautyLevel(faceBeautyLevel);
```

绿幕功能

使用绿幕需要先准备一个用于播放的 MP4 文件，通过调用以下接口即可开启绿幕效果。

```
/**
* 设置绿幕文件:目前图片支持 jpg/png，视频支持 mp4/3gp 等 Android 系统支持的格式
* API 要求18
* @param path ： 绿幕文件位置，支持两种方式：
* 1.资源文件放在 assets 目录，path 直接取文件名
* 2.path 取文件绝对路径
*/
@TargetApi(18)
public void setGreenScreenFile(String path);
```

问题排查

1. License 是否正常使用中?

License 设置成功后（需等待一段时间，具体时间视网络情况而定），SDK 会下载 License 文件到手机。可以通过 TXUGCBase 的 getLicenceInfo() 方法查看 Licence 信息，包含 License 的生效和过期时间，绑定的 app package name 信息等。

```
public void onCreate() {
super.onCreate();
String licenceURL = ""; // 获取到的 licenceURL
String licenceKey = ""; // 获取到的 licenceKey
TXUGCBase.getInstance().setLicence(this, licenceURL, licenceKey);// 设置 Licence

// 打印 licence 信息
Handler handler = new Handler(Looper.getMainLooper());
handler.postDelayed(new Runnable() {
@Override
public void run() {
Log.i(TAG, "onCreate: " + TXUGCBase.getInstance().getLicenceInfo(MApplication.this));
}
```

```
}, 5 * 1000);//5秒后打印 Licence 的信息
}
```

若您需要其他协助，可将打印出来的 License 信息保存，并联系我们的 技术支持。

2. 集成遇到异常怎么解决?

```
java.lang.UnsatisfiedLinkError: No implementation found for void com.tencent.ttpic.util.youtu.YTFaceDetectorBase.nativeSetRefine(boolean) (tried Java_com_tencent_ttpic_util_youtu_YTFaceDetectorBase_nativeSetRefine and Java_com_tencent_ttpic_util_youtu_YTFaceDetectorBase_nativeSetRefine__Z)
at com.tencent.ttpic.util.youtu.YTFaceDetectorBase.nativeSetRefine(Native Method)
```

请检查项目 build.gradle 中是否存在多层 module 结构。例如，app module 引用了 ugckit module，ugckit module 引用了腾讯云 SDK，则您需要在 app module 和 ugckit 的 module 中添加如下配置：

```
packagingOptions {
pickFirst '**/libc++_shared.so'
doNotStrip "*/armeabi/libYTCommon.so"
doNotStrip "*/armeabi-v7a/libYTCommon.so"
doNotStrip "*/x86/libYTCommon.so"
doNotStrip "*/arm64-v8a/libYTCommon.so"
}
```

添加配置后，请 clean 工程后再重新 build。

3. 美容（例如大眼瘦脸）、动效等功能不起作用怎么解决?

- 检查移动直播 License 的有效期 TXUGCBase.getInstance().getLicenceInfo(mContext)。
- 检查优图实验室 License 有效期（购买时通过商务获取）。
- 检查您下载的 SDK 版本和购买的 SDK 版本是否一致。

移动直播只有 企业版 支持 AI 特效（大眼瘦脸、V 脸隆鼻、动效贴纸、绿幕抠图）。

如果您调用接口发现不生效，请查看 Logcat 是否存在 log： support EnterPrise above!!! 。如果存在，说明下载的 SDK 版本和您使用的 Licence 版本不匹配。

> ⚠ 注意：

> 美颜动效请使用最新接口 TXUGCRecord getBeautyManager() 。

查询工具 可以查询 License 的有效期，是一个 xcode 工程，目前仅支持在 mac 上使用，后续会开放其他查询方式。

4. 采用动态加载 jar + so 方式集成需要注意什么?

- 检查动态下发的 so 包个数是否存在下发不全的情况，通过 TXLiveBase.setLibraryPath(soPath); 设置 so 包地址。

> ⚠ 注意：
> 不可以一部分放在本地，一部分动态下发，只能全部动态下发或全部本地集成。

- jar + so 方式解压后的资源分为 assets-static和assets-dynamic 两类，其中 assets-static 只能放在本地，不可以动态下发，asset-dynamic 需要保证动态下发，跟 so 同一个目录下。
- SDK 6.8 以后，请不要人为通过系统的方法加载 so 包，SDK 内部会保证 so 包的加载顺序。

如果您出现以下问题，请按上述几点进行检查。

```
YTFaceDetectorBase: (GLThread 5316)
com.tencent.ttpic.util.youtu.YTFaceDetectorBase(54)[c]: nativeInitCommon, ret = -2
YTFaceDetectorBase: (GLThread 5316)com.tencent.ttpic.util.youtu.YTFaceDetectorBase(57)[c]:
nativeInitCommon failed, ret = -1001
YTFaceDetectorBase: (GLThread 5316)com.tencent.ttpic.util.youtu.YTFaceDetectorBase(26)[a]:
initCommon, ret = -1001
YTFaceDetectorBase: (GLThread 5316)com.tencent.ttpic.util.youtu.YTFaceDetectorBase(28)[a]:
initCommon failed, ret = -1001
```

贴纸和字幕
贴纸和字幕(iOS)

最近更新时间：2021-01-27 15:29:30

静态贴纸

```
- (void) setPasterList:(NSArray *)pasterList;

// TXPaster 的参数如下：
@interface TXPaster: NSObject
@property (nonatomic, strong) UIImage* pasterImage; //贴纸图片
@property (nonatomic, assign) CGRect frame; //贴纸 frame（注意这里的 frame 坐标是相对于渲染 view 的坐标）
@property (nonatomic, assign) CGFloat startTime; //贴纸起始时间(s)
@property (nonatomic, assign) CGFloat endTime; //贴纸结束时间(s)
@end
```

动态贴纸

```
- (void) setAnimatedPasterList:(NSArray *)animatedPasterList;

// TXAnimatedPaster 的参数如下：
@interface TXAnimatedPaster: NSObject
@property (nonatomic, strong) NSString* animatedPasterpath; //动图文件路径
@property (nonatomic, assign) CGRect frame; //动图的 frame（注意这里的 frame 坐标是相对于渲染 view 的坐标）
@property (nonatomic, assign) CGFloat rotateAngle; //动图旋转角度 (0 ~ 360)
@property (nonatomic, assign) CGFloat startTime; //动图起始时间(s)
@property (nonatomic, assign) CGFloat endTime; //动图结束时间(s)
@end
```

Demo 示例：

```
- (void)setVideoPasters:(NSArray*)videoPasterInfos
{
```

```
NSMutableArray* animatePasters = [NSMutableArray new];
NSMutableArray* staticPasters = [NSMutableArray new];
for (VideoPasterInfo* pasterInfo in videoPasterInfos) {
if (pasterInfo.pasterInfoType == PasterInfoType_Animate) {
TXAnimatedPaster* paster = [TXAnimatedPaster new];
paster.startTime = pasterInfo.startTime;
paster.endTime = pasterInfo.endTime;
paster.frame = [pasterInfo.pasterView pasterFrameOnView:_videoPreview];
paster.rotateAngle = pasterInfo.pasterView.rotateAngle * 180 / M_PI;
paster.animatedPasterpath = pasterInfo.path;
[animatePasters addObject:paster];
}
else if (pasterInfo.pasterInfoType == PasterInfoType_static){
TXPaster *paster = [TXPaster new];
paster.startTime = pasterInfo.startTime;
paster.endTime = pasterInfo.endTime;
paster.frame = [pasterInfo.pasterView pasterFrameOnView:_videoPreview];
paster.pasterImage = pasterInfo.pasterView.staticImage;
[staticPasters addObject:paster];
}
}
[_ugcEditer setAnimatedPasterList:animatePasters];
[_ugcEditer setPasterList:staticPasters];
}
```

自定义动态贴纸

动态贴纸的本质是：将一串图片，按照一定的顺序以及时间间隔，插入到视频画面中去，形成一个动态贴纸的效果。

如何自定义贴纸?

以工具包 Demo 中一个动态贴纸为例：

