游戏通用功能底层逻辑 Android SDK 封装

deepwaterooo

December 3, 2022

Contents

1 Android SDK 封装 for unity games

- Initial design, to be modified and linked later
- Will work on packing lower layer Android SDK feature/functionality packing for coming a few days
- 屏幕适配可需要再改一下

2 Unity 安卓共享纹理

- 之前一直有想法: 就是把游戏中所用到的平移 (四个按钮) 与旋转 (六个按钮) 做成安卓原生开发, 做成透明, 与游戏端画面叠加绘制
  - 原本也可以不用, 前提是我能够在游戏端将这两块画面做得漂亮
  - 自己做不漂亮的时候, 就去想, 能不能从安卓端原生画出来, 毕竟画出来的刻度会是精准的, 并将十个按钮设置为半透明, 达到精准完美的程度
- 现在看, 这些实现起来都是有理论支撑, 可以做到的, 参考的 github 项目也已经被我 fork 到自己的了
- 等安卓 SDK 接完, 可能会试一下自己游戏中能够实现这些
- 一些基础原理:

```
public class GLTexture {
    private static final String TAG = "GLTexture";

    private static final String imageFilePath = "/sdcard/1Atest/image.jpg";

    private int mTextureID = 0;
    private int mTextureWidth = 0;
    private int mTextureHeight = 0;

    SurfaceTexture mCameraInputSurface;
    SurfaceTexture mOutputSurfaceTexture;
    int mOutputTex[];

// OpenGL 渲染的上下文及配置: 多线程安全 (安卓 游戏)
    private volatile EGLContext mSharedEglContext;
    private volatile EGLConfig mSharedEglConfig;
```

```
private EGLDisplay mEGLDisplay;
private EGLContext mEglContext;
private EGLSurface mEglSurface;

// 创建单线程池，用于处理 OpenGL 纹理
private final ExecutorService mRenderThread = Executors.newSingleThreadExecutor();
// 使用 Unity 线程 Looper 的 Handler，用于执行 Java 层的 OpenGL 操作
private Handler mUnityRenderHandler;

public GLTexture() { }
public int getStreamTextureWidth() {
    //Log.d(TAG,"mTextureWidth = "+ mTextureWidth);
    return mTextureWidth;
}
public int getStreamTextureHeight() {
    //Log.d(TAG,"mTextureHeight = "+ mTextureHeight);
    return mTextureHeight;
}
public int getStreamTextureID() {
    Log.d(TAG,"getStreamTextureID sucess = "+ mTextureID);
    return mTextureID;
}
private void glLogE(String msg) {
    Log.e(TAG, msg + ", err=" + GLES20.glGetError());
}

// 被 unity 调用
public void setupOpenGL() {
    Log.d(TAG, "setupOpenGL called by Unity ");

    // 注意：该调用一定是从 Unity 绘制线程发起
    if (Looper.myLooper() == null) {
        Looper.prepare();
    }
    mUnityRenderHandler = new Handler(Looper.myLooper());

    // Unity 获取 EGLContext
    mSharedEglContext = EGL14.eglGetCurrentContext();
    if (mSharedEglContext == EGL14.EGL_NO_CONTEXT) {
        glLogE("eglGetCurrentContext failed");
        return;
    }
    glLogE("eglGetCurrentContext success");

    EGLDisplay sharedEglDisplay = EGL14.eglGetCurrentDisplay();
    if (sharedEglDisplay == EGL14.EGL_NO_DISPLAY) {
        glLogE("sharedEglDisplay failed");
        return;
    }
    glLogE("sharedEglDisplay success");

    // 获取 Unity 绘制线程的 EGLConfig
    int[] numEglConfigs = new int[1];
    EGLConfig[] eglConfigs = new EGLConfig[1];
    if (!EGL14.eglGetConfigs(sharedEglDisplay, eglConfigs, 0, eglConfigs.length,
                             numEglConfigs, 0)) {
        glLogE("eglGetConfigs failed");
        return;
    }
    mSharedEglConfig = eglConfigs[0];
    mRenderThread.execute(new Runnable() {
            @Override
            public void run() {
                // 初始化 OpenGL 环境
                initOpenGL();
                // 生成 OpenGL 纹理 ID
                int textures[] = new int[1];
                GLES20.glGenTextures(1, textures, 0);
                if (textures[0] == 0) { glLogE("glGenTextures failed"); return; }
                else { glLogE("glGenTextures success"); }
                mTextureID = textures[0];
                mTextureWidth = 670;
                mTextureHeight = 670;
            }
        });
}
private void initOpenGL() {
    mEGLDisplay = EGL14.eglGetDisplay(EGL14.EGL_DEFAULT_DISPLAY);
    if (mEGLDisplay == EGL14.EGL_NO_DISPLAY) {
```

```
            glLogE("eglGetDisplay failed");
