Live Sticker & Filter SDK

Developer Guide(Android)

Mobile & AI Leader

1. Android Development Overview

1) Project Configuration Overview

- SDK file: Refer to the sample program (LiveSticker) for project import related matters.

face.aar (app / src / main / libs)
model.bf2 (app / src / main / assets)
live_sticker.json (app / src / main / assets)

- Other files: Files for the UI implementation and other processing in the sample program

(Camera preview and other rendering processing)

MainActivity.java, CameraSurfaceView.java, Renderer.java
Data. Java, Fil; terAdaptor.java, ItemAdaptor.java

(Processing related to video recording)

VideoRecorder.java, MediaEncoder.java, MediaVideoEncoder.java, MediaAudiEncoder.java, MediaMuxerWapper.java
EglCore.java, EglSurfaceBase.java, WindowSurface.java

(Replay of the recorded video)

VideoActivity.java

2. Initialization of the SDK

1) Creation of an instance

Instance can be created statically using a singleton

```
JFaceLib jFaceLib = JFaceLib.getInstance();
```

2) Initialization of the SDK

a. ResourceInfo initialization

(org.junsoft.facelibandroid.ResourceInfo)

See the initResource part in the MainActivity.java source

(ResourceInfo structure)

```
public class ResourceInfo {
        public String sceneName;
        public String filterName;
        public String targetPath;
        public int type;

        public ResourceInfo() {
        }
}
```

scenName :"LiveSticker" for the Live Sticker, "Filter" for a filter

filterName : For the Live Sticker, it corrsponds to the name for live_sticker.json, for a filter, to the lookup file name

targetPath : Required only if the resource type is of the network type

type : As for local resources, an int value of the resource type according to the network resource type

```
    (ex)    eResouceType.eNetworkResorce
            this.info.type = eResType.ordinal();
```

b. Initialization of the SDK: Initializing the SDK based on the resource information structure

```
jFaceLib.initLibrary(getApplicationContext(),this.info);
```

3. Camera Initialization

The sample project contains sample camera initialization information. In MainActivity.java, the processing portion of the end of the openCamera function is required. It takes arguments to the function that must be executed in the Renderer

Ex)

(MainActivity.java)

```
mGLSurfaceView.queueEvent(newRunnable() {
        @Override
        public void run() {
                mRenderer.openJFaceSession(previewWidth,
                                previewHeight,
                                horizontalFov,
                                detectionFps,
                                info.orientation,
                                isFrontFacing,
                                fps);

        }
});
```

(Renderer.java)

```
public void openJFaceSession(
        int previewWidth, int previewHeight, float horizontalFov, float detectionFps,
        int orientation, boolean isFrontFacing, float fps)
{
        mIsFrontFacing = isFrontFacing;
        mCameraOrientation = orientation;
        jFaceLib.openJFaceSession(previewWidth,
                        previewHeight,
                        horizontalFov,
                        detectionFps,
                        orientation,
                        isFrontFacing,
                        fps);
}
```

프레임

4. Frame Buffer Processing

In the sample project, camera callbacks are implemented in MainActivity.java.

onPreviewFrame calls the processVideoFrame implemented in the Renderer

(MainActivity.java)

```
public void onPreviewFrame(final byte[] bytes, final Camera camera) {


    mGLSurfaceView.queueEvent(new Runnable() {
        @Override
        public void run() {
…
            mRenderer.processVideoFrame(bytes);
            //processVideoFrame(byte[] videoFrame, int orientation)
            camera.addCallbackBuffer(bytes);

        }

    });

}
```

(Renderer.java)

```
public int processVideoFrame(byte[] videoFrame) {

    if (!mFrameRendered) return -1;

    int ret = jFaceLib.processVideoFrame(videoFrame);
    mFrameRendered = false;
    mSurfaceView.requestRender();

    return ret;

}
```

5. Rendering Processing

Renderer.java calls renderScene from onDrawFrame to render the selected filter or the Live Sticker

Please also refer to the video recording related codes

```
@Override
public void onDrawFrame(GL10 gl10) {


    int orientation = ((mCameraOrientation == 270) ^ (mIsFrontFacing))
            ? POSITION_MINUS_Y_UP : POSITION_PLUS_Y_UP;

    int mirror = (mIsFrontFacing ^ (orientation == POSITION_MINUS_Y_UP))
            ? MIRRORED_P_TO_Q : DEFAULT_P_TO_P;


    runAll(mRunOnDraw);

    if(videoRecorder.isRecording())
        videoRecorder.makeCurrent();

    Scene.renderScene(mSurfaceWidth, mSurfaceHeight, orientation, mirror);

    if(videoRecorder.isRecording())
    {
        videoRecorder.swapBuffers();
        videoRecorder.render();

        Scene.renderScene(mSurfaceWidth, mSurfaceHeight, orientation, mirror);
    }

    mFrameRendered = true;

    runAll(mRunOnDrawEnd);

}
```

6. Live Sticker / Filter Selection

Fill the ResourceInfo with the appropriate information and then call selectScene.

For more information, see the onItemClick (live sticker selection) and onItemClick2 (filter selection) sections in MainActivity.java in the sample project

```
public void setFilter(final ResourceInfo _info)
{
        mGLSurfaceView.queueEvent(new Runnable() {
                @Override
                public void run() {
                        mRenderer.selectScene(_info);

                }
        });

}
```