```
{
"name":"glass", // 贴纸名称
"count":6, // 贴纸数量
"period":480, // 播放周期：播放一次动态贴纸的所需要的时间(ms)
"width":444, // 贴纸宽度
"height":256, // 贴纸高度
```

```
"keyframe":6, // 关键图片：能够代表该动态贴纸的一张图片
"frameArray": [ // 所有图片的集合
{"picture":"glass0"},
{"picture":"glass1"},
{"picture":"glass2"},
{"picture":"glass3"},
{"picture":"glass4"},
{"picture":"glass5"}
]
}
```

SDK 内部将获取到该动态贴纸对应的 config.json，并且按照 json 中定义的格式进行动态贴纸的展示。

> 说明：
> 该封装格式为 SDK 内部强制性要求，请严格按照该格式描述动态贴纸。

添加字幕

1. 气泡字幕

您可以为视频添加字幕，我们支持对每一帧视频添加字幕，每个字幕您也可以设置视频作用的起始时间和结束时间。所有的字幕组成了一个字幕列表， 您可以把字幕列表传给 SDK 内部，SDK 会自动在合适的时间对视频和字幕做叠加。

设置字幕的方法为：

```
- (void) setSubtitleList:(NSArray *)subtitleList;

TXSubtitle 的参数如下：
@interface TXSubtitle: NSObject
@property (nonatomic, strong) UIImage* titleImage; //字幕图片 （这里需要客户把承载文字的控件转成 image 图片）
@property (nonatomic, assign) CGRect frame; //字幕的 frame（注意这里的 frame 坐标是相对于渲染 view 的坐标）
@property (nonatomic, assign) CGFloat startTime; //字幕起始时间(s)
@property (nonatomic, assign) CGFloat endTime; //字幕结束时间(s)
@end
```

- titleImage：表示字幕图片，如果上层使用的是 UILabel 之类的控件，请先把控件转成 UIImage，具体方法可以参照 demo 的示例代码。
- frame：表示字幕的 frame，注意这个 frame 是相对于渲染 view（initWithPreview 时候传入的 view）的 frame，具体可以参照 demo 的示例代码。
- startTime：字幕作用的起始时间。
- endTime：字幕作用的结束时间。

因为字幕这一块的 UI 逻辑比较复杂，我们已经在 demo 层有一整套的实现方法，推荐客户直接参考 demo 实现， 可以大大降低您的接入成本。

Demo 示例：

```
@interface VideoTextInfo : NSObject
@property (nonatomic, strong) VideoTextFiled* textField;
@property (nonatomic, assign) CGFloat startTime; //in seconds
@property (nonatomic, assign) CGFloat endTime;
@end

videoTextInfos = @[VideoTextInfo1, VideoTextInfo2 ...];

for (VideoTextInfo* textInfo in videoTextInfos) {
TXSubtitle* subtitle = [TXSubtitle new];
subtitle.titleImage = textInfo.textField.textImage; //UILabel（UIView） -> UIImage
subtitle.frame = [textInfo.textField textFrameOnView:_videoPreview]; //计算相对于渲染 view 的坐标
subtitle.startTime = textInfo.startTime; //字幕起始时间
subtitle.endTime = textInfo.endTime; //字幕结束时间
[subtitles addObject:subtitle]; //添加字幕列表
}

[_ugcEditer setSubtitleList:subtitles]; //设置字幕列表
```

2. 如何自定义气泡字幕?

气泡字幕所需要的参数

- 文字区域大小： top、left、right、bottom
- 默认的字体大小
- 宽、高

> 说明：
> 以上单位均为 px。

封装格式

由于气泡字幕中携带参数较多，我们建议您可以在 Demo 层封装相关的参数。如腾讯云 Demo 中使用的 json 格式封装：

```
{
"name":"boom", // 气泡字幕名称
"width": 376, // 宽度
"height": 335, // 高度
"textTop":121, // 文字区域上边距
"textLeft":66, // 文字区域左边距
"textRight":69, // 文字区域右边距
"textBottom":123, // 文字区域下边距
"textSize":40 // 字体大小
}
```

> 说明：
> 该封装格式用户可以自行决定，非 SDK 强制性要求。

字幕过长

字幕若输入过长时，如何进行排版才能够使字幕与气泡美观地合并?

我们在 Demo 中提供了一个自动排版的控件。若在当前字体大小下，字幕过长时，控件将自动缩小字号，直到能够恰好放下所有字幕文字为止。

您也可以修改相关控件源代码，来满足自身的业务要求。

版权所有：腾讯云计算（北京）有限责任公司

贴纸和字幕(Android)

最近更新时间：2021-01-27 15:37:46

静态贴纸

设置静态贴纸的方法：

```
public void setPasterList(List pasterList);

// TXPaster 的参数如下：
public final static class TXPaster {
public Bitmap pasterImage; // 贴纸图片
public TXRect frame; // 贴纸的 frame（注意这里的 frame 坐标是相对于渲染 view 的坐标）
public long startTime; // 贴纸起始时间(ms)
public long endTime; // 贴纸结束时间(ms)
}
```

动态贴纸

设置动态贴纸的方法：

```
public void setAnimatedPasterList(List animatedPasterList);

// TXAnimatedPaster 的参数如下：
public final static class TXAnimatedPaster {
public String animatedPasterPathFolder; // 动态贴纸图片地址
public TXRect frame; // 动态贴纸 frame (注意这里的 frame 坐标是相对于渲染 view 的坐标)
public long startTime; // 动态贴纸起始时间(ms)
public long endTime; // 动态贴纸结束时间(ms)
public float rotation;
}
```

Demo示例：

```
List animatedPasterList = new ArrayList<>();
List pasterList = new ArrayList<>();
for (int i = 0; i < mTCLayerViewGroup.getChildCount(); i++) {
```

版权所有：腾讯云计算（北京）有限责任公司

