CMPT433 How-To Guide
Rendering 2D/ 3D graphics on Zen Hat LCD

Henry Chau, Gabriel Cheng, Sreeja Veeraghanta

1 Main goal

In this guide we will show you how to render graphics with GPU and display it on the Zen Hat LCD. BeagleY-AI uses ARM Cortex-A53 processor, it has a IMG BXS-4-64 GPU that can accelerate 2D/ 3D graphics. The GPU supports OpenGL ES 3.2 and Vulkan 1.2 API. In this guide we will be using OpenGL ES.

2 Prerequisite

Follow the improved SPI guide by William Schmidt [1]. The Zen Hat LCD use SPI to transmit data. The guide allows SPI data to be transmitted much faster (from 1MHz to 50MHz), increasing the display refresh rate.

3 Installation

We need to install Mesa 3D Graphics Library on host and target, it is an implementation of OpenGL ES.

1. Install the following development packages on host

   ```
   sudo apt-get install libegl1-mesa-dev libgles2-mesa-dev
   ```

2. Install the OpenGL ES dynamically linked libraries on target

   ```
   sudo apt-get install libegl1-mesa libgles2-mesa
   ```

3. (Optional) If you are using PowerVR SDK, create a symbolic link

   ```
   sudo ln -s /usr/lib/aarch64-linux-gnu/libEGL.so.1
   /usr/lib/aarch64-linux-gnu/libEGL.so
   ```

4 Provided files

We provide an LCD HAL and an example to display a triangle. Follow these instructions to compile and test the program.
On host:

```
cd triangleDemo
mkdir build
cd build
cmake ..
cmake --build .
```

On target:

```
cd /mnt/remote/myApps
sudo ./triangleDemo
```

5 Configuration and Rendering

In this section, we will show the steps for rendering a triangle on the Zen Hat LCD screen.
In your CMakeLists.txt, link EGL and GLESv2 to your executable.

```
target_link_libraries(yourExecutable PUBLIC
    ...other libraries
    EGL
    GLESv2
)
```

Rendering graphics on the Zen Hat LCD requires some workarounds. Since it is connected to the SPI peripheral, the LCD display is not recognized by the Linux OS. If you try running this command on Linux, it does not return a display device.

```
modetest -M tidss -c
```

This means that we cannot use the Linux Graphics Stack [2] for rendering because there is no default display and window system for us to use. The workaround we came up with is shown below. We have skipped the shader compilation and uploading of vertex data to the GPU because they are the same as in regular OpenGL applications. The full code can be found in the support files. Please refer to the OpenGL Wiki [3] for definitions of some of the terminologies used below.

1. Create an off-screen pixel buffer surface(display/ platform independent). We used a EGL extension to create a the display [4]. The configuration have to match the LCD specification (RGB565, 240x240).

```
// Initialize EGL
typedef EGLDisplay (*PFNEGLGETPLATFORMDISPLAYEXTPROC)(
   EGLenum, void*, const EGLint*);
```

```
PFNEGLGETPLATFORMDISPLAYEXTPROC eglGetPlatformDisplayEXT
    = (PFNEGLGETPLATFORMDISPLAYEXTPROC)eglGetProcAddress
   ("eglGetPlatformDisplayEXT");
EGLDisplay eglDisplay = eglGetPlatformDisplayEXT(
   EGL_PLATFORM_SURFACELESS_MESA, EGL_DEFAULT_DISPLAY,
   NULL);
int major, minor;
eglInitialize(eglDisplay, &major, &minor);

EGLint numConfigs;
EGLConfig eglConfig;
static const EGLint configAttribs[] = {
    EGL_SURFACE_TYPE, EGL_PBUFFER_BIT,
    EGL_BLUE_SIZE, 5,
    EGL_GREEN_SIZE, 6,
    EGL_RED_SIZE, 5,
    EGL_ALPHA_SIZE, 8,
    EGL_DEPTH_SIZE, 0,
    EGL_RENDERABLE_TYPE, EGL_OPENGL_ES3_BIT,
    EGL_NONE
};
eglChooseConfig(eglDisplay, configAttribs, &eglConfig,
   1, &numConfigs);

const EGLint pbufferAttribs[] = {
    EGL_WIDTH,  240,
    EGL_HEIGHT, 240,
    EGL_NONE,
};
EGLSurface eglSurface = eglCreatePbufferSurface(
   eglDisplay, eglConfig, pbufferAttribs);
eglBindAPI(EGL_OPENGL_ES_API);

const EGLint contextAttribs[] = {
    EGL_CONTEXT_MAJOR_VERSION, 3,
    EGL_CONTEXT_MINOR_VERSION, 0,
    EGL_NONE
};
EGLContext context = eglCreateContext(eglDisplay,
   eglConfig, EGL_NO_CONTEXT, contextAttribs);
eglMakeCurrent(eglDisplay, eglSurface, eglSurface,
   context);
```

2. Render graphics to the pixel buffer surface

```
glClearColor(0.0f, 0.5f, 0.5f, 1.0f); // Background
   color RGBA
glClear(GL_COLOR_BUFFER_BIT);
glDrawArrays(GL_TRIANGLES, 0, 3);
```

