

Android OpenGL ES Issues

OmniG Software Inc. (www.omnigsoft.com)

Monday, January 21, 2008

Contact: hongkun@omnigsoft.com



X-Benchmark for Android

This document lists all issues we got during the development of OmniGSoft X-Benchmark for Android. X-Benchmark is a comprehensive mobile performance benchmark, which tests the most important capabilities of a mobile device, including: processor, memory, I/O, 2D graphics, 3D graphics.

While developing the 3D graphics test module for Android OpenGL ES implementation, we got several issues in OpenGL ES. The issues listed here are based on X-Benchmark running on the latest Android SDK ([android_sdk_windows_m3-rc37a.zip](#)). We also include the test result of X-Benchmark Java ME edition running on Sun Wireless Toolkit 2.5 with Sun OpenGL ES implementation (JSR-239 Java Binding for OpenGL ES).

All Android OpenGL ES issues listed here are concluded from comparison between Android OpenGL ES test results and Sun OpenGL ES test results. However, they are still not finally confirmed, which means:

1. The issue could be a bug in Android OpenGL ES implementation.
2. This issue might be caused by improper usage of OpenGL ES functions, such as incorrect function parameters.

Issue 1: OpenGL Fog causes wrong 3D rendering

```
gl.glFogx(GL10.GL_FOG_MODE, GL10.GL_LINEAR);  
gl.glFogf(GL10.GL_FOG_START, fog.frontDistance);  
gl.glFogf(GL10.GL_FOG_END, fog.backDistance);
```



Left: Sun Wireless Toolkit (JSR-239); Right: Android OpenGL ES

Issue 2: OpenGL Fog function glFogf() causes “Invalid Enum” exception

When calling glFogf() with any one OpenGL constant of GL_LINEAR, GL_EXP, GL_EXP2, Android OpenGL generate an “Invalid Enum” exception. Following is the Dalvik Debug output:

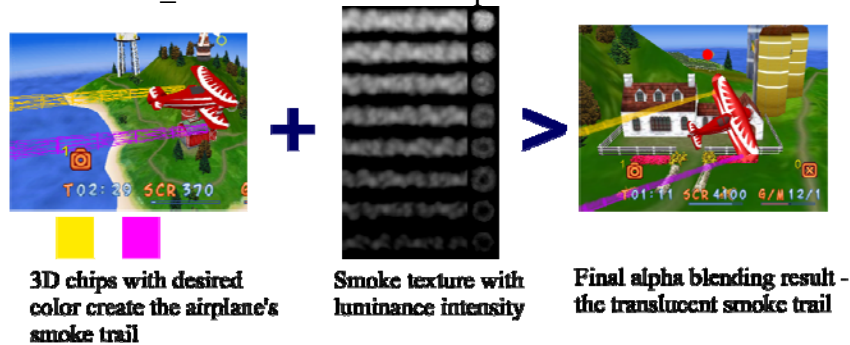
```
01-21 16:42:... I 599 OmniGSoft android.opengl.GLException: invalid enum
01-21 16:42:... I 599 OmniGSoft at android.opengl.GLErrorWrapper.checkError(GLErrorWrapper.java:43)
01-21 16:42:... I 599 OmniGSoft at android.opengl.GLErrorWrapper.glFogf(GLErrorWrapper.java:238)
```

```
gl.glFogf(GL10.GL_FOG_MODE, GL10.GL_LINEAR);
gl.glFogf(GL10.GL_FOG_MODE, GL10.GL_EXP);
gl.glFogf(GL10.GL_FOG_MODE, GL10.GL_EXP2);
```

If remove glFogf() calling, the OpenGL works, but not sure what kind of fog formula is used.

Issue 3: OpenGL Texture does not support GL_LUMINANCE_ALPHA

OpenGL constant GL_LUMINANCE_ALPHA is used to generate some special effect such as smoke, glow and shadow. In this design approach, we use a grayscale image as an alpha channel to indicate the luminance intensity, then we assign this image as a texture for a 3D object to render it as a translucent object. The following picture shows a typical example of using GL_LUMINANCE_ALPHA to simulate airplane’s smoke trail.



When using glTexImage2D() with GL_LUMINANCE_ALPHA, the 3D object disappears in 3D rendering on Android emulator, while Sun JSR-239 generates correct rendering. In the following screenshots, the red translucent “trees” are generated by an image with luminance intensity applied on a 3D chip.



