Mali GPU Code Snippet: “Selecting the Correct EGLConfig”

Document Number:
Date of Issue: 18 May 2010
Product: Mali GPUs Code Snippets
Product Version: 1.0

Abstract

This document describes the code snippet “Selecting the Correct EGLConfig”

This is a working document throughout the product lifecycle and, as such, the content may be modified as new information is uncovered.

Release Information

Proprietary Notice

Document Confidentiality Status

ARM Web Address

The ARM website is located at the following address: http://www.arm.com

Feedback

ARM welcomes feedback on this product and its documentation.

Feedback on this product

If you have any comments or suggestions, please submit them to the Mali Developer Center forum www.malideveloper.com.

Feedback on this document

If you have any comments on or about this document, submit them to the Mali Developer Center forum www.malideveloper.com.

General suggestion for additions and improvements are also welcome.

Selecting the correct EGLConfig

Introduction

Each hardware platform will have its own unique set of display configurations depending on the properties of the LCD panel (or external video output e.g. HDMI), LCD controller, GPU and the available memory and bandwidth. EGL uses the EGLConfig abstraction to allow cross platform portability. For a given HW platform each EGLConfig corresponds to a possible display configuration.

In order to create a Surface (EGL terminology for the destination buffer where the output will be rendered), it is necessary to supply an EGLConfig. This EGLConfig must be chosen from all of those available by making a request detailing what properties (referred to as attributes) you require.

An example of requesting an EGLConfig

```
/* Get a display handle and initalize EGL */
EGLint major, minor;
EGLDisplay display = eglGetDisplay(EGL_DEFAULT_DISPLAY);
eglInitialize(display, &major, &minor);

/* Request an RGB565 config. with 4x anti-aliasing
 * (4x is 'free' on Mali :-)
 */
EGLint attributes = { EGL_RED_SIZE, 5,
                      EGL_GREEN_SIZE, 6,
                      EGL_BLUE_SIZE, 5,
                      EGL_SAMPLES, 4,
                      EGL_NONE }
EGLint numberConfigs;
EGLConfig* matchingConfigs;

/* Number of matching EGLConfig's returned in numberConfigs, but because
 * the 3rd argument is NULL no configs will actually be returned
 */
if (EGL_FALSE == eglChooseConfig(display, attributes, NULL,
                                         0, &numberConfigs)) {
  /* An error */
}

if (numberConfigs == 0) {
  /* An error */
}

/* Allocate some space to store list of matching configs... */
matchingConfigs = (EGLConfig*)malloc( numberConfigs * sizeof(EGLConfig));

/* ...and this time actually get the list (notice the 3rd argument is
 * now the buffer to write matching EGLConfig's to)
 */
if (EGL_FALSE == eglChooseConfig(display, attributes, matchingConfigs,
                                         numberConfigs, &numberConfigs)) {
  /* An error */
}
```

Potential for problems

However EGL does not treat all attributes equally and in some cases EGL does not look for an exact match e.g. if a request is made for an EGLConfig with a particular number of bits in the Z buffer (EGL_DEPTH_SIZE) it is considered a minimum value. An EGLConfig with a higher number of bits will still be considered. If multiple EGLConfig’s match then all will be returned.

Furthermore EGL will sort the compatible EGLConfig’s into a particular order defined in the EGL specification. This order is not always intuitive.

To illustrate these two principles consider the example of an application requiring 16 bit colour. To achieve this it might request an EGL_RED_SIZE of 5, an EGL_GREEN_SIZE of 6 and an EGL_BLUE_SIZE of 5. Colour channel depths are an attribute taken to be a minimum so the list of compatible EGLConfig’s will include all the 16 bit colour depths but also those with more colours e.g. those with EGL_RED_SIZE of 8, EGL_GREEN_SIZE of 8 and EGL_BLUE_SIZE of 8. Having found multiple matching EGLConfig’s the driver will sort them. The sorting behaviour for the colour channel depth attributes means that EGLConfig’s with a higher number of colours appear *first*. So the result is that, despite having requested a 16 bit display, the first EGLConfig in the list will actually be a 24/32bit one.