Use **glReadPixels** to transfer the frame buffer from GPU memory to client memory. Notice that we need to use a temporary frame buffer that is double the size of the actual frame buffer. This is because the underlying frame buffer format (R8G8B8A8) doesn't match what we configured in the EGL initialization for unknown reasons. We have to perform an extra conversion step and move the data from the temporary frame buffer to the actual frame buffer.

```
uint8_t *tempFrameBuffer = (uint8_t*)malloc(
   LCD_FRAME_BUFFER_SIZE * 2);
GLint colorFormat;
glGetIntegerv(GL_IMPLEMENTATION_COLOR_READ_FORMAT, &
   colorFormat);
GLint colorType;
glGetIntegerv(GL_IMPLEMENTATION_COLOR_READ_TYPE, &
   colorType);
glReadPixels(0, 0, LCD_WIDTH, LCD_HEIGHT, colorFormat,
   colorType, tempFrameBuffer);
convertRGBA8888toRGB565(tempFrameBuffer, (uint16_t *)
   LCD_getFrameBuffer(), LCD_WIDTH, LCD_HEIGHT);
```

3. Send frame to the LCD using SPI.

```
...
sendCommand(COMMAND_RAMWR);
...
SPI_sendWords((uint32_t*)s_frameBuffer,
   LCD_FRAME_BUFFER_SIZE);
```

You may ignore this warning.

```
libEGL warning: egl: failed to create dri2 screen.
```

This workaround is not perfect, and there are many areas to optimize. For example, by default, double buffering does not work for the pixel buffer. The expensive pixel transfer and conversion operations may also cause a performance hit.

6 Sample output and performance

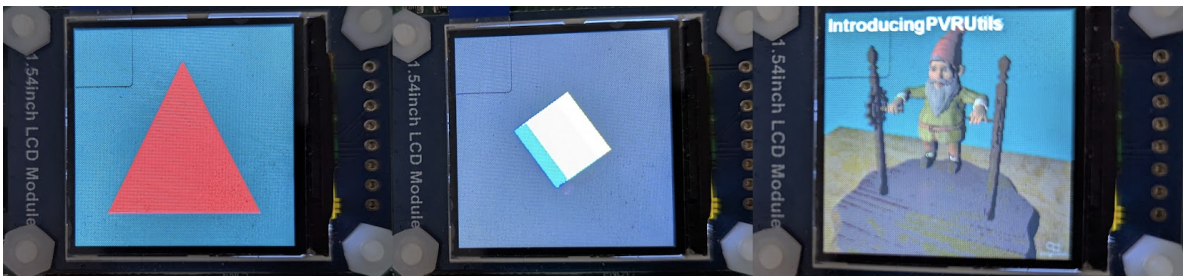

Figure 1: (a) Static Triangle, (b) Rotating cuboid, (c) 3D model

(a) and (b) are examples from the supporting files. (c) is an example from the PowerVR SDK, which is not included in the supporting files. If your setup is correct, you should see the result on your LCD and have similar performance.

Application	FPS
No rendering	43.1
Static Triangle	35.7
Rotating cuboid	33.2
3D model	26.3
3D model with animation	18.9

Table 1: 2D/ 3D Graphics Performance

7 Application development

PowerVR provided an SDK for us to develop graphical applications [5]. It is not required to use it, but it provides a framework and some useful utilities like font rendering, model loading, and an animation tool. If you want to run their example, you have to use the same EGL configuration as above and compile with the option PVR_WINDOW_SYSTEM=NullWS. You may also use some multimedia libraries like SFML and SDL, but we have not tried them yet, and you still need to use the same EGL configuration as above.

OpenGL ES is a simpler version of OpenGL, and you can directly use most OpenGL resources that you find online. The detailed specification can be found online [6]. Here are some useful resources to learn how to develop OpenGL ES applications [7][8].

8 Troubleshoot

- If your dynamically linked library fails, check if you have the following library installed.

```
$ sudo find / -name "libGL*"
...<some .so files>
$ sudo find / -name "libEGL*"
...<some .so files>
```

- If your GPU somehow does not work, you should check if you have the graphics driver and kernel modules enabled.

```
$ sudo find / -name "pvrsrvkm*"
/sys/bus/platform/drivers/pvrsrvkm
/sys/module/pvrsrvkm
/usr/lib/modules/6.1.83-ti-arm64-r63/extra/j722s/pvrsrvkm.ko
/usr/lib/modules/6.1.83-ti-arm64-r65/extra/j722s/pvrsrvkm.ko
/usr/lib/modules/6.1.83-ti-arm64-r68/extra/j722s/pvrsrvkm.ko
```

- OpenGL functions may fail silently; failure doesn't terminate the program. You have to use glGetError or eglGetError to know whether an error happened prior.

References

[1] Improved SPI guide
https://github.com/wcs3/BYAI-mcu_spi0

[2] Linux graphics stacks slides by bootlin
https://bootlin.com/doc/training/graphics/graphics-slides.pdf

[3] OpenGL Wiki
https://www.khronos.org/opengl/wiki/

[4] EGL surfaceless extension
https://registry.khronos.org/EGL/extensions/MESA/EGL_MESA_platform_surfaceless.txt

[5] PowerVR SDK
https://docs.imgtec.com/sdk-documentation/html/introduction.html

[6] OpenGL ES 3.0.6 specification
https://registry.khronos.org/OpenGL/specs/es/3.0/es_spec_3.0.pdf

[7] OpenGL ES 3.0 Quick Reference Card
https://www.khronos.org/files/opengles3-quick-reference-card.pdf

[8] OpenGL ES 3.0 Programming Guide
https://books.google.ca/books/about/OpenGL_ES_3_0_Programming_Guide.html?id=7qT0AgAAQBAJ