Left: Sun Wireless Toolkit (JSR-239); Right: Android OpenGL ES

```

gl.glTexImage2D( GL10.GL_TEXTURE_2D, 0,
GL10.GL_LUMINANCE_ALPHA,
width, height, //Image width and height
0, GL10.GL_LUMINANCE_ALPHA, GL10.GL_UNSIGNED_BYTE,
texBuf );
gl.glTexEnvx(GL10.GL_TEXTURE_ENV, GL10.GL_TEXTURE_ENV_MODE,
GL10.GL_BLEND);

```

Issue 4: OpenGL Lighting disturbs Alpha Test

Alpha test can be used for single-level alpha channel contained in indexed-color Gif and Png Image. The Alpha test has benefit over multi-level alpha blending that it does not cause so called “transparency fighting” because the transparent part of image is completely ignored during the pixel composition, there is no blending at all, so there is no need to sort the objects in rendering queue.

When OpenGL lighting is disabled (`gl.glDisable(GL10.GL_LIGHTING)`), both Sun JSR-239 and Android generate the correct Alpha Test result. When lighting is enabled (`gl.glEnable(GL10.GL_LIGHTING)`), android alpha test fails to ignore the transparent part of image.



Sun Wireless Toolkit (JSR-239)



Android OpenGL ES

```

gl.glTexEnvx(GL10.GL_TEXTURE_ENV, GL10.GL_TEXTURE_ENV_MODE,
(isLightingEnabled?GL10.GL_MODULATE : GL10.GL_REPLACE));
if(isLightingEnabled) //lighting
{
gl.glEnable(GL10.GL_LIGHTING);
gl.glMaterialxv(face, GL10.GL_AMBIENT, OpenGLUtil.getColorxv(_mtlColor,
texture!=null?255:_mtlTransparency), 0);
gl.glMaterialxv(face, GL10.GL_DIFFUSE,
OpenGLUtil.getColorxv(_needTextureMaps?(lumiCol !=
Color.BLACK?Color.BLACK:Color.WHITE):(_mtlColor), _mtlTransparency), 0);
gl.glMaterialxv(face, GL10.GL_SPECULAR, OpenGLUtil.getColorxv(specularCol, 0),
0);

gl.glMaterialx(face, GL10.GL_SHININESS, 100);
gl.glMaterialxv(face, GL10.GL_EMISSION, OpenGLUtil.getColorxv(lumiCol, 0), 0);
}
else //No lighting
{
gl.glDisable(GL10.GL_LIGHTING);
int[] colV = OpenGLUtil.getColorxv(_mtlColor, _mtlTransparency);
gl.glColor4x(colV[0], colV[1], colV[2], colV[3]); //Set the current color
}

```

Issue 5: OpenGL Lighting disturbs Alpha Blending

Alpha blending is done by using an image with alpha channel. When lighting is disabled, Android generates correct alpha blending, when lighting is enabled, the whole object disappear. See trees and wine glasses in the following pictures.



Texture and Alpha Channel



Android OpenGL ES

Issue 6: Texture Coordinate Transformation does not work

Texture coordinate transformation is used to realize special effect like:

1. Environment mapping to realize reflective surface. In this case, the texture coordinate is transformed according to surface normal and current camera (view matrix).
2. Texture shifting and rotation. For example in flight simulation game, we shift terrain texture to simulate that the terrain is moving backward.



Left: Sun Wireless Toolkit (JSR-239); Right: Android OpenGL ES

```
gl.glMatrixMode(GL10.GL_TEXTURE);
if(envmap != null)//Need transform environment mapping
{
    matTexture.setIdentity();
    matTexture.setRotation(_objMat_camera);
    OpenGLUtil.setCurrentMatrix(gl, matTexture);
}
```

Issue 7: Z Buffer precision is very low on Android and cause Z-Fighting

Android does not provide the API to set up EGLConfig and choose pixel format. Instead, Android uses OpenGLContext to do the initialization and returns a GL10/GL11 instance. Comparing with Sun JSR-239 implementation, the depth buffer precision is very low. There should be an API for choosing the depth buffer size, between 8-bit and 16-bit.

