[Android audio and video development and upgrade: OpenGL rendering video screen] 4. In-depth understanding of OpenGL's EGL

[Android audio and video development and upgrade: OpenGL rendering video screen articles] Fourth, in-depth understanding of OpenGL's EGL

Table of contents

In this article you can learn

> As an intermediate bridge between OpenGL and local window rendering, EGL is often ignored by developers who are just getting started with OpenGL. With the deepening of learning, EGL will be something that has to be faced. This article will introduce what EGL is, what it is useful for, and how to use EGL.

1. What is EGL

As an Android developer, EGL seems to be a very strange thing, why?

~~It's all because Android's GLSurfaceView is so well packaged. Hahaha~~~

1. Why does onDrawFrame keep calling back?

As mentioned in the previous article, OpenGL is based on threads. So far, we have not deeply realized this problem, but what we know is that when we inherit GLSurfaceView.Renderer, the system will call back the following methods:

```
override fun onSurfaceCreated(gl: GL10?, config: EGLConfig?) {
}

override fun onSurfaceChanged(gl: GL10?, width: Int, height: Int) {
}

override fun onDrawFrame(gl: GL10?) {
}
```

And the onDrawerFrame method will be called continuously, and we implement the OpenGL drawing process in it.

Here we can guess, is it possible that the thread that can be called continuously is a while loop thread?

The answer is: Yes.

If you look at the source code of GLSurfaceView, you will find a thread called GLThread where EGL related content is initialized. And at the right time, the three methods in the Renderer are called respectively.

So, what exactly is EGL?

2. What is EGL?

We know that OpenGL is a set of APIs that can operate the GPU, but it can only operate the GPU and cannot render images to the display window of the device. Then, an intermediate layer is needed, connecting OpenGL to the device window, and preferably cross-platform.

And so EGL came along, a set of platform-agnostic APIs provided by the Khronos Group.

3. Some basic knowledge of EGL

EGLDisplay

An abstract system display class defined by EGL for operating device windows.

EGLConfig

EGL configuration like rgba bits

EGLSurface

Rendering cache, a memory space, all image data to be rendered to the screen must be cached on EGLSurface first.

EGLContext

OpenGL context, used to store OpenGL drawing state information and data.

> The process of initializing EGL is actually the process of configuring the above information.

2. How to use EGL

Just looking at the above introduction, it is actually quite difficult to understand what EGL does, or how to use EGL.

Please think about a question first

> If there are two GLSurfaceViews rendering video images at the same time, why can OpenGL correctly draw the images into two GLSurfaceViews?

Carefully recall each API of OpenGL ES, is there any API that specifies which GLSurfaceView the current screen is rendered to?

No!

Please read the following with this question in mind.