```
PasterOperationView view = (PasterOperationView) mTCLayerViewGroup.getOperationView
(i);
TXVideoEditConstants.TXRect rect = new TXVideoEditConstants.TXRect();
rect.x = view.getImageX();
rect.y = view.getImageY();
rect.width = view.getImageWidth();
TXCLog.i(TAG, "addPasterListVideo, adjustPasterRect, paster x y = " + rect.x + "," + rect.y);

int childType = view.getChildType();
if (childType == PasterOperationView.TYPE_CHILD_VIEW_ANIMATED_PASTER) {
TXVideoEditConstants.TXAnimatedPaster txAnimatedPaster = new TXVideoEditConstants.TX
AnimatedPaster();

txAnimatedPaster.animatedPasterPathFolder = mAnimatedPasterSDcardFolder + view.getPa
sterName() + File.separator;
txAnimatedPaster.startTime = view.getStartTime();
txAnimatedPaster.endTime = view.getEndTime();
txAnimatedPaster.frame = rect;
txAnimatedPaster.rotation = view.getImageRotate();

animatedPasterList.add(txAnimatedPaster);
TXCLog.i(TAG, "addPasterListVideo, txAnimatedPaster startTimeMs, endTime is : " + txAnima
tedPaster.startTime + ", " + txAnimatedPaster.endTime);
} else if (childType == PasterOperationView.TYPE_CHILD_VIEW_PASTER) {
TXVideoEditConstants.TXPaster txPaster = new TXVideoEditConstants.TXPaster();

txPaster.pasterImage = view.getRotateBitmap();
txPaster.startTime = view.getStartTime();
txPaster.endTime = view.getEndTime();
txPaster.frame = rect;

pasterList.add(txPaster);
TXCLog.i(TAG, "addPasterListVideo, txPaster startTimeMs, endTime is : " + txPaster.startTime
+ ", " + txPaster.endTime);
}
}

mTXVideoEditer.setAnimatedPasterList(animatedPasterList); //设置动态贴纸
mTXVideoEditer.setPasterList(pasterList); //设置静态贴纸
```

自定义动态贴纸

动态贴纸的本质是：将**一串图片**，按照**一定的顺序**以及**时间间隔**，插入到视频画面中去，形成一个动态贴纸的效果。

封装格式

以 Demo 中一个动态贴纸为例：

```
{
"name":"glass", // 贴纸名称
"count":6, // 贴纸数量
"period":480, // 播放周期：播放一次动态贴纸的所需要的时间(ms)
"width":444, // 贴纸宽度
"height":256, // 贴纸高度
"keyframe":6, // 关键图片：能够代表该动态贴纸的一张图片
"frameArray": [ // 所有图片的集合
{"picture":"glass0"},
{"picture":"glass1"},
{"picture":"glass2"},
{"picture":"glass3"},
{"picture":"glass4"},
{"picture":"glass5"}
]
}
```

SDK 内部将获取到该动态贴纸对应的 config.json，并且按照 json 中定义的格式进行动态贴纸的展示。

> 说明：
> 该封装格式为 SDK 内部强制性要求，请严格按照该格式描述动态贴纸。

添加字幕

1. 气泡字幕

您可以为视频设置气泡字幕，我们支持对每一帧视频添加字幕，每个字幕您也可以设置视频作用的起始时间和结束时间。所有的字幕组成了一个字幕列表， 您可以把字幕列表传给 SDK 内部，SDK 会自动在合适的时间对视频和字幕做叠加。

设置气泡字幕的方法为：

```
public void setSubtitleList(List subtitleList);

//TXSubtitle 的参数如下：
public final static class TXSubtitle {
public Bitmap titleImage; // 字幕图片
public TXRect frame; // 字幕的 frame
public long startTime; // 字幕起始时间(ms)
public long endTime; // 字幕结束时间(ms)
}

public final static class TXRect {
public float x;
public float y;
public float width;
}
```

- titleImage：表示字幕图片，如果上层使用的是 TextView 之类的控件，请先把控件转成 Bitmap，具体方法可以参照 demo 的示例代码。
- frame：表示字幕的 frame，注意这个 frame 是相对于渲染 view（initWithPreview 时候传入的 view）的 frame，具体可以参照 demo 的示例代码。
- startTime：字幕作用的起始时间。
- endTime：字幕作用的结束时间。

因为字幕的 UI 逻辑比较复杂，我们已经在 demo 层有一整套的实现方法，推荐客户直接参考 demo 实现， 可以大大降低您的接入成本。

Demo 示例：

```
mSubtitleList.clear();
for (int i = 0; i < mWordInfoList.size(); i++) {
TCWordOperationView view = mOperationViewGroup.getOperationView(i);
TXVideoEditConstants.TXSubtitle subTitle = new TXVideoEditConstants.TXSubtitle();
subTitle.titleImage = view.getRotateBitmap(); //获取Bitmap
TXVideoEditConstants.TXRect rect = new TXVideoEditConstants.TXRect();
rect.x = view.getImageX(); // 获取相对 parent view 的 x 坐标
rect.y = view.getImageY(); // 获取相对 parent view 的 y 坐标
rect.width = view.getImageWidth(); // 图片宽度
subTitle.frame = rect;
```

```
subTitle.startTime = mWordInfoList.get(i).getStartTime(); // 设置开始时间
subTitle.endTime = mWordInfoList.get(i).getEndTime(); // 设置结束时间
mSubtitleList.add(subTitle);
}
mTXVideoEditer.setSubtitleList(mSubtitleList); // 设置字幕列表
```

2.自定义气泡字幕?

气泡字幕所需要的参数

- 文字区域大小： top、left、right、bottom
- 默认的字体大小
- 宽、高

> 说明：
> 以上单位均为 px。

封装格式

由于气泡字幕中携带参数较多，我们建议您可以在 Demo 层封装相关的参数。如腾讯云 Demo 中使用的 json 格式封装。

```
{
"name":"boom", // 气泡字幕名称
"width": 376, // 宽度
"height": 335, // 高度
"textTop":121, // 文字区域上边距
"textLeft":66, // 文字区域左边距
"textRight":69, // 文字区域右边距
"textBottom":123, // 文字区域下边距
"textSize":40 // 字体大小
}
```

> 说明：
> 该封装格式用户可以自行决定，非 SDK 强制性要求。

字幕过长

字幕若输入过长时，如何进行排版才能够使字幕与气泡美观地合并?

我们在 Demo 中提供了一个自动排版的控件。若在当前字体大小下，字幕过长时，控件将自动缩小字号，直到能够恰好放下所有字幕文字为止。

您也可以修改相关控件源代码，来满足自身的业务要求。

版权所有：腾讯云计算（北京）有限责任公司

视频合唱
视频合唱(iOS)

最近更新时间：2021-01-27 15:57:08

本篇教程向大家介绍如何从零开始完成合唱的基础功能。

过程简介

1. 在界面上放两个 View， 一个用来播放，一个用来录制。
2. 再放一个按钮和进度条来开始录制和显示进度。
3. 录制与源视频相同的时长后停止。
4. 把录好的视频与源视频左右合成。
5. 预览合成好的视频。

界面搭建

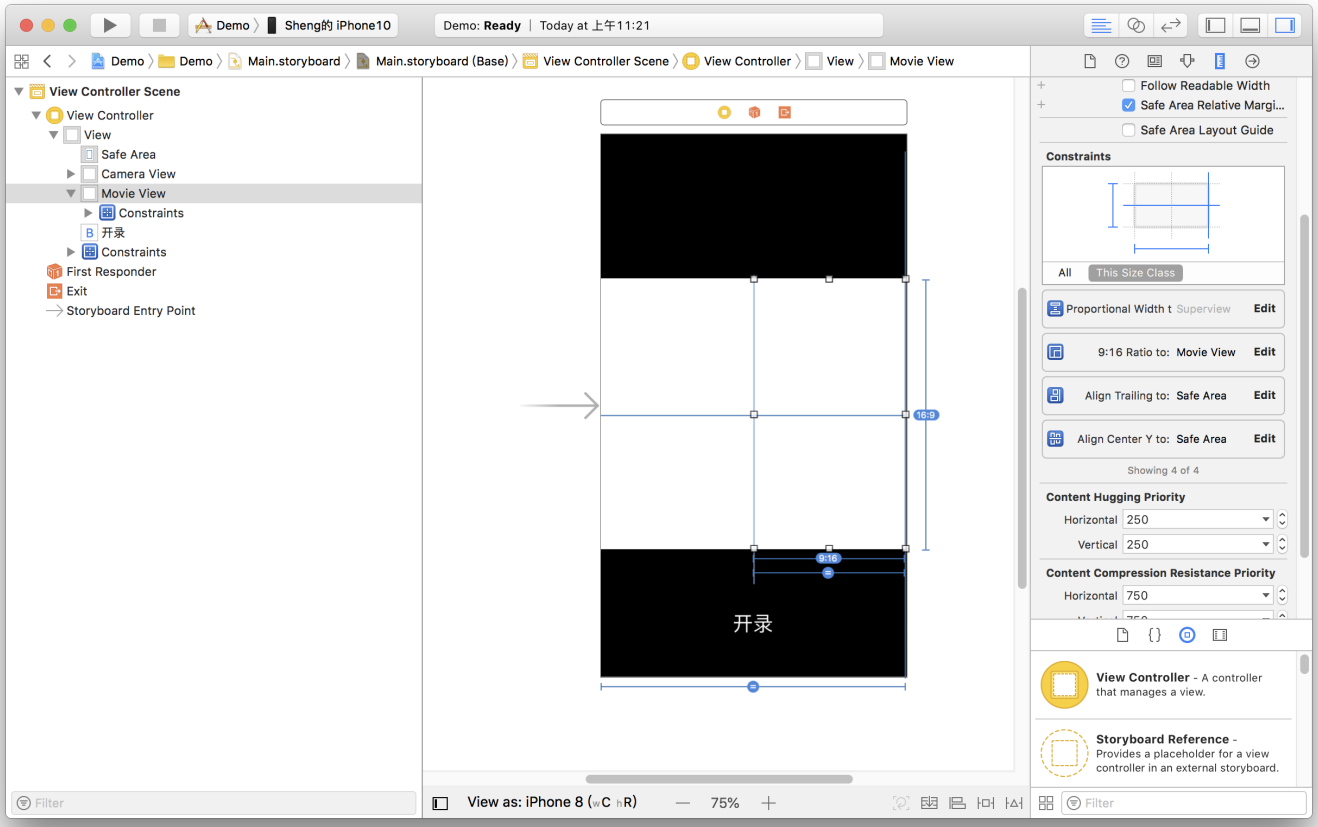
首先来开始工程的创建，打开 Xcode，File > New > Project，然后起好工程名创建工程，这里方便起见叫做 Demo，因为要录像，所以我们需要相机和麦克风的权限，在 Info 中配置一下增加以下两项：

