Random Hacks
Hacks, code and other things

Using OpenGL ES 2.0 on the Raspberry Pi without X windows.

Posted on April 27, 2012

29

[EDIT: This post contains information from the early days of the RasPi and may not be that relevant to its current software build. I don't actually have anything to plug my RasPi into anymore so I cannot check or update any of the video/OpenGLES work. (I have laptops but no TV – go figure.) Please see the comments for some helpful updates and potential links if this no longer works.]

The Raspberry Pi has a surprisingly sophisticated video core with a nice implementation of the OpenGL ES 2.0 interface for 3D graphics. It will be exciting to see what people do with it, but there is a small hurdle in the way at the moment for casual coders. The traditional way (on Linux) is to use the X window system to provide a surface on which the OpenGL/OpenGL ES surface can be rendered. At this moment, the implementation of X for the Raspberry Pi is unable to do this, and so you can't use X to provide your surface. I am very sure that this support will be added in time, but until then, people wanting to play with this will have to connect up the OpenGL ES side of things to the display by another method.

A big thanks to 'friggle'(IRC) aka 'asb' on github and the forums for helping me out and showing me what I might be doing wrong!

Using the display directly (technical)

Before I go any further, I want to point out that for the vast majority of people, this will be a temporary solution and in time, you'll be able to use the conventional Linux/X11 method for getting a surface to draw to. For others, they may wish to skip X11 entirely, and I'm sure that the following method will be improved in time. See below this part for link to the opengles-book examples ported with this method to run on the Raspberry Pi.

I won't be going into the intricacies of OpenGL ES 2.0 and how to write those. I recommend the OpenGL ES 2.0 programming guide (http://opengles-book.com/) as I cut my OpenGL teeth on the Red book (http://fly.cc.fer.hr/~unreal/theredbook/) and my brain is already wired up this way! Please leave links to good learning resources in the comments.

First step: setting up a connection to the hardware 'pipe', a connection to VCHIQ in your C source code.

```
import "bcm_host.h"
...
// early on, perhaps at the start of your 'main.c':
bcm_host_init();
```

The `bcm_host.h` file is within the '`/opt/vc/include`' directory on the Pi, so you will need to add that to your Makefile. The correct EGL libraries are also in a similar location ('`/opt/vc/lib`') so be sure to add that into your linker options or LD path too.

Next, create a display surface that you can use to render the EGL surface onto. This is the key setup to make! If the settings here are off then your might find that your EGL code will run, and you'll get console output, but you won't get any 3D visuals!

```
   static EGL_DISPMANX_WINDOW_T nativewindow;

   DISPMANX_ELEMENT_HANDLE_T dispman_element;
   DISPMANX_DISPLAY_HANDLE_T dispman_display;
   DISPMANX_UPDATE_HANDLE_T dispman_update;
   VC_RECT_T dst_rect;
   VC_RECT_T src_rect;

   int display_width;
   int display_height;

   // create an EGL window surface, passing context width/height
   success = graphics_get_display_size(0 /* LCD */,
                       &display_width, &display_height);
   if ( success < 0 )
   {
      return EGL_FALSE;
   }

   // You can hardcode the resolution here:
   display_width = 640;
   display_height = 480;

   dst_rect.x = 0;
   dst_rect.y = 0;
   dst_rect.width = display_width;
   dst_rect.height = display_height;

   src_rect.x = 0;
   src_rect.y = 0;
   src_rect.width = display_width << 16;
   src_rect.height = display_height << 16;

   dispman_display = vc_dispmanx_display_open( 0 /* LCD */);
   dispman_update = vc_dispmanx_update_start( 0 );

   dispman_element = vc_dispmanx_element_add ( dispman_update,
      dispman_display, 0/*layer*/, &dst_rect, 0/*src*/,
      &src_rect, DISPMANX_PROTECTION_NONE, 0 /*alpha*/,
      0/*clamp*/, 0/*transform*/);

   nativewindow.element = dispman_element;
   nativewindow.width = display_width;
   nativewindow.height = display_height;
   vc_dispmanx_update_submit_sync( dispman_update );

...
   // Pass the window to the display that have been created
   // to the esContext:
   esContext->hWnd = &nativewindow;
```

The `vc_*`, `dispman_*`, and other unusual symbols are taken from the bcm_host.h header so make sure that that is included in the source file that you use to create the context.

Final tasks: alter `eglGetDisplay` and `eglCreateWindowSurface` initialisation calls.