1. Encapsulate EGL core API

First, encapsulate the content of the EGL initialization core (the 4 introduced in the first section) and name it **EGLCore**

```
const val FLAG_RECORDABLE = 0x01

private const val EGL_RECORDABLE_ANDROID = 0x3142

class EGLCore {

    private val TAG = "EGLCore"

    // EGL相关变量
    private var mEGLDisplay: EGLDisplay = EGL14.EGL_NO_DISPLAY
    private var mEGLContext = EGL14.EGL_NO_CONTEXT
    private var mEGLConfig: EGLConfig? = null

    /**
     * 初始化EGLDisplay
     * @param eglContext 共享上下文
     */
    fun init(eglContext: EGLContext?, flags: Int) {
        if (mEGLDisplay !== EGL14.EGL_NO_DISPLAY) {
            throw RuntimeException("EGL already set up")
        }

        val sharedContext = eglContext ?: EGL14.EGL_NO_CONTEXT

        // 1，创建 EGLDisplay
        mEGLDisplay = EGL14.eglGetDisplay(EGL14.EGL_DEFAULT_DISPLAY)
        if (mEGLDisplay === EGL14.EGL_NO_DISPLAY) {
            throw RuntimeException("Unable to get EGL14 display")
        }

        // 2，初始化 EGLDisplay
        val version = IntArray(2)
        if (!EGL14.eglInitialize(mEGLDisplay, version, 0, version, 1)) {
            mEGLDisplay = EGL14.EGL_NO_DISPLAY
            throw RuntimeException("unable to initialize EGL14")
        }

        // 3，初始化EGLConfig，EGLContext上下文
        if (mEGLContext === EGL14.EGL_NO_CONTEXT) {
            val config = getConfig(flags, 2) ?: throw RuntimeException("Unable to
find a suitable EGLConfig")
            val attr2List = intArrayOf(EGL14.EGL_CONTEXT_CLIENT_VERSION, 2,
EGL14.EGL_NONE)
            val context = EGL14.eglCreateContext(
                mEGLDisplay, config, sharedContext,
                attr2List, 0
            )
            mEGLConfig = config
            mEGLContext = context
        }
    }

    /**
     * 获取EGL配置信息
     * @param flags 初始化标记
     * @param version EGL版本
```

```
 */
private fun getConfig(flags: Int, version: Int): EGLConfig? {
    var renderableType = EGL14.EGL_OPENGL_ES2_BIT
    if (version >= 3) {
        // 配置EGL 3
        renderableType = renderableType or EGLExt.EGL_OPENGL_ES3_BIT_KHR
    }

    // 配置数组，主要是配置RAGA位数和深度位数
    // 两个为一对，前面是key，后面是value
    // 数组必须以EGL14.EGL_NONE结尾
    val attrList = intArrayOf(
        EGL14.EGL_RED_SIZE, 8,
        EGL14.EGL_GREEN_SIZE, 8,
        EGL14.EGL_BLUE_SIZE, 8,
        EGL14.EGL_ALPHA_SIZE, 8,
        //EGL14.EGL_DEPTH_SIZE, 16,
        //EGL14.EGL_STENCIL_SIZE, 8,
        EGL14.EGL_RENDERABLE_TYPE, renderableType,
        EGL14.EGL_NONE, 0, // placeholder for recordable [@-3]
        EGL14.EGL_NONE
    )
    //配置Android指定的标记
    if (flags and FLAG_RECORDABLE != 0) {
        attrList[attrList.size - 3] = EGL_RECORDABLE_ANDROID
        attrList[attrList.size - 2] = 1
    }
    val configs = arrayOfNulls<EGLConfig>(1)
    val numConfigs = IntArray(1)

    //获取可用的EGL配置列表
    if (!EGL14.eglChooseConfig(mEGLDisplay, attrList, 0,
            configs, 0, configs.size,
            numConfigs, 0)) {
        Log.w(TAG, "Unable to find RGB8888 / $version EGLConfig")
        return null
    }

    //使用系统推荐的第一个配置
    return configs[0]
}

/**
 * 创建可显示的渲染缓存
 * @param surface 渲染窗口的surface
 */
fun createWindowSurface(surface: Any): EGLSurface {
    if (surface !is Surface && surface !is SurfaceTexture) {
