Computer Graphics

spring, 2013

Chapter 2
Hello Triangle: An OpenGL ES 2.0 Example

The 1st Example

- Concepts covered
  - Creating an on-screen render surface with EGL
  - Loading vertex and fragment shaders
  - Creating a program object, attaching vertex and fragment shaders, and linking a program object.
  - Setting the viewport.
  - Clearing the color buffer.
  - Rendering a simple primitive.
  - Making the contents of the color buffer visible in the EGL window surface.

Examples in the Book

- OpenGL ES 2.0 framework (Appendix D)
  - “wrapper” code framework
  - functions staring with “es”
  - Avoids global variables due to lack of supports on several mobile platforms
  - Due to the differences between platforms, APIs are NOT cross-platform (window creation, message loop, etc.)
- EGL: Built & run on AMD OpenGL ES 2.0 emulator --> PowerVR SDK seems to a nice alternative for Android platform

Setup

- Download & install ADT bundle at http://developer.android.com/sdk
- Download & install Android NDK at http://developer.android.com/tools/sdk/ndk
- Download & install OpenGL ES 2.0 SDKs for PowerVR SGX at http://www.imgtec.com/powervr/insider/sdk/KhronosOpenGLES2xSGX.asp (registration required)

Hello Triangle

- main
  - Initialize the context & ES code framework (esInitContext)
  - Create a window (esCreateWindow)
  - Compile shaders & build a program (Init)
  - Register callback functions (esRegister*Func)
  - Enter the message processing loop (esMainLoop)

Shaders

- Both vertex & fragment shaders are always required for any OpenGL ES 2.0 program
- In practice, shaders are stored in external text files
- Compiling a shader
  1. Create a shader object (glCreateShader with GL_VERTEXSHADER or GL_FRAGMENT_SHADER)
  2. Load the shader source (glShaderSource)
  3. Compile the shader (glCompileShader)
  4. Check the compile status (glGetShaderiv with GL_COMPILE_STATUS)

Program Object

- An “executable”
- Both shaders need to be attached
- Steps
  - Create a program object (glCreateProgram)
  - Attach both shaders (glAttachShader)
  - Bind vertex shader attributes to locations (glBindAttribLocation)
  - Link the program (glLinkProgram)
  - Check the link status (glGetProgramiv with GL_LINK_STATUS)
  - Use the program object before rendering in the callback function (glUseProgram)

Rendering

- A rendering callback function is registered with esRegisterDrawFunc
- Steps

1. Set the viewport (glViewport)
   - Usually done in reshape callback
2. Clear the color buffer (glClear)
   - Color buffer only (No depth buffer)
3. Use the program (glUseProgram)
4. Draw geometry (glVertexAttribPointer, glEnableVertexAttribArray, glDrawArray)
   - Usually done once in initialization
   - Geometry data in the CPU mem (Usually uploaded to the GPU mem)
5. Swap buffers for double buffering (eglSwapBuffers)

Rendering Primitives

- Position data only
- Stored in the CPU memory (usually stored in the GPU memory)
- Specify where the vertex attributes are located by glVertexAttribPointer
- Enable attribute indices by glEnableVertexAttribArray
- Draw the primitive by glDrawArrays