GL ES Guiding Principles

• Create a compact but powerful 3D rendering standard for embedded platforms
  – Leverage the best and most appropriate parts of OpenGL, without taking on legacy features
  – Extend the result to add functionality required by the embedded and mobile multimedia space
  – Be nimble and up-to-date w.r.t. (or ahead of) commercial 3D hardware and software
GL ES Version Specification Methodology

• Each version of GL ES is specified as:
  – A subset of a version of full OpenGL
  – New commands and tokens suited to embedded platforms via
    • Core additions (added to the APIs as-is)
    • Mandatory extensions (added with the OES suffix)
    • Optional extensions (just like OpenGL – must be queried)
GL ES: Parallel Tracks

- GL ES 1.x: Fixed-function pipeline implementations
- GL ES 2.x: Programmable pipeline implementations
- The two tracks move in parallel
  - Versions are backwards compatible within a track
  - But 2.x is not backwards compatible to 1.x
GL Features not in GL ES

• “Workstation” or “Heavyweight” features:
  – Selection
  – Feedback
  – Evaluators
  – Display lists
  – Attribute stacks
  – Occlusion queries

• All of these are left out of GL ES 1.x and 2.0
  – Does not apply to GL ES SC (Safety Critical)
GL Features not in GL ES

- Features that were less popular in modern GL or that map poorly to modern HW
  - Color index mode
  - Quad, quad strip and general polygon primitives
  - Texture proxies, prioritization, and residency
  - Vertex-at-a-time rendering (glBegin, glEnd)
  - Polygon mode (the mode is always GL_FILL)
  - Line and polygon stippling (can be emulated)
GL ES 1.x

• Focused on fixed-function hardware
  – Some vendors have added programmable shading extensions

• Designed for both HW and SW implementations
  – Later versions of 1.x are a bit more strongly focused on HW implementations
GL ES 1.x “Real” Datatypes

- **GLfloat**
- **GLfixed** (s15.16 format)
  - Most supported GL functions that use **GLfloat** are paired with versions (x) that accept **GLfixed**
  - This is a “core addition” to GL ES 1.x
- **GLdouble** is not supported
  - All GL functions that use **GLdoubles** are replaced with fixed (x suffix) and float (f suffix) versions
GL ES 1.x Profiles

• Common “Lite”
  – GL ES 1.x with only the fixed-point functions/data
  – No support for floating-point entrypoints or data
  – Targeted mainly at SW-only implementations

• Common
  – Contains support for both floating-point and fixed-point entrypoints and data
  – Common is what we’ll consider GL ES 1.x today
• Based on OpenGL 1.3
  – Suited for both HW and SW implementations
Geometry specification

- `glBegin` / `glEnd` are not supported
- Geometry is specified using vertex arrays
  - `glVertexPointer` (2D, 3D, 4D)
  - `glNormalPointer` (3D)
  - `glTexCoordPointer` (2D, 3D, 4D)
  - `glColorPointer` (4D!)
- Interleaved vertex data is supported
  - But not via `glInterleavedArrays`
Per-object Vertex Components

• Three “immediate mode” functions were kept
  – `glColor4f`
  – `glNormal3f`
  – `glMultiTexCoord4f`

• These specify constant, *per-object* values
  – These are specified in floating-point

• The constant values are ignored if the pointer for that component is enabled
Supported Vertex Component Formats

• All components except color:
  – GLfloat
  – GLfixed
  – GLshort
  – GLbyte

• Color:
  – GLfloat
  – GLfixed
  – GLubyte
Primitive rendering

- Indexed / non-indexed primitives supported:
  - `glDrawArrays`
  - `glDrawElements`
- Index arrays must be `GLubyte/GLushort`
- `glDrawRangeElements` is not supported
Transforms

- Most matrices supported (as stacks)
  - MODELVIEW, PROJECTION, TEXTURE
  - COLOR matrix is not supported