            return;
        }
        glLogE("eglGetDisplay success");

        int[] version = new int[2];
        if (!EGL14.eglInitialize(mEGLDisplay, version, 0, version, 1)) {
            mEGLDisplay = null;
            glLogE("eglInitialize failed");
            return;
        }
        glLogE("eglInitialize success");

        int[] eglContextAttribList = new int[]{
            EGL14.EGL_CONTEXT_CLIENT_VERSION, 3, // 该值需与 Unity 绘制线程使用的一致
            EGL14.EGL_NONE
        };
        // 创建 Java 线程的 EGLContext 时，将 Unity 线程的 EGLContext 和 EGLConfig 作为参数传递给 eglCreateContext，
        // 从而实现两个线程共享 EGLContext
        mEglContext = EGL14.eglCreateContext(mEGLDisplay, mSharedEglConfig, mSharedEglContext,
                                             eglContextAttribList, 0);
        if (mEglContext == EGL14.EGL_NO_CONTEXT) {
            glLogE("eglCreateContext failed");
            return;
        }
        glLogE("eglCreateContext success");

        int[] surfaceAttribList = {
            EGL14.EGL_WIDTH, 64,
            EGL14.EGL_HEIGHT, 64,
            EGL14.EGL_NONE
        };
        // Java 线程不进行实际绘制，因此创建 PbufferSurface 而非 WindowSurface
        // 创建 Java 线程的 EGLSurface 时，将 Unity 线程的 EGLConfig 作为参数传递给 eglCreatePbufferSurface
        mEglSurface = EGL14.eglCreatePbufferSurface(mEGLDisplay, mSharedEglConfig, surfaceAttribList, 0);
        if (mEglSurface == EGL14.EGL_NO_SURFACE) {
            glLogE("eglCreatePbufferSurface failed");
            return;
        }
        glLogE("eglCreatePbufferSurface success");

        if (!EGL14.eglMakeCurrent(mEGLDisplay, mEglSurface, mEglSurface, mEglContext)) {
            glLogE("eglMakeCurrent failed");
            return;
        }
        glLogE("eglMakeCurrent success");

        GLES20.glFlush();
    }
    public void updateTexture() {
        // Log.d(TAG,"updateTexture called by unity");
        mRenderThread.execute(new Runnable() {
                @Override
                public void run() {
                    final Bitmap bitmap = BitmapFactory.decodeFile(imageFilePath);
//                  if(bitmap == null)
//                      Log.d(TAG,"bitmap decode faild" + bitmap);
//                  else
//                      Log.d(TAG,"bitmap decode success" + bitmap);
                      mUnityRenderHandler.post(new Runnable() {
                              @Override
                              public void run() {
                                  GLES20.glBindTexture(GLES20.GL_TEXTURE_2D, mTextureID);
                                  GLES20.glTexParameteri(GLES11Ext.GL_TEXTURE_EXTERNAL_OES, GLES20.GL_TEXTURE_MIN_FILTER, GLE
                                  GLES20.glTexParameteri(GLES11Ext.GL_TEXTURE_EXTERNAL_OES, GLES20.GL_TEXTURE_MAG_FILTER, GLE
                                  GLES20.glTexParameteri(GLES20.GL_TEXTURE_2D, GLES20.GL_TEXTURE_WRAP_S, GLES20.GL_CLAMP_TO_E
                                  GLES20.glTexParameteri(GLES20.GL_TEXTURE_2D, GLES20.GL_TEXTURE_WRAP_T, GLES20.GL_CLAMP_TO_E
                                  GLES20.glTexParameteri(GLES20.GL_TEXTURE_2D, GLES20.GL_TEXTURE_MAG_FILTER, GLES20.GL_LINEAR
                                  GLES20.glTexParameteri(GLES20.GL_TEXTURE_2D, GLES20.GL_TEXTURE_MIN_FILTER, GLES20.GL_LINEAR
                                  GLUtils.texImage2D(GLES20.GL_TEXTURE_2D, 0, bitmap, 0);
                                  GLES20.glBindTexture(GLES20.GL_TEXTURE_2D, 0);
                                  bitmap.recycle();
                              }
                          });
                }
            });
    }
    public void destroy() {
        mRenderThread.shutdownNow();
```

```
    }
}
```