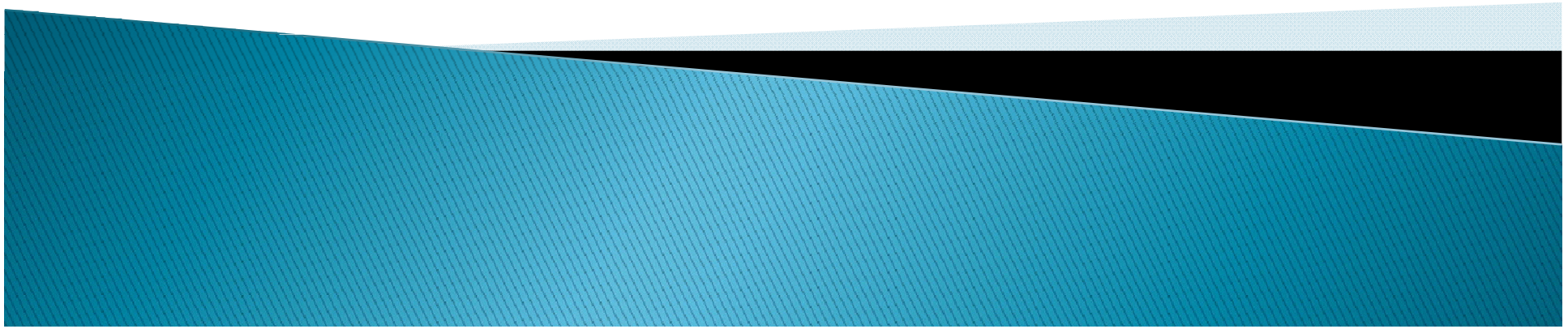
# OpenGL for Embedded Systems
## (OpenGL ES )

2008.07.21
Sungjin, Son



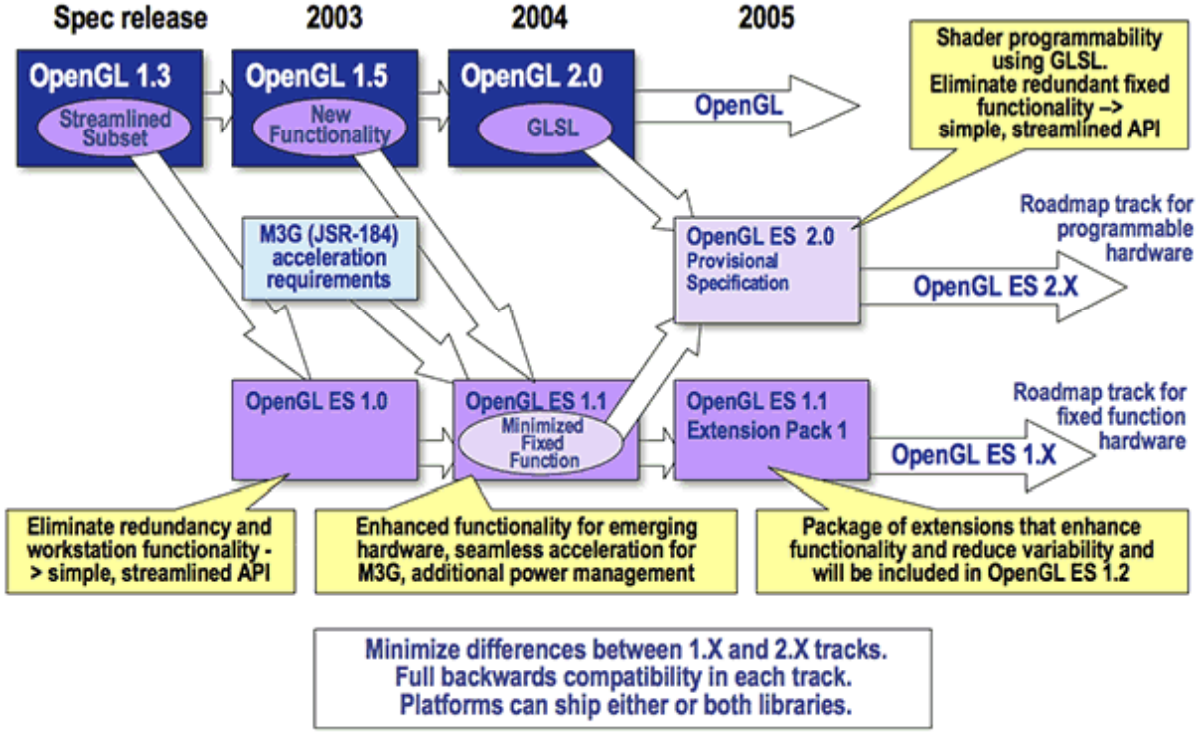
# OpenGL ES (OpenGL for Embedded Systems)