- Matrices are in fixed-point or float

- Most matrix loading and concatenation operations are supported
  - except the “TransposeMatrix” operations
Vertex processing state

- `glTexGen` not supported
- Lighting is supported (up to 8 lights)
- No support for separate secondary color
- Vertex normal rescaling and renormalization are supported
- User clipping planes are not supported
Vertex Lighting

• `glMaterial` must be `FRONT_AND_BACK`
  – Front and back materials must be the same

• `COLOR_MATERIAL` mode is supported
  – But only in `AMBIENT_AND_DIFFUSE` mode

• `glLightModel` only supports
  – `GL_AMBIENT`
  – `GL_LIGHT_MODEL_TWO_SIDE`
Texturing

• Only 2D textures are supported in GL ES 1.0
  – 1D and 3D textures are not supported
  – Cube map textures are out (of GL ES 1.0 and 1.1)
• Mipmapping is supported
  – Including all filtering modes
  – Explicit texture LOD control not supported
• Borders, clamp-to-border addressing are out
Texture Image Formats

• Only RGB, RGBA, LUMINANCE, ALPHA and LUMINANCE_ALPHA formats are supported

• Only UNSIGNED_BYTE component types are supported for the above types, with the exception that it supports the following 16-bpp formats

  – UNSIGNED_SHORT_5_6_5 (RGB)
  – UNSIGNED_SHORT_4_4_4_4 (RGBA)
  – UNSIGNED_SHORT_5_5_5_1 (RGBA)
Paletted Textures

- Supported via a required extension
- Supports 4- or 8-bit indices (16/256 entries)
- Supports palette entry formats that correspond to the supported RGB(A) texture formats
- Palette entries must be specified with each texture data call
  - No shared palettes
  - Thus no palette animation
Texture Environments

- Multiple texture stages supported, *but not required*
- Most texture environment modes supported
  - `GL_REPLACE`, `GL_MODULATE`
  - `GL_ADD`
  - `GL_BLEND`, `GL_DECAL`
- But not `GL_COMBINE`, thus no
  - `COMBINE_RGB`, `COMBINE_ALPHA`
  - `SOURCE_{012}_RGB`, `SOURCE_{012}_ALPHA`
Fragment processing state

- Most GL fragment processing state is supported in GL ES 1.0
- Depth and stencil ops are available
  - But implementations are *not* required to support depth or stencil buffers
- Multi-sample antialiasing is supported
  - But implementations need not support a multisample buffer
- Scissoring is supported and required
Pixel Blending

• Only the additive blending equation is supported
  – There is no support for the imaging subset
  – So no `glBlendEquation` or `glBlendColor`

• But all of the various blending *functions* are available
  – `GL_SRC_COLOR`
  – `GL_DEST_COLOR`
  – etc
Whole-Framebuffer Ops

- Multiple drawing buffers are not supported
  - No stereo, aux buffers, etc
  - So `glDrawBuffer` is not supported/included

- Accumulation buffers are not supported

- Color and depth masking is supported

- Necessary clear operations are supported
  - COLOR, DEPTH, STENCIL
Raster/Pixel Ops

- Most pixel, bitmap, rectangle operations are \textit{not} in GL ES
  - No \texttt{glDrawPixels}, \texttt{glCopyPixels}, \texttt{glPixelZoom}, etc

- \texttt{glReadPixels} is supported
  - but format conversions are very limited
  - \texttt{glPixelStorei} is supported in a limited way (packing and unpacking stride)
Dynamic Render State Query

- Most dynamic render state queries have been removed
- Applications must shadow render state that they would otherwise have queried
GL ES 1.1

- Based on OpenGL 1.5
- Focuses more on HW implementations
  - Adds paths to better feed HW vertex processing
  - Adds more powerful texturing environment
Geometry Specification