```
Privacy - Microphone Usage Description
Privacy - Camera Usage Description
```

这两项的值可以随便填写，如"录制视频"。

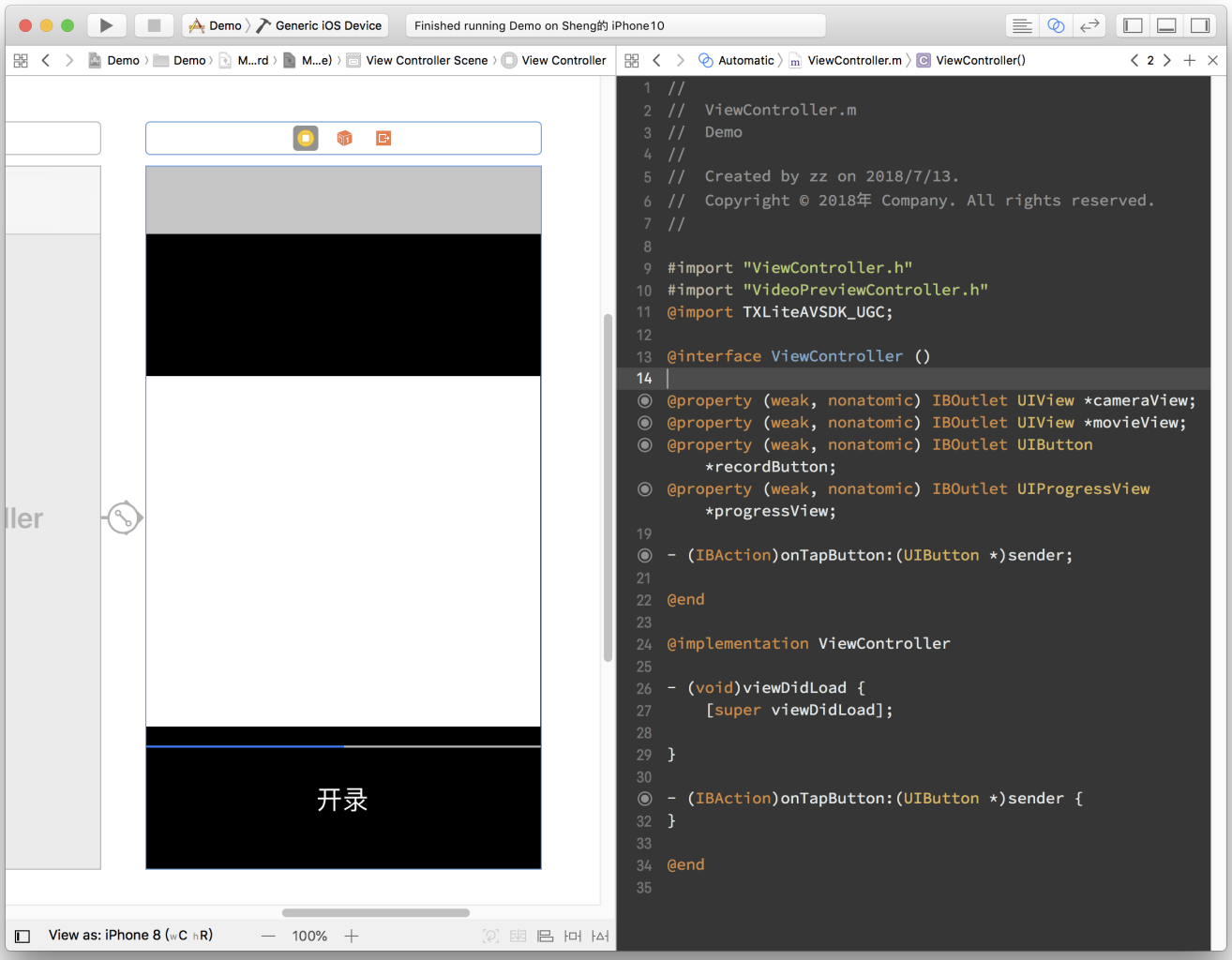
接下来我们配置一个简单的录制界面，打开 Main.storyboard，拖进去两个 UIView，配置宽度为 superview 的0.5倍，长宽比16:9。

版权所有：腾讯云计算（北京）有限责任公司



然后加上进度条，在 ViewController.m 中设置 IBOutlet 绑定界面，并设置好按钮的 IBAction。因为录制好后我们还要跳转到预览界面，还需要一个导航，单击黄色 VC 图标，在菜单栏依次进入 Editor > Embeded In，

单击 Navigation Controller 给 ViewController 套一层 Navigation Controller。完成基本 UI 的搭建。



代码部分

对于合唱功能主要使用三大块功能：播放、录制、以及录制后和原视频进行合成，这三个功能对应到 SDK 的类为： TXVideoEditer、TXUGCRecord、TXVideoJoiner。

在使用前要配置 SDK 的 Licence，打开 AppDelegate.m 在里面添加以下代码：

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
[TXUGCBase setLicenceURL:@"<Licence的URL>" key:@"<Licence的Key>"];
return YES;
}
```

