3D渲染插件MetaKit 集成Doc --Android ver2.2

准备环境

在开始集成MetaKit前，请确保开发环境满足以下要求：

- android studio4.2及以上版本。
- Android5.1及以上的系统版本(建议使用2019年及之后生产的机型)。
- 设备的前置摄像头与麦克风功能正常。

功能列表

模块	功能
虚拟人Avatar	avatar切换
	avatar面捕驱动
	avata捏脸
	avatar换妆
	avater视角切换
Animoji特效	Animoji切换
	Animoji面捕驱动
挂件贴纸sticker	眼镜、面纱、口罩、帽子
背景特效	2D背景替换
	全景背景替换
	3D场景背景替换
灯光特效	3D打光
	氛围灯
	广告灯
律动特效	屏幕波纹
	极光特效
	人像边缘火焰
	人像边缘光线

注：Agora提供美术资源的设计及制定规范，如有自定义美术资源的需求请联系技术支持

集成SDK

1. API Example

metakit依赖于agora rtc sdk，本集成方法需在具备基本的agora rtc能力的基础APK上方能生效，
请参照跑通API Example项目https://doc.shengwang.cn/doc/rtc/android/get-started/run-demo
完成基本的项目后再做集成。

2. 集成AgoraSDK 4.3.2.3及以后版本

下载地址：https://download.agora.io/sdk/release/Agora_Native_SDK_for_Android_rel.v4.3.2.3_63563_FULL_20240808_2121_326446.zip

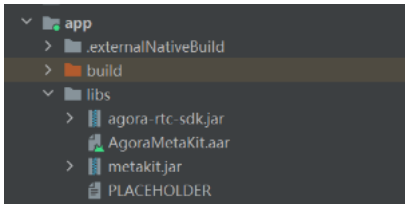
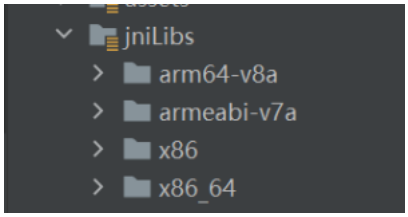
3. 集成 Agora插件

各个功能所需的agora插件(背景分割及面捕插件包含在AgoraSDK full包里)

功能模块	所需插件

虚拟人	metakit 插件： AgoraMetakit.aar metakit.jar libagora_metakit_extension.so 面捕驱动插件： libagora_face_capture_extension.so
Animoji特效 Sticker挂件贴纸	metakit 插件： AgoraMetakit.aar metakit.jar libagora_metakit_extension.so 面捕驱动插件： libagora_face_capture_extension.so
背景特效	metakit 插件： AgoraMetakit.aar metakit.jar libagora_metakit_extension.so 背景分割插件： libagora_segmentation_extension.so
灯光律动特效	metakit 插件： AgoraMetakit.aar metakit.jar libagora_metakit_extension.so 背景分割插件： libagora_segmentation_extension.so

(3.1). metakit相关的库及集成位置

库	功能	集成位置示例
AgoraMetaKit.aar	metakit渲染runtime层	
metakit.jar	metakit wrapper层 java包	
libagora_metakit_extension.so	metakit wrapper层 so	集成时置于jniLibs\arm64-v8a 或jniLibs\armeabi-v7a 

(3.2). 添加aar/jar依赖引用

进入app目录 ，在在build.gradle中，在dependencies节点添加对libs/下aar及jar包的引用

```
implementation fileTree(dir: 'libs', include: ['*.jar', '*.aar'])
```

(3.3). 使用metakit集成到app中所需权限

权限	功能	在metakit中的用途

CAMERA	访问手机相机	metakit中表情驱动/背景分割等功能需要访问摄像头以做AI推理
INTERNET	访问网络	AI模块授权所需
WRITE_EXTERNAL_STORAGE	访问外部存储	记录SDK相关log文件所必需

需要注意的是，android6.0及以上的版本，对权限管理更加严格，不能仅仅在AndroidManifest.xml中进行静态申请，需要执行类似如下代码，进行动态