• Includes vertex and index buffer objects (from GL 1.5)
  – Based on a cut-down version of GL VBOs
• GL VBOs’ memory mapping not required
  – No glMapBuffer / glUnmapBuffer
• VBOs are important on handheld devices, which often have slow/narrow system busses
Rendering Primitives

• Adds required support for point sprites
  – Also requires point size array support
  – Put together, these make it more likely (although not universal) that applications can avoid having to use tri-based “screen-aligned quad” particle systems

• \texttt{glMultiDraw}* are not supported
  – Some vendors support them via extensions
Transformations and Vertex Processing

- Adds required support for at least one user-supplied clipping plane
  - Implementations can support/expose more

- Optional extension OES_matrix_palette adds:
  - Array of at least 9 matrices per object
    - single array – not a stack
  - Arrays of at least 3 matrix indices/weights per-vertex
  - Allows for HW-based skin deformations
Texturing / Tex Environment

- Adds requirement of at least two texture stages
- Includes *all* the GL 1.5 modes except crossbar
  - E.g. *DOT3* support is included
- Adds automatic mipmap generation (GL 1.5)
  - Useful for rendered textures
- No support for mirrored texture repeat addressing
- No support for depth textures
Other additions/subtractions

- The ability to query many of the dynamic render state values is added in GL ES 1.1
  - Mainly added for “layered” applications needing to implement their own render state push and pop
  - See the spec for a list of supported states
  - This is not push/pop

- The new GL 1.4 stencil ops were not added (INC_WRAP/DEC_WRAP)
GL ES 1.1 Extension Pack and Proposed GL ES 1.2

- GL ES 1.1 includes a set of optional extensions that, if supported as a set, are called the "GL ES 1.1 Extension Pack"
- Implementations that support the entire pack are *likely* to be GL ES 1.2-ready as well
  - We’ll discuss them as one and the same
Extended Matrix Palettes

• OES\_extended\_matrix\_palette

• Makes palette skinning more useful and (potentially) efficient by increasing minima
  – At least 32 matrices entries in a palette
  – At least 4 matrices per vertex

• In general, this allows more skinned geometry to be rendered in a single draw call
Cube Maps and Texture Coordinate Generation

- **OES_texture_cube_map**
  - Adds cube map support

- Two modes of texgen are enabled
  - `REFLECTION_MAP`
  - `NORMAL_MAP`

- There are several restrictions on the cube maps
  - Each face of a cube map must be square
  - All faces of a cube map must be the same size
Texturing and Texture Environment

- **OES_texture_mirrored_repeat**
  - adds mirrored texture wrapping (GL 1.4)

- **OES_texture_env_crossbar**
  - Adds support for the texture crossbar (GL 1.4)
  - Allows the use of the color from any texture unit as a source to a combine operation
Per-Fragment Operations

• **OES_blend_*** add more blending equations
  - \( C = C_S S - C_D D \) ([GL\_FUNC\_SUBTRACT](#))
  - \( C = C_D S - C_S D \) ([GL\_FUNC\_REVERSE\_SUBTRACT](#))
  - Also adds support for independent RGB and Alpha blending equations/functions

• **OES_stencil_wrap** adds the GL 1.4 stencil actions that were not in GL ES 1.1
  - DEC.WRAP
  - INC.WRAP
Framebuffer Objects

- **OES_framebuffer_object** is a subset of GL’s **EXT_framebuffer_object**
  - Allows for direct render-to-texture, including rendering to cube map faces
  - An additional optional extension allows rendering to mipmap levels (**OES_fbo_render_mipmap**)

- Avoids the need for a context switch or a buffer copy when rendering to texture
EGL

- EGL is GL ES’s native platform interface
- It is designed to replace the per-platform systems used for GL
  - e.g. WGL on MS Windows, GLX on X Windows
- It does *not* implement or replace the native platform’s windowing system or native graphics system
EGL and other Khronos APIs

- EGL also allows for resource sharing and synchronization of rendering between multiple Khronos APIs
- This is becoming more and more important
What EGL Manages