这里的 Licence 参数需要到 短视频控制台 去申请，提交申请后一般很快就会审批下来。然后页面上就会有相关的信息。

1. 首先是声明与初始化。

   打开 ViewContorller.m，引用 SDK 并声明上述三个类的实例。另外这里播放、录制和合成视频都是异步操作，需要监听他们的事件，所以要加上实现 TXVideoJoinerListener、TXUGCRecordListener、TXVideoPreviewListener 这三个协议的声明。加好后如下所示：

```
#import "ViewController.h"
@import TXLiteAVSDK_UGC;
@interface ViewController () <txvideojoinerlistener, txugcrecordlistener, txvideopreviewlis
tener>
{
TXVideoEditer *_editor;
TXUGCRecord *_recorder;
TXVideoJoiner *_joiner;
TXVideoInfo *_videoInfo;

NSString *_recordPath;
NSString *_resultPath;
}
@property (weak, nonatomic) IBOutlet UIView *cameraView;
@property (weak, nonatomic) IBOutlet UIView *movieView;
@property (weak, nonatomic) IBOutlet UIButton *recordButton;
@property (weak, nonatomic) IBOutlet UIProgressView *progressView;
- (IBAction)onTapButton:(UIButton *)sender;
@end
```

   准备好成员变量和接口实现声明后，我们在 viewDidLoad 中对上面的成员变量进行初始化。

```
- (void)viewDidLoad {
[super viewDidLoad];
// 这里找一段 mp4 视频放到了工程里，或者用手机录制的 mov 格式视频也可以
NSString *mp4Path = [[NSBundle mainBundle] pathForResource:@"demo" ofType:@"mp4"
];
_videoInfo = [TXVideoInfoReader getVideoInfo:mp4Path];
TXAudioSampleRate audioSampleRate = AUDIO_SAMPLERATE_48000;
if (_videoInfo.audioSampleRate == 8000) {
audioSampleRate = AUDIO_SAMPLERATE_8000;
```

```
}else if (_videoInfo.audioSampleRate == 16000){
audioSampleRate = AUDIO_SAMPLERATE_16000;
}else if (_videoInfo.audioSampleRate == 32000){
audioSampleRate = AUDIO_SAMPLERATE_32000;
}else if (_videoInfo.audioSampleRate == 44100){
audioSampleRate = AUDIO_SAMPLERATE_44100;
}else if (_videoInfo.audioSampleRate == 48000){
audioSampleRate = AUDIO_SAMPLERATE_48000;
}

// 设置录像的保存路径
_recordPath = [NSTemporaryDirectory() stringByAppendingPathComponent:@"record.mp
4"];
_resultPath = [NSTemporaryDirectory() stringByAppendingPathComponent:@"result.mp4"
];

// 播放器初始化
TXPreviewParam *param = [[TXPreviewParam alloc] init];
param.videoView = self.movieView;
param.renderMode = RENDER_MODE_FILL_EDGE;
_editor = [[TXVideoEditer alloc] initWithPreview:param];
[_editor setVideoPath:mp4Path];
_editor.previewDelegate = self;

// 录像参数初始化
_recorder = [TXUGCRecord shareInstance];
TXUGCCustomConfig *recordConfig = [[TXUGCCustomConfig alloc] init];
recordConfig.videoResolution = VIDEO_RESOLUTION_720_1280;
//这里保证录制视频的帧率和合唱视频的帧率一致，否则可能出现音画不同步的现象
//注意：这里获取的合唱视频的帧率是平均帧率，有可能为小数，做一下四舍五入操作
recordConfig.videoFPS = (int)(_videoInfo.fps + 0.5);
//这里保证录制视频的音频采样率和合唱视频的音频采样率一致，否则可能出现音画不同步的现象
recordConfig.audioSampleRate = audioSampleRate;
recordConfig.videoBitratePIN = 9600;
recordConfig.maxDuration = _videoInfo.duration;
_recorder.recordDelegate = self;

// 启动相机预览
[_recorder startCameraCustom:recordConfig preview:self.cameraView];
```

```
// 视频拼接
_joiner = [[TXVideoJoiner alloc] initWithPreview:nil];
_joiner.joinerDelegate = self;
[_joiner setVideoPathList:@[_recordPath, mp4Path]];
}
```

2. 接下来是录制部分，只要响应用户单击按钮调用 SDK 方法就可以了，为了方便起见，这里复用了这个按钮来显示当前状态。另外加上在进度条上显示进度的逻辑。

```
- (IBAction)onTapButton:(UIButton *)sender {
[_editor startPlayFromTime:0 toTime:_videoInfo.duration];
if ([_recorder startRecord:_recordPath coverPath:[_recordPath stringByAppendingString:
@".png"]] != 0) {
NSLog(@"相机启动失败");
}
[sender setTitle:@"录像中" forState:UIControlStateNormal];
sender.enabled = NO;
}
#pragma mark TXVideoPreviewListener
-(void) onPreviewProgress:(CGFloat)time
{
self.progressView.progress = time / _videoInfo.duration;
}
```

3. 录制好后开始完成拼接部分，这里需要指定两个视频在结果中的位置，这里设置一左一右。

```
-(void)onRecordComplete:(TXUGCRecordResult*)result;
{
NSLog(@"录制完成，开始合成");
[self.recordButton setTitle:@"合成中..." forState:UIControlStateNormal];

//获取录制视频的宽高
TXVideoInfo *videoInfo = [TXVideoInfoReader getVideoInfo:_recordPath];
CGFloat width = videoInfo.width;
CGFloat height = videoInfo.height;

//录制视频和原视频左右排列
CGRect recordScreen = CGRectMake(0, 0, width, height);
```

```
CGRect playScreen = CGRectMake(width, 0, width, height);
[_joiner setSplitScreenList:@[[NSValue valueWithCGRect:recordScreen],[NSValue valueWit
hCGRect:playScreen]] canvasWidth:width * 2 canvasHeight:height];
[_joiner splitJoinVideo:VIDEO_COMPRESSED_720P videoOutputPath:_resultPath];
}
```

4. 实现合成进度的委托方法，在进度条中显示进度。

```
-(void) onJoinProgress:(float)progress
{
NSLog(@"视频合成中%d%%",(int)(progress * 100));
self.progressView.progress = progress;
}
```

5. 实现合成完成的委托方法，并切换到预览界面。

```
#pragma mark TXVideoJoinerListener
-(void) onJoinComplete:(TXJoinerResult *)result
{
NSLog(@"视频合成完毕");
VideoPreviewController *controller = [[VideoPreviewController alloc] initWithVideoPath:_re
sultPath];
[self.navigationController pushViewController:controller animated:YES];
}
```

此就制作完成了，上面提到了一个视频预览的 VideoPreviewController 代码如下：

- VideoPreviewController.h

```
#import <uikit uikit.h>
@interface VideoPreviewController : UIViewController
- (instancetype)initWithVideoPath:(NSString *)path;
@end
```