        throw RuntimeException("Invalid surface: $surface")
    }

    val surfaceAttr = intArrayOf(EGL14.EGL_NONE)

    val eglSurface = EGL14.eglCreateWindowSurface(
                                    mEGLDisplay, mEGLConfig, surface,
                                    surfaceAttr, 0)
```

```
        if (eglSurface == null) {
            throw RuntimeException("Surface was null")
        }

        return eglSurface
    }

    /**
     * 创建离屏渲染缓存
     * @param width 缓存窗口宽
     * @param height 缓存窗口高
     */
    fun createOffscreenSurface(width: Int, height: Int): EGLSurface {
        val surfaceAttr = intArrayOf(EGL14.EGL_WIDTH, width,
                                                EGL14.EGL_HEIGHT, height,
                                                EGL14.EGL_NONE)

        val eglSurface = EGL14.eglCreatePbufferSurface(
                                        mEGLDisplay, mEGLConfig,
                                        surfaceAttr, 0)

        if (eglSurface == null) {
            throw RuntimeException("Surface was null")
        }

        return eglSurface
    }

    /**
     * 将当前线程与上下文进行绑定
     */
    fun makeCurrent(eglSurface: EGLSurface) {
        if (mEGLDisplay === EGL14.EGL_NO_DISPLAY) {
            throw RuntimeException("EGLDisplay is null, call init first")
        }
        if (!EGL14.eglMakeCurrent(mEGLDisplay, eglSurface, eglSurface,
mEGLContext)) {
            throw RuntimeException("makeCurrent(eglSurface) failed")
        }
    }

    /**
     * 将当前线程与上下文进行绑定
     */
    fun makeCurrent(drawSurface: EGLSurface, readSurface: EGLSurface) {
        if (mEGLDisplay === EGL14.EGL_NO_DISPLAY) {
            throw RuntimeException("EGLDisplay is null, call init first")
        }
        if (!EGL14.eglMakeCurrent(mEGLDisplay, drawSurface, readSurface,
mEGLContext)) {
            throw RuntimeException("eglMakeCurrent(draw,read) failed")
        }
    }

    /**
```

```
     * 将缓存图像数据发送到设备进行显示
     */
    fun swapBuffers(eglSurface: EGLSurface): Boolean {
        return EGL14.eglSwapBuffers(mEGLDisplay, eglSurface)
    }

    /**
     * 设置当前帧的时间，单位：纳秒
     */
    fun setPresentationTime(eglSurface: EGLSurface, nsecs: Long) {
        EGLExt.eglPresentationTimeANDROID(mEGLDisplay, eglSurface, nsecs)
    }

    /**
     * 销毁EGLSurface，并解除上下文绑定
     */
    fun destroySurface(elg_surface: EGLSurface) {
        EGL14.eglMakeCurrent(
            mEGLDisplay, EGL14.EGL_NO_SURFACE, EGL14.EGL_NO_SURFACE,
            EGL14.EGL_NO_CONTEXT
        )
        EGL14.eglDestroySurface(mEGLDisplay, elg_surface);
    }

    /**
     * 释放资源
     */
    fun release() {
        if (mEGLDisplay !== EGL14.EGL_NO_DISPLAY) {
            // Android is unusual in that it uses a reference-counted EGLDisplay.
So for
            // every eglInitialize() we need an eglTerminate().
            EGL14.eglMakeCurrent(
                mEGLDisplay, EGL14.EGL_NO_SURFACE, EGL14.EGL_NO_SURFACE,
                EGL14.EGL_NO_CONTEXT
            )
            EGL14.eglDestroyContext(mEGLDisplay, mEGLContext)
            EGL14.eglReleaseThread()
            EGL14.eglTerminate(mEGLDisplay)
        }

        mEGLDisplay = EGL14.EGL_NO_DISPLAY
        mEGLContext = EGL14.EGL_NO_CONTEXT
        mEGLConfig = null
    }
}
```