Instead of casting an X display or something similar in `eglGetDisplay`, you can simply do the following:

```
display = eglGetDisplay(EGL_DEFAULT_DISPLAY);
if ( display == EGL_NO_DISPLAY )
{
  return EGL_FALSE;
}
```

The next change is to `eglCreateWindowSurface`. Remember the `esContext->hWnd` pointer we set earlier? We'll use that now:

```
   surface = eglCreateWindowSurface(display, config,
                        (EGLNativeWindowType)hWnd, NULL);
   if ( surface == EGL_NO_SURFACE )
   {
      return EGL_FALSE;
   }

...
   // 'context' is returned by a "eglCreateContext" call

   // Make the context current
   if ( !eglMakeCurrent(display, surface, surface, context) )
   {
      return EGL_FALSE;
   }
```

That seems to be all you need to access and display OpenGL ES visuals directly on the Raspberry Pi without using X. Why some of the steps are the way they are I can only guess, but this at least works for me!

A port of the OpenGL ES 2.0 Programming guide examples

https://github.com/benosteen/opengles-book-samples/tree/master/Raspi (https://github.com/benosteen/opengles-book-samples/tree/master/Raspi)

Most of the key changes are in the Common/esUtil.c file where the display surface and so on are set up. This is currently hardcoded to be a 640×480 resolution window, that will appear at the bottom-left of the screen.

To build this on your Raspberry Pi:

Get the source files (https://github.com/benosteen/opengles-book-samples) onto your Pi. Either:

- `wget --no-check-certificate https://github.com/benosteen/opengles-book-samples/tarball/master (https://github.com/benosteen/opengles-book-samples/tarball/master); tar -xvzf master` Or,
- `sudo apt-get install git-core; git clone git://github.com/benosteen/opengles-book-samples.git`

Then:

```
pi@raspberrypi:~$ cd opengles-book-samples/

pi@raspberrypi:~/opengles-book-samples$ cd Raspi/

pi@raspberrypi:~/opengles-book-samples/Raspi$ make

gcc -DRPI_NO_X ./Common/esShader.c ./Common/esTransform.c ./Common/esShapes.c ./Common/esUtil.c ./Cha
gcc -DRPI_NO_X ./Common/esShader.c ./Common/esTransform.c ./Common/esShapes.c ./Common/esUtil.c ./Cha
gcc -DRPI_NO_X ./Common/esShader.c ./Common/esTransform.c ./Common/esShapes.c ./Common/esUtil.c ./Cha
gcc -DRPI_NO_X ./Common/esShader.c ./Common/esTransform.c ./Common/esShapes.c ./Common/esUtil.c ./Cha
gcc -DRPI_NO_X ./Common/esShader.c ./Common/esTransform.c ./Common/esShapes.c ./Common/esUtil.c ./Cha
gcc -DRPI_NO_X ./Common/esShader.c ./Common/esTransform.c ./Common/esShapes.c ./Common/esUtil.c ./Cha
gcc -DRPI_NO_X ./Common/esShader.c ./Common/esTransform.c ./Common/esShapes.c ./Common/esUtil.c ./Cha
gcc -DRPI_NO_X ./Common/esShader.c ./Common/esTransform.c ./Common/esShapes.c ./Common/esUtil.c ./Cha
gcc -DRPI_NO_X ./Common/esShader.c ./Common/esTransform.c ./Common/esShapes.c ./Common/esUtil.c ./Cha
gcc -DRPI_NO_X ./Common/esShader.c ./Common/esTransform.c ./Common/esShapes.c ./Common/esUtil.c ./Cha
```

Now, to test it:

```
pi@raspberrypi:~/opengles-book-samples/Raspi$ cd Chapter_2/Hello_Triangle
pi@raspberrypi:~/opengles-book-samples/Raspi/Chapter_2/Hello_Triangle$
       ./CH02_HelloTriangle
 123 frames rendered in 2.0036 seconds -> FPS=61.3894
```

NB As no X server is involved, you can actually invoke the examples from an ssh connection and have them render out to whatever video device is connected to the Pi 🙂

Tagged: *#raspberrypi*, *#raspi*, *bcm*, *howto*, *linux*, *opengles*
Posted in: *linux (https://benosteen.wordpress.com/category/linux/)*

29 Responses “Using OpenGL ES 2.0 on the Raspberry Pi without X windows.” →

Robert

April 27, 2012

Thank you! Good compilation of the information available.

