> VideoFileRenderer实现了VideoRenderer的Callbacks，可以回调到产生的Frame。 它的主要功能是将产生的视频流以文件的形式存储在本地。

```
  //定义一个新的工作的线程
  private final HandlerThread renderThread;

  //定义一个线程锁
  private final Object handlerLock = new Object();

  //定义一个Handler
  private final Handler renderThreadHandler;

  //文件输出的流
  private final FileOutputStream videoOutFile;

  //流的属性
  private final int outputFileWidth;
  private final int outputFileHeight;

  //帧的数量
  private final int outputFrameSize;

  //一个输出的Buffer
  private final ByteBuffer outputFrameBuffer;

  //全局的上下文
  private EglBase eglBase;

  //信号的转化
  private YuvConverter yuvConverter;
```

```
//构造方法，四个参数分别是目标文件，文件的宽，文件的高度，上下文
public VideoFileRenderer(String outputFile, int outputFileWidth, int outputFileHeight,
      final EglBase.Context sharedContext) throws IOException {
    //向外写文件宽高都应该是偶数
    if ((outputFileWidth % 2) == 1 || (outputFileHeight % 2) == 1) {
      throw new IllegalArgumentException("Does not support uneven width or height");
    }

    this.outputFileWidth = outputFileWidth;
    this.outputFileHeight = outputFileHeight;

    //计算frameSize
    outputFrameSize = outputFileWidth * outputFileHeight * 3 / 2;
    outputFrameBuffer = ByteBuffer.allocateDirect(outputFrameSize);

    //写文件
    videoOutFile = new FileOutputStream(outputFile);
    videoOutFile.write(
        ("YUV4MPEG2 C420 W" + outputFileWidth + " H" + outputFileHeight + " Ip F30:1 A1:1\n")
            .getBytes());

    //开启一个新线程
    renderThread = new HandlerThread(TAG);
    renderThread.start();
    renderThreadHandler = new Handler(renderThread.getLooper());

    ThreadUtils.invokeAtFrontUninterruptibly(renderThreadHandler, new Runnable() {
      @Override
      public void run() {
        //在新的线程中初始化全局，并且进行编码转化
        eglBase = EglBase.create(sharedContext, EglBase.CONFIG_PIXEL_BUFFER);
        eglBase.createDummyPbufferSurface();
        eglBase.makeCurrent();
        yuvConverter = new YuvConverter();
      }
    });
  }
```

```
@Override
  public void renderFrame(final VideoRenderer.I420Frame frame) {
    renderThreadHandler.post(new Runnable() {
      @Override
      public void run() {
        //当有流产生的时候要进行回调4
        renderFrameOnRenderThread(frame);
      }
    });
  }

  private void renderFrameOnRenderThread(VideoRenderer.I420Frame frame) {
    //呈现画面在新的线程中

    //画面呈现的比例
    final float frameAspectRatio = (float) frame.rotatedWidth() / (float) frame.rotatedHeight();

    //旋转抽样矩阵
    final float[] rotatedSamplingMatrix =
        RendererCommon.rotateTextureMatrix(frame.samplingMatrix, frame.rotationDegree);


    final float[] layoutMatrix = RendererCommon.getLayoutMatrix(
        false, frameAspectRatio, (float) outputFileWidth / outputFileHeight);
    final float[] texMatrix = RendererCommon.multiplyMatrices(rotatedSamplingMatrix, layoutMatrix);

    //==========以下是图像处理的代码，真的看不太懂，日后钻研明白再来解析====================
    try {
      videoOutFile.write("FRAME\n".getBytes());
      if (!frame.yuvFrame) {
        yuvConverter.convert(outputFrameBuffer, outputFileWidth, outputFileHeight, outputFileWidth,
            frame.textureId, texMatrix);

        int stride = outputFileWidth;
        byte[] data = outputFrameBuffer.array();
        int offset = outputFrameBuffer.arrayOffset();

        // Write Y
        videoOutFile.write(data, offset, outputFileWidth * outputFileHeight);

        // Write U
        for (int r = outputFileHeight; r < outputFileHeight * 3 / 2; ++r) {
          videoOutFile.write(data, offset + r * stride, stride / 2);
        }

        // Write V
        for (int r = outputFileHeight; r < outputFileHeight * 3 / 2; ++r) {
          videoOutFile.write(data, offset + r * stride + stride / 2, stride / 2);
        }
      } else {
        nativeI420Scale(frame.yuvPlanes[0], frame.yuvStrides[0], frame.yuvPlanes[1],
            frame.yuvStrides[1], frame.yuvPlanes[2], frame.yuvStrides[2], frame.width, frame.height,
            outputFrameBuffer, outputFileWidth, outputFileHeight);
        videoOutFile.write(
            outputFrameBuffer.array(), outputFrameBuffer.arrayOffset(), outputFrameSize);
      }
    } catch (IOException e) {
      Logging.e(TAG, "Failed to write to file for video out");
      throw new RuntimeException(e);
    } finally {
      VideoRenderer.renderFrameDone(frame);
    }
  }
```

```
//释放资源，关闭文件等处理
public void release() {
    final CountDownLatch cleanupBarrier = new CountDownLatch(1);
    renderThreadHandler.post(new Runnable() {
      @Override
      public void run() {
        try {
          videoOutFile.close();
        } catch (IOException e) {
          Logging.d(TAG, "Error closing output video file");
        }
        yuvConverter.release();
        eglBase.release();
        renderThread.quit();
        cleanupBarrier.countDown();
      }
    });
    ThreadUtils.awaitUninterruptibly(cleanupBarrier);
  }
```