The above is the most basic and concise EGL initialization package, basically every method is necessary.

Specifically look at:

Initializing init is divided into 3 steps:

- Create EGLDisplay via eglGetDisplay
- Initialized EGLDisplay via eglInitialize
- Initialize EGLContext via eglCreateContext

Among them, the getConfig method is called when EGLCongtext is initialized.

Configuration context getConfig:

- According to the selected EGL version, configure the version flag
- Initialize the configuration list, configure the number of rgba bits and depth bits for rendering, two are a pair, the former is the type, the latter is the value, and must end with EGL14.EGL_NONE.
- Configure the Android-specific property **EGL_RECORDABLE_ANDROID** .
- According to the above configuration information, through eglChooseConfig, the system will return a list of matching configuration information. Generally, the first configuration information is returned.

> **The Android-specified flag EGL_RECORDABLE_ANDROID**
> tells EGL that the surface it creates must be compatible with the video codec.
> Without this flag, EGL may use a Buffer that MediaCodec does not understand.
> This variable comes with the system after api26. For compatibility, we write the value 0x3142 ourselves.

Create EGLSurface, divided into two modes:

- Displayable window, created with eglCreateWindowSurface.
- Offscreen (invisible) window, created with eglCreatePbufferSurface.

The first is the most commonly used, usually the Surface or SurfaceTexture held by the SurfaceView on the page is passed in for binding. In this way, the image data processed by OpenGL can be displayed on the screen.

The second is used for off-screen rendering, that is, the image data processed by OpenGL is stored in the cache and will not be displayed on the screen, but the entire rendering process is the same as the normal mode, which can handle some things that users do not need to see. image data.

Bind OpenGL rendering thread and drawing context: makeCurrent

Use eglMakeCurrent to implement binding.

At this point, EGL can be initialized using the method encapsulated in EGLCore. But still did not answer the question mentioned above.

The answer lies in glMakeCurrent.

> The glMakeCurrent method implements the binding of the device display window (EGLDisplay), the OpenGL context (EGLContext), the image data cache (GLSurface), and the current thread.

Note here: " **Bindings of the current thread** ".

Now to answer the question posed above: Why does OpenGL draw correctly in multiple GLSurfaceViews?

After EGL is initialized, that is, after the rendering environment (EGLDisplay, EGLContext, GLSurface) is ready, glMakeCurrent needs to be called explicitly in the rendering thread (the thread that draws the image). At this time, the bottom layer of the system will bind the OpenGL rendering environment to the current thread.

After that, as long as you call any OpenGL ES API in the rendering thread (such as the method GLES20.glGenTextures for generating texture IDs), OpenGL will automatically switch contexts (that is, switch OpenGL rendering information and resources) according to the current thread.

In other words, if you call the OpenGL API in a thread that does not call glMakeCurrent, the system will not be able to find the corresponding OpenGL context and the corresponding resources, which may cause an exception error.

This is why some articles say that OpenGL rendering must be done in the OpenGL thread.

> In fact, the three callback methods of GLSurfaceView#Renderer are all called in GLThread.

- Swap cached data, and display images: swapBuffers

  > eglSwapBuffers is a method provided by EGL to display EGLSurface data to the device screen. After OpenGL draws the image, call this method to actually display it.
- Unbind data cache surfaces, and release resources
  - When the Surface on the page is destroyed (such as the App goes to the background), the resources need to be unbound.
  - When the page exits, the SurfaceView is destroyed and all resources need to be released.

The above only encapsulates the core API, and then we need to create a new class to call it.

2. Call the EGL core method

Here, create a new EGLSurfaceHolder for operating EGLCore

```
class EGLSurfaceHolder {

    private val TAG = "EGLSurfaceHolder"

    private lateinit var mEGLCore: EGLCore

    private var mEGLSurface: EGLSurface? = null

    fun init(shareContext: EGLContext? = null, flags: Int) {
        mEGLCore = EGLCore()
        mEGLCore.init(shareContext, flags)
    }

    fun createEGLSurface(surface: Any?, width: Int = -1, height: Int = -1) {
        mEGLSurface = if (surface != null) {
            mEGLCore.createWindowSurface(surface)
        } else {
            mEGLCore.createOffscreenSurface(width, height)
        }
    }

    fun makeCurrent() {
        if (mEGLSurface != null) {
            mEGLCore.makeCurrent(mEGLSurface!!)
        }
    }

    fun swapBuffers() {
        if (mEGLSurface != null) {
            mEGLCore.swapBuffers(mEGLSurface!!)
        }
    }

    fun destroyEGLSurface() {
        if (mEGLSurface != null) {
            mEGLCore.destroySurface(mEGLSurface!!)
            mEGLSurface = null
        }
    }

    fun release() {
        mEGLCore.release()
    }
}
```

The code is very simple, the most important thing is to hold EGLSurface (of course, you can also put EGLSurface in EGLCore), and open more concise EGL operation methods for external calls.

3. Simulate GLSurfaceView and use EGL to achieve rendering

In order to better understand EGL, here is how to use EGL by simulating GLSurfaceView.