申请

```
private static final String[] REQUESTED_PERMISSIONS = {
        Manifest.permission.RECORD_AUDIO,
        Manifest.permission.CAMERA,
        Manifest.permission.WRITE_EXTERNAL_STORAGE,
        Manifest.permission.READ_EXTERNAL_STORAGE,
        Manifest.permission.READ_PHONE_STATE
};
```

```
private boolean checkSelfPermission_confirm(String permission, int requestCode) {
    if (ContextCompat.checkSelfPermission( context: this, permission) !=
            PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions( activity: this, REQUESTED_PERMISSIONS, requestCode);
        return false;
    }
    return true;
}

1 usage
private boolean checkSelfPermissions() {
    boolean a = checkSelfPermission_confirm(REQUESTED_PERMISSIONS[0], PERMISSION_REQ_ID);
    boolean b = checkSelfPermission_confirm(REQUESTED_PERMISSIONS[1], requestCode: PERMISSION_REQ_ID + 1);
    boolean c = checkSelfPermission_confirm(REQUESTED_PERMISSIONS[2], requestCode: PERMISSION_REQ_ID + 2);
    boolean d = checkSelfPermission_confirm(REQUESTED_PERMISSIONS[3], requestCode: PERMISSION_REQ_ID + 3);
    boolean e = checkSelfPermission_confirm(REQUESTED_PERMISSIONS[4], requestCode: PERMISSION_REQ_ID + 4);

    if(BuildConfig.VIDEO_ENABLED){
        return a&&b&&c&&d&&e;
    }else{
        return a&&c&&d&&e;
    }
}
```

(3.4). CPU架构选择

目前的metakit支持arm64-v8a及armeabi-v7a，考虑到apk体积，在集成时，选择所需的架构即可

```
ndk{
    abiFilters "arm64-v8a"
    }
```

资源

包含场景及各个功能所需的bundle文件，在assets目录下

资源类型	必须与否	用途	size
material_avatar_xxx	根据功能选择	虚拟人Avatar的分包资源。包含虚拟人形像"girl", "huamulan"。可使用面捕驱动，捏脸，换装等能力，集成时需要将该资源的绝对路径设置给MetaKit。	• 虚拟人"girl"(支持捏脸换装) • 虚拟人"boy" (仅支持捏脸) • 虚拟人"huamulan"(不支持捏脸换装)
material_package_xxx	根据功能选择	虚拟人Avatar的套装资源，包含虚拟人形像"girl"的3套衣服。	• 虚拟人"girl"套装1 • 虚拟人"girl"套装2 • 虚拟人"girl"套装3
material_animoji_xxx	根据功能选择	遮脸头套Animoji的分包资源。包含头套"dog"，"girlhead"。可用面捕驱动能力，集成时需要将该资源的绝对路径设置给MetaKit。	• 头套"dog" • 头套"girlhead" • 头套"arkit"
material_sticker_xxx	根据功能选择	挂件贴纸Sticker的分包资源。	• 挂件"眼镜" • 挂件"口罩" • 挂件"面纱" • 挂件"龙头帽"
material_bgeffect_xxx	根据功能选择	背景特效的分包资源。	• "全景图"背景 • "3D房间"背景
material_effect_xxx	根据功能选择	特效Effect的分包资源。包含“3D打光“，”氛围灯“，”广告灯“，”火焰律动“，”光线律动“，”极光“，”屏幕波纹“，集成时需要将该资源的绝对路径设置给MetaKit	• 特效"3D打光" • 特效"氛围灯" • 特效"广告灯" • 特效"火焰律动" • 特效"光线律动" • 特效"极光" • 特效"屏幕波纹"
捏脸换装部位信息.txt	Avatar捏脸换装的信息		
素材.md	素材包的对照目录		

关键步骤代码example

1.创建rtcEngine，并注册extension回调监听事件接口

```
mRtcConfig.mContext = mContext;
mRtcConfig.mAppId = appId;
mRtcConfig.mEventHandler = mEngineEventHandler.mRtcEventHandler;
mRtcConfig.mExtensionObserver = mEngineEventHandler.mExtEventHandler;
mRtcConfig.mChannelProfile = mEngineConfig.mChannelProfile;
mRtcEngine = RtcEngine.create(mRtcConfig);
new observer implemets IMediaExtensionObserver {

void onEvent(String provider, String extension, String key, String value);

void onStarted(String provider, String extension);
void onStopped(String provider, String extension);
void onError(String provider, String extension, int error, String message);
```