- VideoPreviewController.m:

```
@import TXLiteAVSDK_UGC;
@interface VideoPreviewController () <txvideopreviewlistener>
{
```

```
TXVideoEditer *_editor;
}
@property (strong, nonatomic) NSString *videoPath;
@end
@implementation VideoPreviewController
- (instancetype)initWithVideoPath:(NSString *)path {
if (self = [super initWithNibName:nil bundle:nil]) {
self.videoPath = path;
}
return self;
}
- (void)viewDidLoad {
[super viewDidLoad];
TXPreviewParam *param = [[TXPreviewParam alloc] init];
param.videoView = self.view;
param.renderMode = RENDER_MODE_FILL_EDGE;
_editor = [[TXVideoEditer alloc] initWithPreview:param];
_editor.previewDelegate = self;
[_editor setVideoPath:self.videoPath];
[_editor startPlayFromTime:0 toTime:[TXVideoInfoReader getVideoInfo:self.videoPath].d
uration];
}
-(void) onPreviewFinished
{
[_editor startPlayFromTime:0 toTime:[TXVideoInfoReader getVideoInfo:self.videoPath].d
uration];
}
@end
```

至此就完成了全部合唱的基础功能，功能更加丰富的示例可以参考 小视频源码。

视频合唱(Android)

最近更新时间：2021-01-27 15:59:41

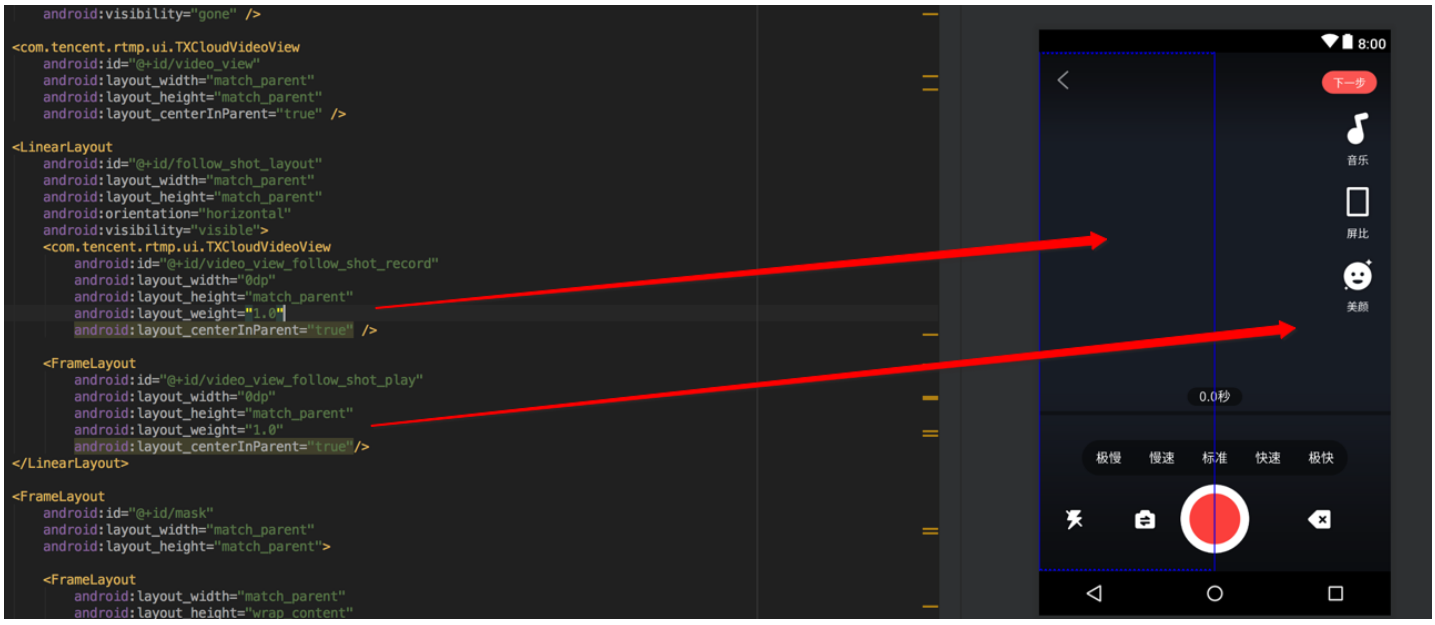
本篇教程向大家介绍如何完成合唱的基础功能。

过程简介

1. 在界面上放两个 View, 一个用来播放，一个用来录制。
2. 再放一个按钮和进度条来开始录制和显示进度。
3. 录制与源视频相同的时长后停止。
4. 把录好的视频与源视频左右合成。
5. 预览合成好的视频。

界面搭建

在录制界面 TCVideoRecordActivity 的 activity_video_record.xml 中创建两个 view，左半边是录制界面，右半边是播放界面。



代码部分

对于合唱功能主要使用三大块功能：播放、录制、以及录制后和原视频进行合成，这三个功能对应到 SDK 的类为： TXVideoEditer、TXUGCRecord、TXVideoJoiner，其中播放也可以换成 TXVodPlayer 去播放。

1. 从小视频主页的视频列表中，选择一个视频进入播放界面 TCVodPlayerActivity，单击右下角的“合拍”按钮。

首先会下载该视频到本地 sdcard 中，并获取该视频的音频采样率以及 fps 等信息后进入录制界面。

2. 进入录制界面 TCVideoRecordActivity 进行合唱。需要注意以下几点：
   - 录制进度条以跟拍视频的进度为最大长度。
   - 保证录制视频的帧率和合唱视频的帧率一致，否则可能出现音画不同步的现象。
   - 保证录制视频的音频采样率和合唱视频的音频采样率一致，否则可能出现音画不同步的现象。
   - 录制设置渲染模式为自适应模式，在9:16的宽高比时能等比例缩放。
   - Android 的录制需要设置静音，否则会造成与跟拍视频的“二重唱”。

```
// 录制的界面
mVideoView = mVideoViewFollowShotRecord;
// 播放的视频
mFollowShotVideoPath = intent.getStringExtra(TCConstants.VIDEO_EDITER_PATH);
mFollowShotVideoDuration = (int)(intent.getFloatExtra(TCConstants.VIDEO_RECORD_D
URATION, 0) * 1000);
initPlayer();
// 录制进度条以跟拍视频的进度为最大长度，fps 以跟拍视频的 fps 为准
mMaxDuration = (int)mFollowShotVideoDuration;
mFollowShotVideoFps = intent.getIntExtra(TCConstants.RECORD_CONFIG_FPS, 20);
mFollowShotAudioSampleRateType = intent.getIntExtra(TCConstants.VIDEO_RECORD_A
UDIO_SAMPLE_RATE_TYPE, TXRecordCommon.AUDIO_SAMPLERATE_48000);
// 初始化合拍的接口
mTXVideoJoiner = new TXVideoJoiner(this);
mTXVideoJoiner.setVideoJoinerListener(this);
```

```
// 播放器初始化，这里使用 TXVideoEditer，也可以使用 TXVodPlayer
mTXVideoEditer = new TXVideoEditer(this);
mTXVideoEditer.setVideoPath(mFollowShotVideoPath);
TXVideoEditConstants.TXPreviewParam param = new TXVideoEditConstants.TXPreview
Param();
param.videoView = mVideoViewPlay;
param.renderMode = TXVideoEditConstants.PREVIEW_RENDER_MODE_FILL_EDGE;
mTXVideoEditer.initWithPreview(param);
```

```
customConfig.videoFps = mFollowShotVideoFps;
customConfig.audioSampleRate = mFollowShotAudioSampleRateType; // 录制的视频的音
频采样率必须与跟拍的音频采样率相同
customConfig.needEdit = false;
```

```
mTXCameraRecord.setVideoRenderMode(TXRecordCommon.VIDEO_RENDER_MODE_ADJUST_RESOLUTION); // 设置渲染模式为自适应模式
mTXCameraRecord.setMute(true); // 跟拍不从喇叭录制声音，因为跟拍的视频声音也会从喇叭发出来被麦克风录制进去，造成跟原视频声音的"二重唱"。
```