> Customize a renderer CustomerRender

```
class CustomerGLRenderer : SurfaceHolder.Callback {

    //OpenGL渲染线程
    private val mThread = RenderThread()

    //页面上的SurfaceView弱引用
    private var mSurfaceView: WeakReference<SurfaceView>? = null

    //所有的绘制器
    private val mDrawers = mutableListOf<IDrawer>()

    init {
        //启动渲染线程
        mThread.start()
    }

    /**
     * 设置SurfaceView
     */
    fun setSurface(surface: SurfaceView) {
        mSurfaceView = WeakReference(surface)
        surface.holder.addCallback(this)

        surface.addOnAttachStateChangeListener(object :
View.OnAttachStateChangeListener{
            override fun onViewDetachedFromWindow(v: View?) {
                mThread.onSurfaceStop()
            }

            override fun onViewAttachedToWindow(v: View?) {
            }
        })
    }

    /**
     * 添加绘制器
     */
    fun addDrawer(drawer: IDrawer) {
        mDrawers.add(drawer)
    }

    override fun surfaceCreated(holder: SurfaceHolder?) {
        mThread.onSurfaceCreate()
    }

    override fun surfaceChanged(holder: SurfaceHolder?, format: Int, width: Int,
height: Int) {
        mThread.onSurfaceChange(width, height)
    }

    override fun surfaceDestroyed(holder: SurfaceHolder?) {
        mThread.onSurfaceDestroy()
    }
}
```

Mainly as follows:

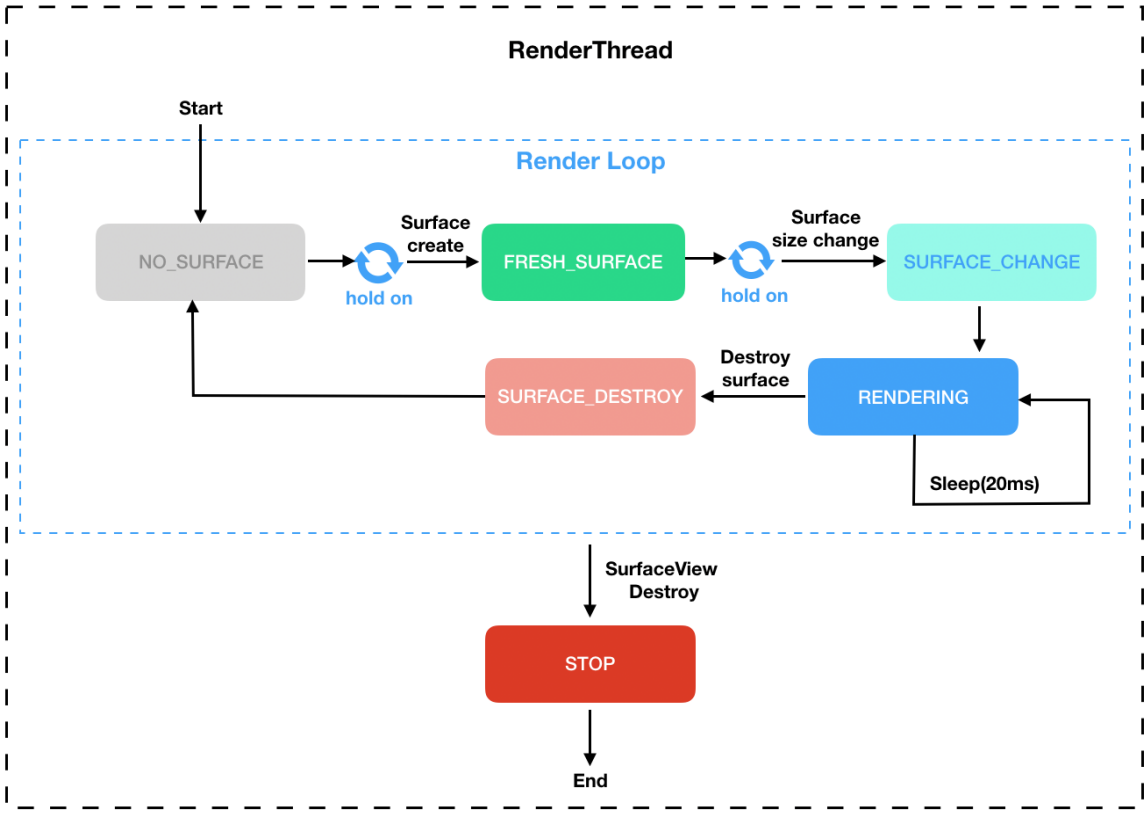
1. A custom render thread RenderThread
2. A weak reference to SurfaceView
3. a list of drawers