- Display devices
- Display and surface configurations
  - Pixel format, depth/stencil, multisample, etc
- Rendering contexts
  - E.g. GL ES contexts
- Rendering surfaces
  - Onscreen (“Window”), Pbuffer, Pixmap
EGL: Devices and Contexts

- Display initialization/shutdown
  - eglGetDisplay
  - eglInitialize, eglTerminate
- Configuration management
  - eglGetConfigs, eglChooseConfig
- GL ES context creation/sharing/destruction
  - eglCreateContext, eglDestroyContext
  - Like GLX (and unlike WGL), creation sets up sharing
EGL: Surfaces

- Render surface creation/destruction
  - eglCreateWindowSurface
  - eglCreatePbufferSurface
  - eglCreatePixmapSurface
  - eglDestroySurface
EGL: Rendering Targets

- Context/Surface selection for rendering
  `eglMakeCurrent`
- Buffer swapping (window surfaces only)
  `eglSwapBuffers`
  `eglSwapInterval`
- Binding surfaces as textures (optional)
  `eglBindTexImage`
  `eglReleaseTexImage`
What EGL Supports: Misc

- Extension function pointer query (EGL and GL ES)
  ```c
  eglGetProcAddress
  ```
  - But the GL ES extension string is still queried from GL ES, not EGL

- Cross-API synchronization
  ```c
  eglWaitGL
  eglWaitNative
  ```
GL ES 2.0

• Designed to feed programmable-pipeline 3D hardware
• Based on OpenGL 2.0, but is shaders-only
• Really two parts
  – The APIs (discussed here)
  – The shading language (next session)
GL ES 2.0 and GL ES 1.x

- GL ES 2.0 is not backwards compatible with GL ES 1.x!
- This is a part of the more general GL ES goal of not carrying around API entrypoints solely for the sake of backwards compatibility.
GL ES 2.0: The Big Changes

- The GL 2.0 fixed-function vertex transform and lighting pipeline is *not* supported
  - GL ES 2.0 only supports vertex shaders/programs

- The GL 2.0 fixed-function texture pipeline is *not* supported
  - GL ES 2.0 only supports fragment shaders/programs

- *Many* GL 2.0 entrypoints are not in GL ES 2.0
  - Leads to a simple, lean, uncomplicated API spec
GL ES 2.0 Pipeline

API

Primitive Processing

Vertices

Vertex Buffer Objects

Triangles/Lines/Points

VerteShader

Primitive Assembly

Vertices

Rasterizer

Fragment Shader

Depth Stencil

Color Buffer Blend

Dither

Frame Buffer
Supported Types

- Unlike GL ES 1.x, 2.0 does not support fixed-point versions of command parameters
  - To simplify the APIs, only floating-point are supported for most immediate “Real” parameters
- Fixed-point data is still supported for vertex attribute arrays
Geometry Specification

- All geometry is specified generally, using:
  
  ```c
  glVertexAttrib{1234}f[v]
  glVertexAttribPointer
  ```

- Thus, no:
  
  ```c
  glVertexPointer, glNormalPointer, 
glColorPointer
  ```

- Implementations must support at least 8 of these 4D attribute vectors
  (MAX_VERTEX_ATTRIBS)
Vertex/Primitive Attributes

- `glVertexAttribPointer` supports all types; fixed, float, byte, short, etc
  - 16-bit float components (s5e10m) supported by an optional extension (OES_vertex_half_float)
- `glVertexAttribf[v]` supports float only
- `glDrawElements` supports 8/16-bit indices
  - 32-bit are added as an optional extension (OES_element_index_uint)
Passing Data to Vertex Shaders

• “Global” data is passed to the vertex shader as Uniform values
  – Same as the GL 2.0 shader pipeline
  – All major GL 2.0 uniform entrypoints supported
  – But `glUniformMatrix` cannot transpose