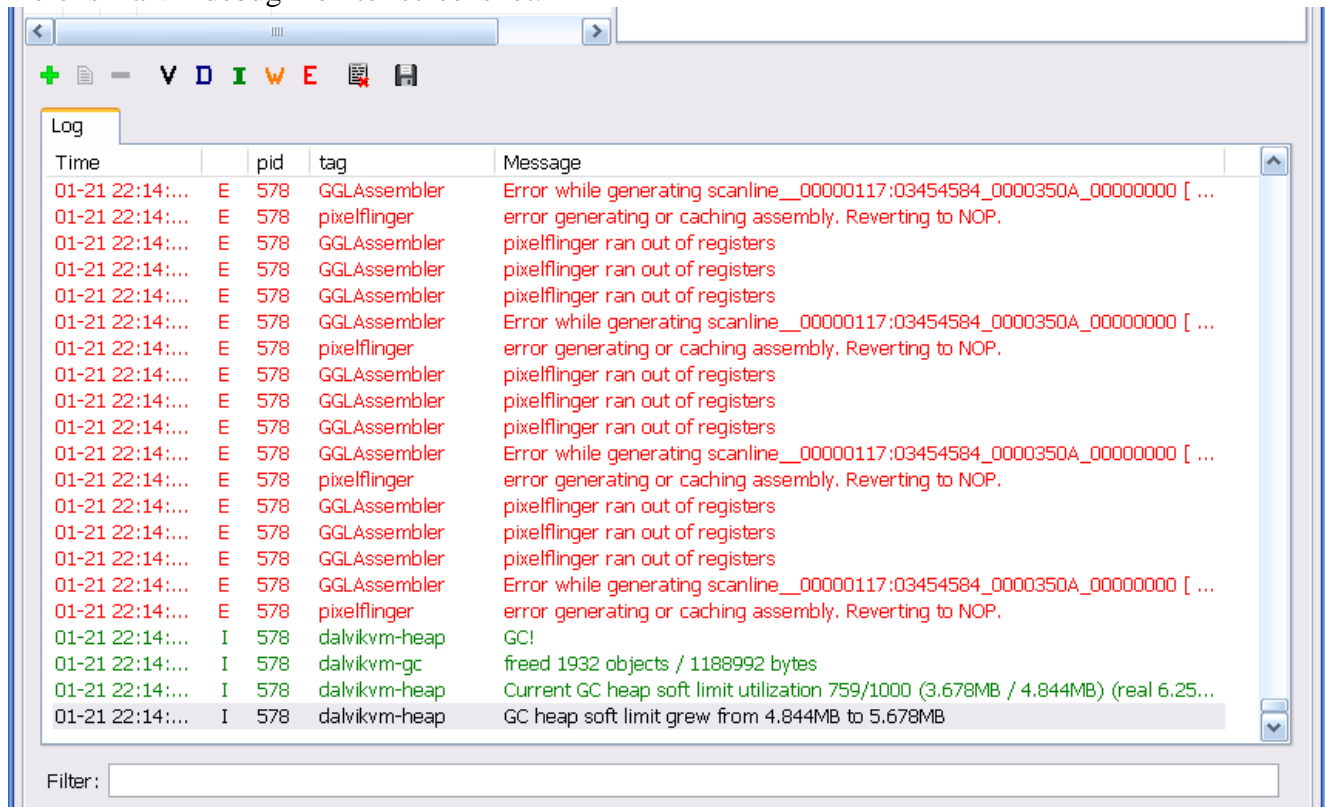
For co-planar surfaces, such as decals, we usually use Z-Bias to avoid Z-Fighting with the cost of slight performance penalty. So for the nearly co-planar surfaces, OpenGL implantation should handle it properly.



Issue 7: Keep generating “PixelFlinger ran out of Register” error

During the OpenGL rendering, Android keeps generating “pixelflinger ran out of register” in “GGLAssembler”. However, the application continue running without crash.

Here is Dalvik debug monitor screenshot.



Issue 8: Android OpenGL initialization takes very long time on Android emulator

The following code for OpenGLContext acquisition:

```
OpenGLContext staticGlc = new OpenGLContext(OpenGLContext.DEPTH_BUFFER);
```

Issue 9: glTexImage2D() takes too much time to load a texture onto OpenGL video memory

We use the following code to load texture onto OpenGL video memory:

```
gl.glTexImage2D(GL10.GL_TEXTURE_2D, 0, GL10.GL_RGBA, textureWidth, textureHeight, 0, GL10.GL_RGBA, GL10.GL_UNSIGNED_BYTE, textureBuf);
```

It seems like function glTexImage() takes long time to load a texture onto OpenGL video memory, which makes a 3D initialization of a 3D application or game very slow.

Issue 10: Overall OpenGL ES performance is not reasonable on Android emulator

Here are some screenshots of our 3D games running on Android emulator. The frame rate number (FPS) is printed at the corner of game window. We hope this is not the case for the real android devices; otherwise the 3D gaming application on Android platform will be impossible.