2. 注册metakit插件及其他所需插件

```
(a). +metakit
mRtcEngine.registerExtension("agora_video_filters_face_capture",
"face_capture", Constants.MediaSourceType.PRIMARY_CAMERA_SOURCE);
mRtcEngine.registerExtension("agora_video_filters_metakit", "metakit",
Constants.MediaSourceType.PRIMARY_CAMERA_SOURCE);
mRtcEngine.enableExtension("agora_video_filters_face_capture",
"face_capture", true, Constants.MediaSourceType.PRIMARY_CAMERA_SOURCE);
mRtcEngine.enableExtension("agora_video_filters_metakit", "metakit", true,
Constants.MediaSourceType.PRIMARY_CAMERA_SOURCE);


// (license)
mRtcEngine.setExtensionProperty("agora_video_filters_face_capture","
face_capture", "authentication_information",
 "{\"company_id\":\"agoraDemo\"," +
                "\"license\":\"" +

"PFeVRXMZ1hJd075NSZtrZD3tYYl154QTs1Ui2i5ztvOcgOhklZcZdl2f5dYmf6GuLUebbBs1xp
5I" +

"crywJ+NaZ8ncYAfekfg3oR34cYJ8Pe3z4EVzw+CshGOfL0hcSYmtXjkXxG298c64+SGZdP6
/UBVJ" +
                "/wIOYJRMgEUTBTqewcQ=\"}", Constants.MediaSourceType.
PRIMARY_CAMERA_SOURCE);

(b). +metakit
VirtualBackgroundSource source = new VirtualBackgroundSource();
source.backgroundSourceType = 0; //0 for background none; 1 for background
color
source.color = 0xFFFFFF;//0xffffff;
source.source = "";
source.blurDegree = 1;
SegmentationProperty param = new SegmentationProperty();
param.modelType = 1; //1 for AI, 2 for green screen
param.greenCapacity = 0.5f;
mRtcEngine.enableVirtualBackground(true, source, param);

mRtcEngine.registerExtension("agora_video_filters_metakit", "metakit",
Constants.MediaSourceType.PRIMARY_CAMERA_SOURCE);
mRtcEngine.enableExtension("agora_video_filters_metakit", "metakit", true,
Constants.MediaSourceType.PRIMARY_CAMERA_SOURCE);
```

3. 设置activity及启动metakit渲染引擎

注： android metakit需要一个activity 以启动渲染引擎

```
Activity mActivity
JSONObject valueObj = new JSONObject();
try {
    long address = getContextHandler(mActivity);
         valueObj.put("activityContext", String.valueOf(address));
} catch (JSONException e) {
    e.printStackTrace();
}
mRtcEngine.setExtensionProperty("agora_video_filters_metakit", "metakit",
"setActivityContext", valueObj.toString());
addressjniobject global ref

JNIEXPORT jlong JNICALL Java_xxx_getContextHandler
(JNIEnv *env, jobject obj, jobject context)
{
    m_contextRef = env->NewGlobalRef(context);
         jlong handler = reinterpret_cast<jlong>(m_contextRef);
         return handler;
}
mRtcEngine.setExtensionProperty("agora_video_filters_metakit", "metakit",
"initialize","{}");
```

4. 销毁metakit引擎

```
mRtcEngine.setExtensionProperty("agora_video_filters_metakit", "metakit",
"destroy", "{}");
```

5. 加载场景资源

注：加载资源可以反复执行切换，不需要load与unload配对：loadMaterial(pathA)->loadMaterial(pathB)->loadMaterial(pathC)→ ...→ unloadMaterial()

场景加载默认使用纹理接口，当前引擎仅支持一张渲染纹理

```
// pathunity
JSONObject valueObj = new JSONObject();
try {
         valueObj.put("path", path);
         valueObj.put("view", "0"); // "0"
} catch (JSONException e) {
    e.printStackTrace();
}
mRtcEngine.setExtensionProperty("agora_video_filters_metakit", "metakit",
"loadMaterial", valueObj.toString());
```