Reply

Mike

May 1, 2012

Glad to see that X11 is not a requirement for hardware acceleration on the pi! I’d like to see a ‘console gaming’ library and framework for the pi running on a stripped-down linux image and X is really rather clunky for these purposes.

Reply

Joe Dobmeier

June 6, 2012

I just got tutorial 0 on the legacy GPGPU section of gpgpu.org working using simpletexture2D from chapter 9, thanks for giving me something to start from!

Reply

Mike Redrobe

July 15, 2012

Great work !
I did have a minor problem compiling it all on pi under debian wheezy:

fatal error: vcos_platform.h: No such file or directory

but solved that by copying the relevant files from the pthreads directory:

sudo cp /opt/vc/include/interface/vcos/pthreads/* /opt/vc/include/interface/vcos/

Reply

Mikael

July 19, 2012

I tried this with the new Raspbian “wheezy” image and first I got a missing include file (vcos_platform_types.h). After a quick copy of the file from pthreads dir I could compile your examples.

Unfortinatly that did not help much since I get:

* failed to open vchiq instance

when I try to run the executables.

I am starting to wonder if the Raspbian guys/gals missed something…

Reply

Elliot

July 19, 2012

Haven’t tried it yet, but this guide seems great, thanks. For those of us who have not programmed linux “the traditional way”, a quick rundown and comparison between that and this would be really helpful too.

Reply

benosteen

July 22, 2012

The main difference is the you have to get the EGL context through Xlib. You don’t really have to understand what it is, and just treat it as some boilerplate code. There is an example here: esUtil.c on github – line 112, function WinCreate does the heavy lifting to get a screen context via X

Reply

Piku

July 28, 2012

This is really helpful as I’m currently writing GLES code for iOS and want to see if I can run it on my Raspberry Pi too. The samples compile fine but some of them are a bit … strange.

For example the Chapter 8 Simple Vertex Shader doesn’t draw anything, but claims to be rendering at 60FPS. Hello Triangle works, as does the particle one from CH13. Some of the others draw random green mess instead of anything recognisable. Is this ‘normal’?

Reply

benosteen

July 28, 2012

I had something like that when running it over a composite connection, at resolution that wasn’t the full 1920*1080. Regardless of what the true resolution was, the bcm interface would reply that it was at this full resolution.

Due to the absence of documentation, I couldn’t pin down if this as expected, or if I was doing something wrong with my code. I found that if I stuck to full rez, fullscreen then it was fine. I certainly saw the green flickering mess for the chpt13 when trying to run it in anything else.

Reply

Shchvova (@Svoka)

August 20, 2012

I currently have same problem on examples. Just getting black screen at 60 fps. Is there any config to set to get it working?

Barry

August 28, 2012

I have exactly the same issues with Chapter 8 and 13 using an HDMI connection. This program is less complicated than the Hello_Pi examples – so I am somewhat at a loss.

Adam

August 7, 2012

Thanks a million! I was left scratching my head when the original examples compiled but wouldn't run, but you have made it all clear here.

Reply

FedericoRamos

September 1, 2012

In Chapter 8 only appear a black screen, ¿Some suggestion?

Reply

Rich

September 11, 2012

Hey I am having some problems linking – compile and link stage fine but then when I run ldd on Chapter2 executable, libs are listed as not found :(. Any ideas?

Reply

benosteen

September 26, 2012

Sorry for late reply, didn't spot your comment til now (wordpress…). I can only guess that the distro you are running doesn't include the binary blob /opt/vc/libs in the ld.so.conf configuration so the linker isn't finding them?

Reply

contender

September 18, 2012

Hello,

As you have been into porting chromiumos on a board.. i think this is the right place to ask about
an issue that we are facing in a similar task.

I need some insight into the way we have been proceeding in porting the chromiumos onto one of out targets which is arm based..
ours is a TV which itself is linux based.. and the idea is to support the existing TV apps along with the chromiumos..

We hacked up to boot chromiumos as follows..
we first boot into the chromium rootfs, then chroot into the the existing rootfs of our proprietary TV os, and startup few scripts that would finally start X server inside this chroot.
then we walk out of this chroot and come back on the chronos prompt..
from here we start the chrome exec once our Xserver is inited in that chroot..

Now the issue is that though the chrome browser is running and we are able to use it as a browser, we are facing an issue where the EGL is not being initialized… and the error comes as EGL_BAD_ALLOC. Though this error comes when we try to test WEBGL using chromeexperiments apps.Otherwise we dont get this error..