On initialization, start the rendering thread. Then it's just forwarding the SurfaceView lifecycle to the rendering thread, and nothing else.

Define the rendering state

```
/**
 * 渲染状态
 */
enum class RenderState {
    NO_SURFACE, //没有有效的surface
    FRESH_SURFACE, //持有一个未初始化的新的surface
    SURFACE_CHANGE, // surface尺寸变化
    RENDERING, //初始化完毕，可以开始渲染
    SURFACE_DESTROY, //surface销毁
    STOP //停止绘制
}
```

According to these states, in RenderThread, switch the execution state of the thread.



Rendering state switching flow chart

described as follows:

1. When the thread starts and enters the while(true) loop, the state is NO_SURFACE, and the thread enters the wait (hold on);

2. After Surface create, the state becomes FRESH_SURFACE;
3. After Surface change, enter SURFACE_CHANGE state;
4. After executing SURFACE_CHANGE, it will automatically enter the RENDERING state;
5. In the absence of other interruptions, execute Render every 20ms;
6. If the Surface is destroyed, re-enter the NO_SURFACE state; if there is a new surface, re-execute 2-5;
7. If the SurfaceView is destroyed and enters the STOP state, the rendering thread exits, end.

   Execute the render loop

```
inner class RenderThread: Thread() {

    // 渲染状态
    private var mState = RenderState.NO_SURFACE

    private var mEGLSurface: EGLSurfaceHolder? = null

    // 是否绑定了EGLSurface
    private var mHaveBindEGLContext = false

    //是否已经新建过EGL上下文，用于判断是否需要生产新的纹理ID
    private var mNeverCreateEglContext = true

    private var mWidth = 0
    private var mHeight = 0

    private val mWaitLock = Object()

//------------第1部分：线程等待与解锁-----------------

    private fun holdOn() {
        synchronized(mWaitLock) {
            mWaitLock.wait()
        }
    }

    private fun notifyGo() {
        synchronized(mWaitLock) {
            mWaitLock.notify()
        }
    }

//------------第2部分：Surface声明周期转发函数------------

    fun onSurfaceCreate() {
        mState = RenderState.FRESH_SURFACE
        notifyGo()
    }

    fun onSurfaceChange(width: Int, height: Int) {
        mWidth = width
        mHeight = height
        mState = RenderState.SURFACE_CHANGE
        notifyGo()
    }

    fun onSurfaceDestroy() {
        mState = RenderState.SURFACE_DESTROY
        notifyGo()
    }

    fun onSurfaceStop() {
        mState = RenderState.STOP
        notifyGo()
    }
```

```
//------------第3部分：OpenGL渲染循环------------

    override fun run() {
        //  【1】初始化EGL
        initEGL()
        while (true) {
            when (mState) {
                RenderState.FRESH_SURFACE -> {
                    //【2】使用surface初始化EGLSurface，并绑定上下文
                    createEGLSurfaceFirst()
                    holdOn()
                }
                RenderState.SURFACE_CHANGE -> {
                    createEGLSurfaceFirst()
                    //【3】初始化OpenGL世界坐标系宽高
                    GLES20.glViewport(0, 0, mWidth, mHeight)
                    configWordSize()
                    mState = RenderState.RENDERING
                }
                RenderState.RENDERING -> {
                    //【4】进入循环渲染
                    render()
                }
                RenderState.SURFACE_DESTROY -> {
                    //【5】销毁EGLSurface，并解绑上下文
                    destroyEGLSurface()
                    mState = RenderState.NO_SURFACE
                }
                RenderState.STOP -> {
                    //【6】释放所有资源
                    releaseEGL()
                    return
                }
                else -> {
                    holdOn()
                }
            }
            sleep(20)
        }
    }

//------------第4部分：EGL相关操作------------

    private fun initEGL() {
        mEGLSurface = EGLSurfaceHolder()
        mEGLSurface?.init(null, EGL_RECORDABLE_ANDROID)
    }

    private fun createEGLSurfaceFirst() {
        if (!mHaveBindEGLContext) {
            mHaveBindEGLContext = true
            createEGLSurface()
            if (mNeverCreateEglContext) {
                mNeverCreateEglContext = false
                generateTextureID()
            }
```

```
        }
    }

    private fun createEGLSurface() {
        mEGLSurface?.createEGLSurface(mSurfaceView?.get()?.holder?.surface)
        mEGLSurface?.makeCurrent()
    }

    private fun destroyEGLSurface() {
        mEGLSurface?.destroyEGLSurface()
        mHaveBindEGLContext = false
    }

    private fun releaseEGL() {
        mEGLSurface?.release()
    }

//------------第5部分：OpenGL ES相关操作-------------

    private fun generateTextureID() {
        val textureIds = OpenGLTools.createTextureIds(mDrawers.size)
        for ((idx, drawer) in mDrawers.withIndex()) {
            drawer.setTextureID(textureIds[idx])
        }
    }

    private fun configWordSize() {
        mDrawers.forEach { it.setWorldSize(mWidth, mHeight) }
    }

    private fun render() {
        GLES20.glClear(GLES20.GL_COLOR_BUFFER_BIT or GLES20.GL_DEPTH_BUFFER_BIT)
        mDrawers.forEach { it.draw() }
        mEGLSurface?.swapBuffers()
    }
}
```