6. 卸载场景资源

注：卸载场景资源后，只能进行销毁引擎，不可再loadMaterial

```
JSONObject valueObj = new JSONObject();
try {
          valueObj.put("path", path);
} catch (JSONException e) {
    e.printStackTrace();
}
mRtcEngine.setExtensionProperty("agora_video_filters_metakit", "metakit",
"unloadMaterial", "{}");
```

7. 开启/关闭某个纹理输出

可以开启/关闭其纹理输出功能，输出的纹理可用于上屏或编码传输，加入频道的观众可看到推流端的画面

```
long addressHanlde = 0; // "0"

boolean onoff = true;

JSONObject valueObj = new JSONObject();
try {
    valueObj.put("view", String.valueOf(addressHandle));
         valueObj.put("enable", onoff);
} catch (JSONException e) {
    e.printStackTrace();
}
int ret = mRtcEngine.setExtensionProperty("agora_video_filters_metakit","
metakit", "enableSceneVideo", valueObj.toString());
```

8. 在纹理上渲染avatar，资源名“material_avatar_xx”

捏脸及换装对应的各个模块参数对应的详细细节见附件捏脸换装部位信息.txt

```
long addressHanlde = 0;
JSONObject valueObj = new JSONObject();
try {
    valueObj.put("view", String.valueOf(addressHandle));
         valueObj.put("path", path_to_avatar);
} catch (JSONException e) {
    e.printStackTrace();
}
mRtcEngine.setExtensionProperty("agora_video_filters_metakit", "metakit",
"loadMaterial", valueObj.toString());


// avatar
// avatar material_package_dress_xx,(loadMaterial)


// avatar
// avatar material_package_faceshape_xx, (loadMaterial)
```

9. 在纹理上渲染遮脸头套animoji/挂件sticker

animoji资源(material_animoji_xx) sticker资源(material_sticker_xx)

```
long addressHanlde = 0;

JSONObject valueObj = new JSONObject();
try {
    valueObj.put("view", String.valueOf(addressHandle));
         valueObj.put("path", path_to_animoji);
} catch (JSONException e) {
    e.printStackTrace();
}
mRtcEngine.setExtensionProperty("agora_video_filters_metakit", "metakit",
"loadMaterial", valueObj.toString());
```

10. 在纹理上开启背景替换功能，资源名“material_bgeffect_xx”

```
long addressHanlde = 0;

JSONObject valueObj = new JSONObject();
try {
    valueObj.put("view", String.valueOf(addressHandle));
         valueObj.put("path", path_to_bgeffect);
} catch (JSONException e) {
    e.printStackTrace();
}
mRtcEngine.setExtensionProperty("agora_video_filters_metakit", "metakit",
"loadMaterial", valueObj.toString());
```

11. 在纹理上开启特效功能，资源名“material_effect_xx”

参数名称解释：

参数	值类型	解释
color	int64	普通颜色
startColor	int64	渐变颜色-起始
endColor	int64	渐变颜色-结束
intensity	float	强度
scale	float	缩放因子
speed	float	速度
size	float	尺寸
range	float	场景扫描范围
text	string	文本内容
animation	int	动画类型