• GL ES 2.0 requires support for at least 384 Uniform `floats` within vertex shaders
Transform & Lighting Functions

• Fixed-function T&L functions are removed:
  – GL transform/matrix stack functions are gone
    • All transforms passed to the shaders as uniforms
    • The app is responsible for managing transforms
  – GL 2.0 lighting/material functions are gone
    • Light info and material info is placed in uniforms
    • Lighting is done in shader code
  – TexGen and user clipping planes are also gone
Overall Vertex Shader Block

- **Uniforms**
- **Textures (opt)**
- **Vertex Shader**
  - **Attributes**:
    - Attribute 0
    - Attribute 1
    - Attribute 2
    - Attribute 3
    - Attribute 4
    - Attribute 5
    - Attribute 6
    - Attribute 7
  - **Varying Variables**:
    - Varying 0
    - Varying 1
    - Varying 2
    - Varying 3
    - Varying 4
    - Varying 5
    - Varying 6
    - Varying 7
  - **Temporary Variables**:
    - `gl_Position`
    - `gl_PointSize`
Viewport Transformations

- `glViewport` and `glDepthRange` are supported
- These happen post-vertex shader and are still fixed-function operations
Overall Fragment Shader Block

- Uniforms
- Textures
- Temporary variables
- gl_FragCoord
- gl_FrontFacing
- Varying 0
- Varying 1
- Varying 2
- Varying 3
- Varying 4
- Varying 5
- Varying 6
- Varying 7
- gl_FragColor

Fragment Shader
Textures

- 1D, 3D and depth textures are not required
  - An optional extension adds 3D textures (OES_texture_3D)

- Non-power-of-two textures have restrictions
  - Mipmapping need not be supported
  - Only clamp addressing needs to be supported
  - Both of these restrictions can be removed by an optional extension (OES_texture_npot)
More Optional Texture Features

- **OES_texture_[half_]float**
  - Support 16- or 32-bit floating-point textures with only NEAREST texture filtering (w/ mipmapping)

- **OES_texture_[half_]float_linear**
  - As above, but add linear texture filtering

- **OES_compressed_ETC1_RGB8_texture**
  - Adds ETC/iPACKMAN compressed texture format
Fragment APIs

- The GL 2.0 pipeline stages and APIs:

- Are all replaced in GL ES 2.0 by:

- Depth (16b) and stencil (8b) buffers are required
- Pixel blending is similar to GL ES 1.2
- But LogicOps are out
Final Programming Model

- **Attributes** (~8 4-vectors)
- **Vertex Uniforms** (~384 floats)
- **Vertex Shader**
  - **Primitive Assembly & Rasterize**
  - **Varyings** (~8 4-vectors)
- **Fragment Uniforms** (~64 floats)
- **Fragment Shader**
  - **Per-Sample Operations**

### Attributes
- (~8 4-vectors)

### Vertex Uniforms
- (~384 floats)

### Varyings
- (~8 4-vectors)

### Fragment Uniforms
- (~64 floats)
Whole Framebuffer APIs

• Roughly equivalent to GL ES 1.2
  – Color masking is supported
  – `glReadPixels` is supported
  – `glCopyPixels` is not supported
  – Multiple draw buffers are not supported
  – Accumulation buffers are not supported

• Support for `OES_framebuffer_object` is required
Loading and Using Shaders

- GL 2.0 requires a shader compiler in the driver; it only loads shader source code
  - This could be too slow or heavyweight for GL ES
- GL ES 2.0 requires implementations to support at least one of the following:
  - Load source at runtime (GL 2.0 model)
  - Load platform-specific precompiled binary shaders
    - Possibly even a pre-linked vertex/fragment pair
Closing

• GL ES 1.x for fixed function implementations
• GL ES 2.0 for programmable implementations
• The specs at www.khronos.org are amazingly readable!
• www.khronos.org also includes a wealth of programming resources for OpenGL ES and all other Khronos APIs