In addition there is also a concept of priority which determines the order attributes will be used to sort the matching EGLConfig’s. To illustrate this let us imagine adding an EGL_BUFFER_SIZE of 16 to the list of attributes in the request. EGL_BUFFER_SIZE is the total colour buffer depth. The sort order for EGL_BUFFER_SIZE is smallest to largest and might therefore appear to solve the problem described above because the first EGLConfig will be a 16 bit one. Unfortunately this still does not work because the priority of the EGL_BUFFER_SIZE attribute is lower than the priority of EGL_[RED|GREEN|BLUE]_SIZE. This means the driver will still sort the list by largest values of individual colour channels. In order to get a 16 bit colour buffer the, higher priority, individual colour channel attributes must be removed from the request.

Finally one has to consider whether the attributes in the request are precise enough. Using a request with just an attribute of EGL_BUFFER_SIZE of 16 will select an EGLConfig with a total colour buffer depth of 16 bits but this could be RGB565, RGBA5551 or RGBA4444. If the purpose of the EGLConfig is to draw to an OS bitmap[1] then it will be important to match the format precisely[2].

Consequences

Depending on the platform, failing to select a suitable EGLConfig, particularly in the case of colour depth, could potentially reduce performance or cause a runtime failure.

For example, in the case of mismatching colour depth/format the GPU may have to render to an off-screen temporary buffer (which requires extra memory) and then the CPU must convert each pixel from the selected format to the format actually required and finally perform a copy to the real destination (consuming many CPU cycles in the process).

[1] See the description of Pixmap’s, an EGL surface that render to OS bitmaps, in the EGL specification e.g. eglCreatePixmapSurface

[2] See the attribute EGL_MATCH_NATIVE_PIXMAP (introduce in EGL1.3) which can be used to ensure the EGLConfig is suitable

Continuation of previous example to demonstrate checking which EGLConfig is suitable

```
/* The EGLConfig we will eventually use.
 * Set to NULL to detect the case where no suitable config. Is found
 */
EGLConfig chosenConfig = NULL;

/* Look at each EGLConfig */
for (int ii=0; ii<numberConfigs; ii++) {
  EGLBoolean success;
  EGLint red, green, blue;

  /* Read the relevent attributes of the EGLConfig */
  success  = eglGetConfigAttrib(display, matchingConfigs[ii],
                                         EGL_RED_SIZE, &red);
  success &= eglGetConfigAttrib(display, matchingConfigs[ii],
                                         EGL_BLUE_SIZE, &blue);
  success &= eglGetConfigAttrib(display, matchingConfigs[ii],
                                         EGL_GREEN_SIZE, &green);

  /* Check that no error occurred and the attributes match */
  if ( success == EGL_TRUE) && (red==5) && (green==6) && (blue==5) ) {
      chosenConfig=match[ii];
      break;
    }
}

if (chosenConfig == NULL) {
  /* An error */
}

free(matchingConfigs);
```

Conculsion

Hopefully from these notes it is clear that if an application or platform has specific requirements for an EGLConfig then care must be taken to use the right attributes in the request, and if there is any possibility of an unsuitable EGLConfig being returned then the relevant attributes of each EGLConfig should be examined.

This is only a brief introduction to the subject; the precise selection, sorting and priority behaviour is described in the EGL specification available from http://www.khronos.org/registry/egl/

We recommend reading (at least) section 3.4 "Configuration Management"

Additional information

For a more complete (but different) example please see the SimpleLibrary (and related examples) available from http://www.malideveloper.com/documentation/index.php?tab=SAMPLE%20CODE

Finally we have included the source for ListConfigs; a small utility to enumerate all EGLConfig’s available on a platform. It can be modified to output all attributes of interest.