I am getting an error like this:
[1619:1619:162948687:ERROR:gl_surface_linux.cc(77)] GLSurfaceEGL::InitializeOneOff failed.
[1619:1619:162949466:INFO:gpu_main.cc(78)] gfx::GLSurface::InitializeOneOff failed
[1619:1619:162967605:INFO:gpu_child_thread.cc(95)] Exiting GPU process due to errors during initialization
eglInitialize failed with error EGL_BAD_ALLOC

I am still unsure how the chromium browser is running at the moment as the Xserver has been inited from inside a chroot..
is this a sane thing to do?

Is such a scenario in itself not letting EGL to initiallize properly?

Please help.
Thanks.

Reply

benosteen

September 18, 2012

I'd really like to help people here as there seems to be some real issues with certain of the demos. I can only think that the RasPi's odd colour internals might play a factor (eg I found that only 888 colour surfaces seemed to work, but this was a while ago)

My issue is that I don't have a display I can plug the Pi into at the moment, which as you can guess is a real hindrance when trying to debug these sorts of things! Please, try the forums!

Reply

shell provider

November 8, 2012

Pretty! This was a really wonderful post. Thank you for providing this information.

Reply

Peter Onion

November 16, 2012

There is a fix for the problem with the CH8 demo on the Raspberry Pi Forum:
http://www.raspberrypi.org/phpBB3/viewtopic.php?p=88616#p88616

Also I've fixed a minor problem with the code in WinCreate so it opens full screen and sets the viewport correctly.

Reply

vimal

November 27, 2012

what the hal….

Reply

vimal mac

November 27, 2012

i hate this opengl 2.0

Reply

Huw

January 13, 2013

hello, thanks for the great work. my only feedback is that you hardcoded the resolution and the viewport, i would have preferred it to be fullscreen by default. but it didnt take long to change it.

anyway, i had the issue that i think others might have been having. i was getting fizzy, noisy, half transparent, high frequency output. i started from the chapter 8 demo, and i got it after adding a uniform to the fragment program.

well this solved it for me:

http://www.raspberrypi.org/phpBB3/viewtopic.php?t=23030&p=216760

this is quite odd, might be a bug in the api?

misc info. im on hdmi. latest software and gpu firmware at time of writing. the overscan setting didnt make a difference for me.

Reply

> Huw
>
> January 13, 2013
>
> oh yes they indicate further down that thread that its a driver issue
>
> Reply

v/

March 29, 2013

A solution : NWTPi : C++ Native Window Toolkit : forget bcm dispmanx and just code OpengGL/ES ==>

https://code.google.com/p/nwtpi/

Regards,
v/

Reply

Matt

April 29, 2013

Hello, I am having an issue compiling some of the examples. vchost_config.h: No such file or directory. I have tried to modify the include path as describes here – http://www.raspberrypi.org/phpBB3/viewtopic.php?f=67&t=31805#p274386

and I have also tried to locate the vchost_config.h file in an attempt to copy it, and I can not find it anywhere.

I'm using wheezy

Reply

Martin M.

May 11, 2013

Hi, thanks to the link provided by Matt, I was able to build the examples on the recent version of Raspbian by modifying the Makefile. All I had to to, was adding another path to the INCDIR variable:

INCDIR=-I./Common -I$(SDKSTAGE)/opt/vc/include -I$(SDKSTAGE)/opt/vc/include/interface/vcos/pthreads -I$(SDKSTAGE)/opt/vc/include/interface/vmcs_host/linux

Regards

Reply

Sergio Ricardo Murguia Santana

August 15, 2013

Thanks Martin! That worked for me! I will try to port some GLSL projects I have to raspberry and will post the results somewhere

Reply

2 Trackbacks For This Post

1. *» Framebuffer on Raspberry Pi Fan's blog* →
December 29th, 2012 → 3:25 am
[…] 要直接看Framebuffer运行的例子，这里有个。编译后，在字符界面的控制台下运行，可以在屏幕上看到一个红色的方块。这个时候屏幕没有切换显示模式或者分辨率，感觉和x86下的文字/图形界面切换大不一样。更多脱离xwindows运行opengl ES的例子可以看这里。[…]

2. *OpenGL ES on Raspberry Pi without X | Coytar's Blog* →
May 4th, 2013 → 9:45 am
[…] Here's the source link: https://benosteen.wordpress.com/2012/04/27/using-opengl-es-2-0-on-the-raspberry-pi-without-x-windows/ […]