3. 接下来就可以开始录制了，在录制到最大长度后，会回调 onRecordComplete，继续完成拼接部分，这里需要指定两个视频在结果中的位置。

```
private void prepareToJoiner(){
List<string> videoSourceList = new ArrayList<>();
videoSourceList.add(mRecordVideoPath);
videoSourceList.add(mFollowShotVideoPath);
mTXVideoJoiner.setVideoPathList(videoSourceList);
mFollowShotVideoOutputPath = getCustomVideoOutputPath("Follow_Shot_");
// 以左边录制的视频宽高为基准，右边视频等比例缩放
int followVideoWidth;
int followVideoHeight;
if ((float) followVideoInfo.width / followVideoInfo.height >= (float)recordVideoInfo.width / recordVideoInfo.height) {
followVideoWidth = recordVideoInfo.width;
followVideoHeight = (int) ((float)recordVideoInfo.width * followVideoInfo.height / followVideoInfo.width);
} else {
followVideoWidth = (int) ((float)recordVideoInfo.height * followVideoInfo.width / followVideoInfo.height);
followVideoHeight = recordVideoInfo.height;
}

TXVideoEditConstants.TXAbsoluteRect rect1 = new TXVideoEditConstants.TXAbsoluteRect();
rect1.x = 0; //第一个视频的左上角位置
rect1.y = 0;
rect1.width = recordVideoInfo.width; //第一个视频的宽高
rect1.height = recordVideoInfo.height;

TXVideoEditConstants.TXAbsoluteRect rect2 = new TXVideoEditConstants.TXAbsoluteRect();
rect2.x = rect1.x + rect1.width; //第2个视频的左上角位置
rect2.y = (recordVideoInfo.height - followVideoHeight) / 2;
```

```
rect2.width = followVideoWidth; //第2个视频的宽高
rect2.height = followVideoHeight;

List<txvideoeditconstants.txabsoluterect> list = new ArrayList<>();
list.add(rect1);
list.add(rect2);
mTXVideoJoiner.setSplitScreenList(list, recordVideoInfo.width + followVideoWidth, recordVi
deoInfo.height); //第2、3个 param：两个视频合成画布的宽高
mTXVideoJoiner.splitJoinVideo(TXVideoEditConstants.VIDEO_COMPRESSED_540P, mFollow
ShotVideoOutputPath);
}
```

4. 监听合成的回调，在 onJoinComplete 后跳转到预览界面播放。

```
@Override
public void onJoinComplete(TXVideoEditConstants.TXJoinerResult result) {
mCompleteProgressDialog.dismiss();
if(result.retCode == TXVideoEditConstants.JOIN_RESULT_OK){
runOnUiThread(new Runnable() {
@Override
public void run() {
isReadyJoin = true;
startEditerPreview(mFollowShotVideoOutputPath);
if(mTXVideoEditer != null){
mTXVideoEditer.release();
mTXVideoEditer = null;
}
}
});
}else{
runOnUiThread(new Runnable() {
@Override
public void run() {
Toast.makeText(TCVideoRecordActivity.this, "合成失败", Toast.LENGTH_SHORT).show();
}
});
}
}
```

至此就完成了全部合唱的基础功能，完整代码可以参考 小视频源码。

图片转场特效
图片转场(iOS)

最近更新时间：2021-01-27 15:22:16

SDK 在4.7版本后增加了图片编辑功能，用户可以选择自己喜欢的图片，添加转场动画、BGM、贴纸等效果。
接口函数如下：

```
/*
*pitureList：转场图片列表,至少设置三张图片（tips：图片最好压缩到720P以下（参考 demo 用法），否则内存占用可能过大，导致编辑过程异常）
*fps：转场图片生成视频后的 fps（15 - 30）
* 返回值：
* 0：设置成功；
* -1：设置失败，请检查图片列表是否存在，图片数量是否大于等于3张，fps 是否正常；
*/
- (int)setPictureList:(NSArray<uiimage *> *)pitureList fps:(int)fps;

/*
*transitionType：转场类型，详情见 TXTransitionType
* 返回值：
* duration：转场视频时长（tips：同一个图片列表，每种转场动画的持续时间可能不一样，这里可以获取转场图片的持续时长）；
*/
- (void)setPictureTransition:(TXTransitionType)transitionType duration:(void(^)(CGFloat))duration;
```

- setPictureList 接口用于设置图片列表，最少设置三张，如果设置的图片过多，要注意图片的大小，防止内存占用过多而导致编辑异常。
- setPictureTransition 接口用于设置转场的效果，目前提供了6种转场效果供用户设置，每种转场效果持续的时长可能不一样，这里可以通过 duration 获取转场的时长。
- 需要注意接口调用顺序，先调用 setPictureList，再调用 setPictureTransition。
- 图片编辑暂不支持的功能：重复、倒放、快速/慢速、片尾水印。其他视频相关的编辑功能，图片编辑均支持，调用方法和视频编辑完全一样。

版权所有：腾讯云计算（北京）有限责任公司

图片转场(Android)

最近更新时间：2021-01-27 15:21:05

SDK 在4.9版本后增加了图片编辑功能，用户可以选择自己喜欢的图片，添加转场动画、BGM、贴纸等效果。
接口函数如下：

```
/*
* bitmapList：转场图片列表,至少设置三张图片（tips：图片最好压缩到720P以下（参考 demo 用法），否则内存占用可能过大，导致编辑过程异常）
* fps：转场图片生成视频后的 fps（15 - 30）
* 返回值：
* 0：设置成功；
* -1：设置失败，请检查图片列表是否存在
*/
public int setPictureList(List<bitmap> bitmapList, int fps);

/*
* type：转场类型，详情见 TXVideoEditConstants
* 返回值：
* duration：转场视频时长（tips：同一个图片列表，每种转场动画的持续时间可能不一样，这里可以获取转场图片的持续时长）；
*/
public long setPictureTransition(int type)
```

- 其中，setPictureList 接口用于设置图片列表，最少设置三张，如果设置的图片过多，要注意图片的大小，防止内存占用过多而导致编辑异常。
- setPictureTransition 接口用于设置转场的效果，目前提供了6种转场效果供用户设置，每种转场效果持续的时长可能不一样，这里可以通过返回值获取转场的时长。
- 需要注意接口调用顺序，先调用 setPictureList，再调用 setPictureTransition。
- 图片编辑暂不支持的功能：重复、倒放、快速/慢速。其他视频相关的编辑功能，图片编辑均支持，调用方法和视频编辑完全一样。

版权所有：腾讯云计算（北京）有限责任公司

定制视频数据
定制视频数据(iOS)

最近更新时间：2021-07-26 16:48:55

录制预处理回调

```
/**
* 在 OpenGL 线程中回调，在这里可以进行采集图像的二次处理
* @param texture 纹理 ID
* @param width 纹理的宽度
* @param height 纹理的高度
* @return 返回给 SDK 的纹理
* 说明：SDK 回调出来的纹理类型是 GL_TEXTURE_2D，接口返回给 SDK 的纹理类型也必须是 GL_TEX
TURE_2D; 该回调在 SDK 美颜之后. 纹理格式为 GL_RGBA
*/
- (GLuint)onPreProcessTexture:(GLuint)texture width:(CGFloat)width height:(CGFloat)height;

/**
* 五官位置回调（企业版接口）
* @prama points 五官坐标
* 说明：使用了挂件的相关功能如动效贴纸、大眼或者瘦脸等。此回调在 onPreProcessTexture:width:h
eight: 之前会被调用
*/
- (void)onDetectFacePoints:(NSArray *)points;

/**
* 在 OpenGL 线程中回调，可以在这里释放创建的 OpenGL 资源
*/
- (void)onTextureDestoryed;
```