setEffectVideo						
id	name	str	param	type	建议值	默认值
1001	3D打光	3dLight	color # intensity # scale	int64 # float # float	\ # 1.0~2.0 # 0.3~0.6	\ # 1.6 # 0.4
1002	屏幕波纹	screenRipple	color # speed # scale	int64 # float # float	\ # -0.2~0.2 # 3.0~6.0	\ # -0.12 # 4.0
1003	极光	aurora	color # intensity	int64 # float	\ # 0.8~1.5	\ # 1.0
2001	紫色火焰	purpleFlame	color # intensity	int64 # float	\ # 0.2~1.5	\ # 0.2
3001	氛围灯光组	lightGroup	quality	int	0/1/2	0
3002	广告灯	advertisinglight	startColor # endColor # size # intensity # range	int64 # int64 # float # float # float	\ # \ # 8~15 # 100~1000 # 10-40	\ # \ # 10 # 1000 # \15

```
long addressHanlde = 0;

JSONObject valueObj = new JSONObject();
try {
    valueObj.put("view", String.valueOf(addressHandle));
         valueObj.put("path", path_to_effect);
} catch (JSONException e) {
    e.printStackTrace();
}
mRtcEngine.setExtensionProperty("agora_video_filters_metakit", "metakit",
"loadMaterial", valueObj.toString());


// 1001“3D”
JSONObject valueObj = new JSONObject();
JSONObject paramObj = new JSONObject();
try {
        paramObj.put("intensity", 2.0);
        paramObj.put("scale", 0.3);
    valueObj.put("id", 1001);
        valueObj.put("enable", true);
         valueObj.put("param", paramObj);
} catch (JSONException e) {
    e.printStackTrace();
}
mRtcEngine.setExtensionProperty("agora_video_filters_metakit", "metakit",
"setEffectVideo", valueObj.toString());
```

12. 设置渲染profile

针对性能不同的机型，可以选择低中高三种渲染策略以达到渲染效果和性能的匹配

```
int level = 0;  // 0/1/2//
JSONObject valueObj = new JSONObject();
try {
    valueObj.put("general", level);
} catch (JSONException e) {
    e.printStackTrace();
}
mRtcEngine.setExtensionProperty("agora_video_filters_metakit", "metakit",
"setRenderQuality", valueObj.toString());
```

13. metakit引擎信息监听同步

在创建rtc engine的时候，注册的extension observer会监听到来自插件的消息回调，通过区分provider/extension，可以过滤到来自metakit的同步状态信息回调

```
public void onEvent(String provider, String ext, String key, String msg) {
    if (!provider.equals("agora_video_filters_metakit") || !ext.equals
("metakit")) return;
         switch(key) {
    case "initializeFinish":        //initializeunity
         break;
         case "unityLoadFinish":         //unity
         break;
        case "materialLoaded":          //response
    break;
        case "materialUnLoaded":        //response,
         break;
         }
}

@Override
public void onError(String provider, String ext, int key, String msg) {
    if (!provider.equals("agora_video_filters_metakit") || !ext.equals
("metakit")) return;
        Log.i("[MetaKit]", "onError: " + key + ", msg: " + msg);
}

@Override
public void onStart(String provider, String ext) {
    if (!provider.equals("agora_video_filters_metakit") || !ext.equals
("metakit")) return;
         Log.i("[MetaKit]", "onStart");
}

@Override
public void onStop(String provider, String ext) {
    if (!provider.equals("agora_video_filters_metakit") || !ext.equals
("metakit")) return;
         Log.i("[MetaKit]", "onStop");
}
```

14. 关闭纹理请求功能

目前只支持index 0

```
int index = 0;
JSONObject valueObj = new JSONObject();
try {
        valueObj.put("index", index);
        valueObj.put("enable", false);
} catch (JSONException e) {
        e.printStackTrace();
}
mRtcEngine.setExtensionProperty("agora_video_filters_metakit", "metakit",
"requestTexture", valueObj.toString());
```

注意事项

1. metakit库跨SDK大版本（4.2.x，4.3.x，4.4.x）是无法保证兼容的，请依据自己集成的SDK版本获取对应的library，以免出现稳定性问题。
2. 资源如果是走下载解压的方式，请业务层开发自己保证下载解压后资源的完备性。如果目录不对，或有资源bundle丢失，加载时会出现异常。
3. 安卓需要把activity的context共享到metakit引擎层，且是C++指针的形式，因此需要业务层做好jni转换，切记不是java地址。
4. 请确保在收到确切的引擎状态后，执行对应的操作，以免出现不必要的时序问题：
    1. 在收到unityLoadFinish后表示引擎启动完成，可以触发资源加载loadMaterial。
    2. 在收到materialLoaded后进行一些状态切换（如特效参数设置），行为变更（如捏脸换装）。
    3. 在收到materialUnLoaded后仅可执行引擎销毁，不可重复进行loadMaterial操作。
    4. 如果页面有进行metakit initialize，在退出页面前确保有发送destroy，保证引擎的正常生命周期，减少性能和内存的影响。