Mainly divided into 5 parts, 1-2 is very simple, I believe everyone can understand. As for 4-5, they are all methods called in run.

Focus on the third part, which is the run method.

[1] Before entering while(true), initEGL uses EGLSurfaceHolder to initialize EGL.

> It should be noted that initEGL will only be called once, which means that EGL is only initialized once, no matter how many times the surface is destroyed and rebuilt later.

[2] After there is an available surface, enter the FRESH_SURFACE state and call createEGLSurface and makeCurrent of EGLSurfaceHolder to bind threads, contexts and windows.

[3] According to the width and height of the surface window, set the width and height of the OpenGL window, and then automatically enter the RENDERING state. This part corresponds to the callback onSurfaceChanged method in GLSurfaceView.Renderer.

[4] Enter the loop rendering render, where the screen is rendered every 20ms. Corresponds to the callback onDrawFrame method in GLSurfaceView.Renderer.

For the convenience of comparison, here is the SimpleRender defined in the previous article as follows:

```
class SimpleRender: GLSurfaceView.Renderer {

    private val drawers = mutableListOf<IDrawer>()

    override fun onSurfaceCreated(gl: GL10?, config: EGLConfig?) {
        GLES20.glClearColor(0f, 0f, 0f, 0f)
        //开启混合，即半透明
        GLES20.glEnable(GLES20.GL_BLEND)
        GLES20.glBlendFunc(GLES20.GL_SRC_ALPHA, GLES20.GL_ONE_MINUS_SRC_ALPHA)

        val textureIds = OpenGLTools.createTextureIds(drawers.size)
        for ((idx, drawer) in drawers.withIndex()) {
            drawer.setTextureID(textureIds[idx])
        }
    }

    override fun onSurfaceChanged(gl: GL10?, width: Int, height: Int) {
        GLES20.glViewport(0, 0, width, height)
        for (drawer in drawers) {
            drawer.setWorldSize(width, height)
        }
    }

    override fun onDrawFrame(gl: GL10?) {
        GLES20.glClear(GLES20.GL_COLOR_BUFFER_BIT or GLES20.GL_DEPTH_BUFFER_BIT)
        drawers.forEach {
            it.draw()
        }
    }

    fun addDrawer(drawer: IDrawer) {
        drawers.add(drawer)
    }
}
```

[5] If the surface is destroyed (for example, entering the background), call destroyEGLSurface of EGLSurfaceHolder to destroy and unbind the window.