编辑预处理回调

```
/**
在 OpenGL 线程中回调，在这里可以进行采集图像的二次处理
@param texture 纹理 ID
@param width 纹理的宽度
```

```
@param height 纹理的高度
@param timestamp 纹理 timestamp 单位 ms
@return 返回给 SDK 的纹理
说明：SDK 回调出来的纹理类型是 GL_TEXTURE_2D，接口返回给 SDK 的纹理类型也必须是 GL_TEXT
URE_2D; 该回调在 SDK 美颜之后. 纹理格式为 GL_RGBA
timestamp 为当前视频帧的 pts ，单位是 ms ，客户可以根据自己的需求自定义滤镜特效
*/
- (GLuint)onPreProcessTexture:(GLuint)texture width:(CGFloat)width height:(CGFloat)height ti
mestamp:(UInt64)timestamp;

/**
* 在 OpenGL 线程中回调，可以在这里释放创建的 OpenGL 资源
*/
- (void)onTextureDestoryed;
```

定制视频数据(Android)

最近更新时间：2021-01-27 15:19:28

录制预处理回调

```
public interface VideoCustomProcessListener {
/**
* 在 OpenGL 线程中回调，在这里可以进行采集图像的二次处理
* @param textureId 纹理 ID
* @param width 纹理的宽度
* @param height 纹理的高度
* @return 返回给 SDK 的纹理 ID，如果不做任何处理，返回传入的纹理 ID 即可
* 说明：SDK 回调出来的纹理类型是 GLES20.GL_TEXTURE_2D，接口返回给 SDK 的纹理类型也必须是 GLES20.GL_TEXTURE_2D
*/
int onTextureCustomProcess(int textureId, int width, int height);

/**
* 增值版回调人脸坐标
* @param points 归一化人脸坐标，每两个值表示某点 P 的 X、Y 值。值域[0.f,1.f]
*/
void onDetectFacePoints(float[] points);

/**
* 在 OpenGL 线程中回调，可以在这里释放创建的 OpenGL 资源
*/
void onTextureDestroyed();
}
```

编辑预处理回调

```
public interface TXVideoCustomProcessListener {
/**
* 在 OpenGL 线程中回调，在这里可以进行采集图像的二次处理
*
* @param textureId 纹理 ID
```

```
 * @param width 纹理的宽度
 * @param height 纹理的高度
 * @return 返回给 SDK 的纹理 ID，如果不做任何处理，返回传入的纹理 ID 即可
 * <p>
 * 说明：SDK 回调出来的纹理类型是 GLES20.GL_TEXTURE_2D，接口返回给 SDK 的纹理类型也必须是 GLES20.GL_TEXTURE_2D
 */
int onTextureCustomProcess(int textureId, int width, int height, long timestamp);

/**
 * 在 OpenGL 线程中回调，可以在这里释放创建的 OpenGL 资源
 */
void onTextureDestroyed();
}
```

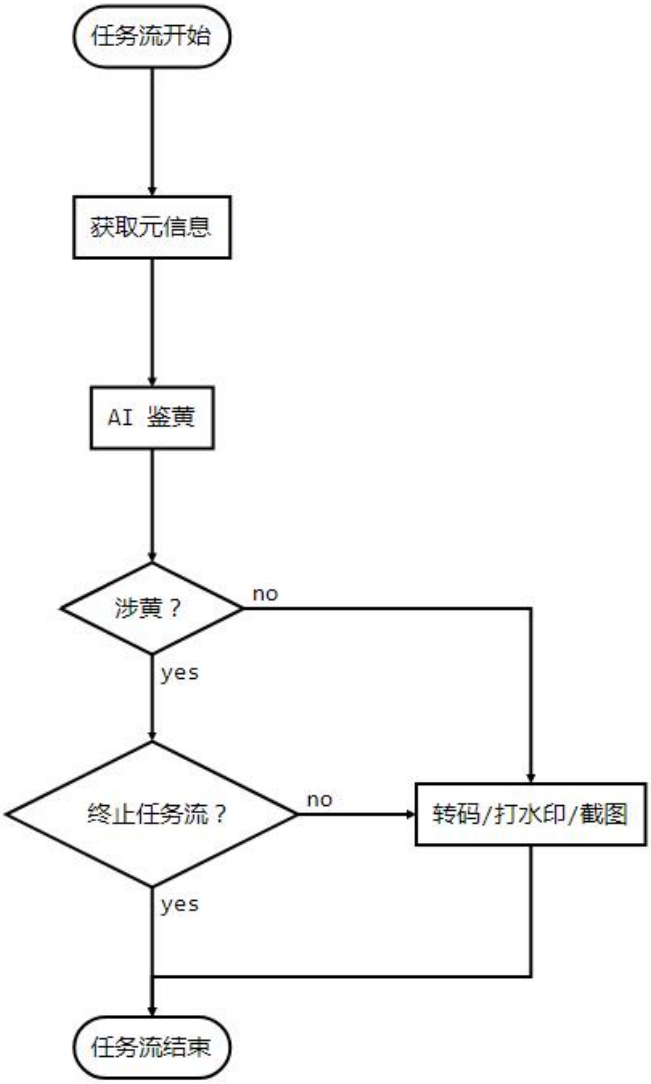
视频鉴黄

最近更新时间：2021-01-27 16:01:16

在个人直播录制、UGC 短视频等场景中，视频内容是不可预知的。为了避免一些违规内容出现在点播平台上，开发者会要求先对上传的视频做审核，确认合规后再进行转码和分发。腾讯云短视频解决方案支持对视频进行 AI 鉴黄，自动识别视频是否涉及色情内容。

使用 AI 鉴黄

AI 鉴黄功能需要集成在视频处理任务流中使用，它依赖于云点播后台的 视频 AI - 内容审核，审核结果通过事件通知的方式来通知 App 后台服务。云点播内置了一个任务流 QCVB_ProcessUGCFile 用于 UGC 短视频鉴黄场景，当用户使用该任务流并指定进行 AI 鉴黄时，鉴黄操作将优先执行，并根据鉴黄结果来决定是否进行后续处理（转码、打水印和截图等）。流程图如下：



AI 模版介绍

模版 ID	涉黄处理	采样频率
10	终止任务流（转码、打水印、截图等均不会执行）	时长小于500秒的视频，每1秒采样1次；时长大于或等于500秒的视频，每1%时长采样1次
20	继续执行任务流	无

AI 鉴定接入示例

步骤1：在生成上传签名时提交 AI 鉴黄任务

```
/**
* 响应签名请求并上传带有 AI 鉴黄任务
*/
function getUploadSignature(req, res) {
res.json({
code: 0,
message: 'ok',
data: {
//上传时指明模版参数与任务流
signature: gVodHelper.createFileUploadSignature({ procedure: 'QCVB_SimpleProcessFile({1,1,1,1})' })
}
});
}
```

步骤2：在获事件通知时获取 AI 鉴黄结果

```
/**
* 获取事件的 AI 鉴黄结果
*/
function getAiReviewResult(event){
let data = event.eventContent.data;
if(data.aiReview){
return data.aiReview;
}
return {};
}
```

版权所有：腾讯云计算（北京）有限责任公司