> Note: When the page returns to the foreground, the surface will be recreated. At this time, as long as the EGLSurface is recreated and the context and EGLSurface are bound, the screen can continue to be rendered without opening a new rendering thread.

[6] SurfaceView is destroyed (for example, page finish), then rendering is no longer needed, all EGL resources need to be released, and the thread exits.

4. Use the renderer

New page EGLPlayerActivity

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
xmlns:app="http://schemas.android.com/apk/res-auto">
    <SurfaceView
            android:id="@+id/sfv"
            app:layout_constraintTop_toTopOf="parent"
            android:layout_width="match_parent"
            android:layout_height="match_parent"/>
</android.support.constraint.ConstraintLayout>
```

```
class EGLPlayerActivity: AppCompatActivity() {
    private val path = Environment.getExternalStorageDirectory().absolutePath +
"/mvtest_2.mp4"
    private val path2 = Environment.getExternalStorageDirectory().absolutePath +
"/mvtest.mp4"

    private val threadPool = Executors.newFixedThreadPool(10)

    private var mRenderer = CustomerGLRenderer()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_egl_player)
        initFirstVideo()
        initSecondVideo()
        setRenderSurface()
    }

    private fun initFirstVideo() {
        val drawer = VideoDrawer()
        drawer.setVideoSize(1920, 1080)
        drawer.getSurfaceTexture {
            initPlayer(path, Surface(it), true)
        }
        mRenderer.addDrawer(drawer)
    }

    private fun initSecondVideo() {
        val drawer = VideoDrawer()
        drawer.setAlpha(0.5f)
        drawer.setVideoSize(1920, 1080)
        drawer.getSurfaceTexture {
            initPlayer(path2, Surface(it), false)
        }
        mRenderer.addDrawer(drawer)

        Handler().postDelayed({
            drawer.scale(0.5f, 0.5f)
        }, 1000)
    }

    private fun initPlayer(path: String, sf: Surface, withSound: Boolean) {
        val videoDecoder = VideoDecoder(path, null, sf)
        threadPool.execute(videoDecoder)
        videoDecoder.goOn()

        if (withSound) {
            val audioDecoder = AudioDecoder(path)
            threadPool.execute(audioDecoder)
            audioDecoder.goOn()
        }
    }

    private fun setRenderSurface() {
        mRenderer.setSurface(sfv)
```

```
    }
}
```

The entire usage process is almost the same as in the previous article, using GLSurfaceView to render video images.

The only difference is that SurfaceView needs to be set to CustomerRenderer.

At this point, the video can be played. The basic knowledge of EGL and how to use it are basically finished.

However, it seems that I haven't found the real use of EGL. GLSurfaceView has all the things that should be there. Why should I learn EGL?

And listen to me continue to blow water, hahaha.

3. The purpose of EGL

1. Deepen your understanding of OpenGL

If you haven't studied EGL seriously, your OpenGL career will be incomplete, because you can't deeply understand what the OpenGL rendering mechanism is, and you will be very powerless when dealing with some problems.

2, Android video hard coding must use EGL

If you need to use the encoding capabilities of Android Mediacodec, then EGL is essential. In the subsequent articles about video encoding, you will see how to use EGL to achieve encoding.

3. FFmpeg codec requires EGL-related knowledge

At the JNI layer, Android does not implement a tool similar to GLSurfaceView to help us hide EGL-related content. Therefore, if you need to implement FFmpeg encoding and decoding at the C++ layer, you need to implement the entire OpenGL rendering process yourself.

> This is the real purpose of learning EGL. If it is only used for rendering video pictures, GLSurfaceView is enough for us.

So, EGL, must learn!

4. Reference articles

EGL use practice of OpenGL

Analysis of Android system EGL and GL threads from the perspective of source code