- OpenGL ES (OpenGL for Embedded Systems)
  - OpenGL® ES is a royalty-free, cross-platform API for full-function 2D and 3D graphics on embedded systems
  - It includes consoles, phones, appliances and vehicles on Symbian, Brew and etc
  - It consists of well-defined subsets of desktop OpenGL, creating a flexible and powerful low-level interface between software and graphics acceleration.
  - It is defined and promoted by the Khronos Group(http://www.khronos.org)

- Developer Advantages
  - Industry Standard and Royalty Free
  - Small footprint & low power consumption
  - Seamless transition from software to hardware rendering
  - Extensible & Evolving
  - Easy to use(Based on OpenGL)
  - Well-documented

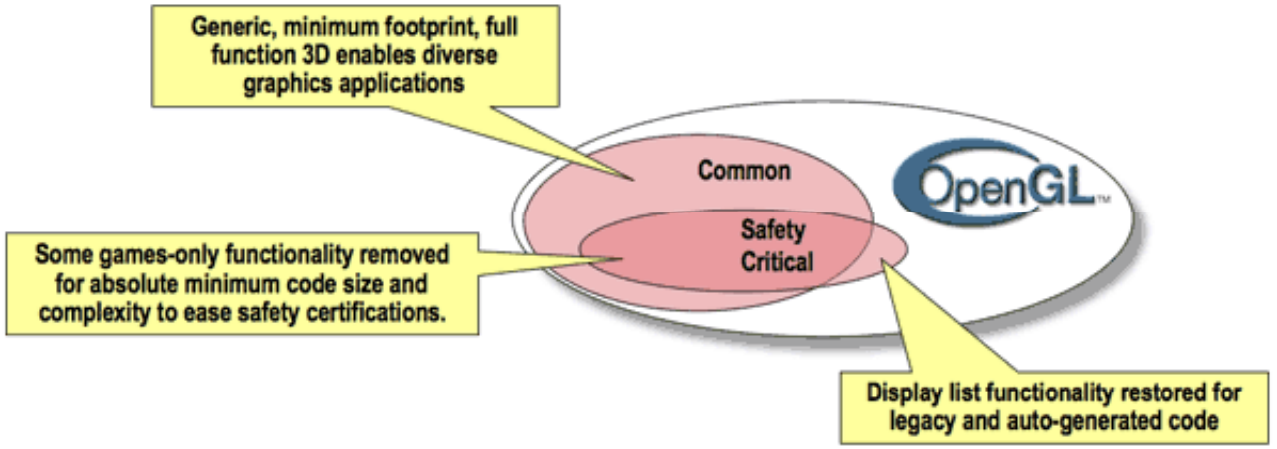
# OpenGL ES History

- OpenGL ES 1.0
  - Basic 3D functionality
- OpenGL ES 1.1+
  - Comprehensive set of fixed-function hardware
- OpenGL ES 2.0
  - Full programmable 3D graphics
    - Vertex and Fragment shader



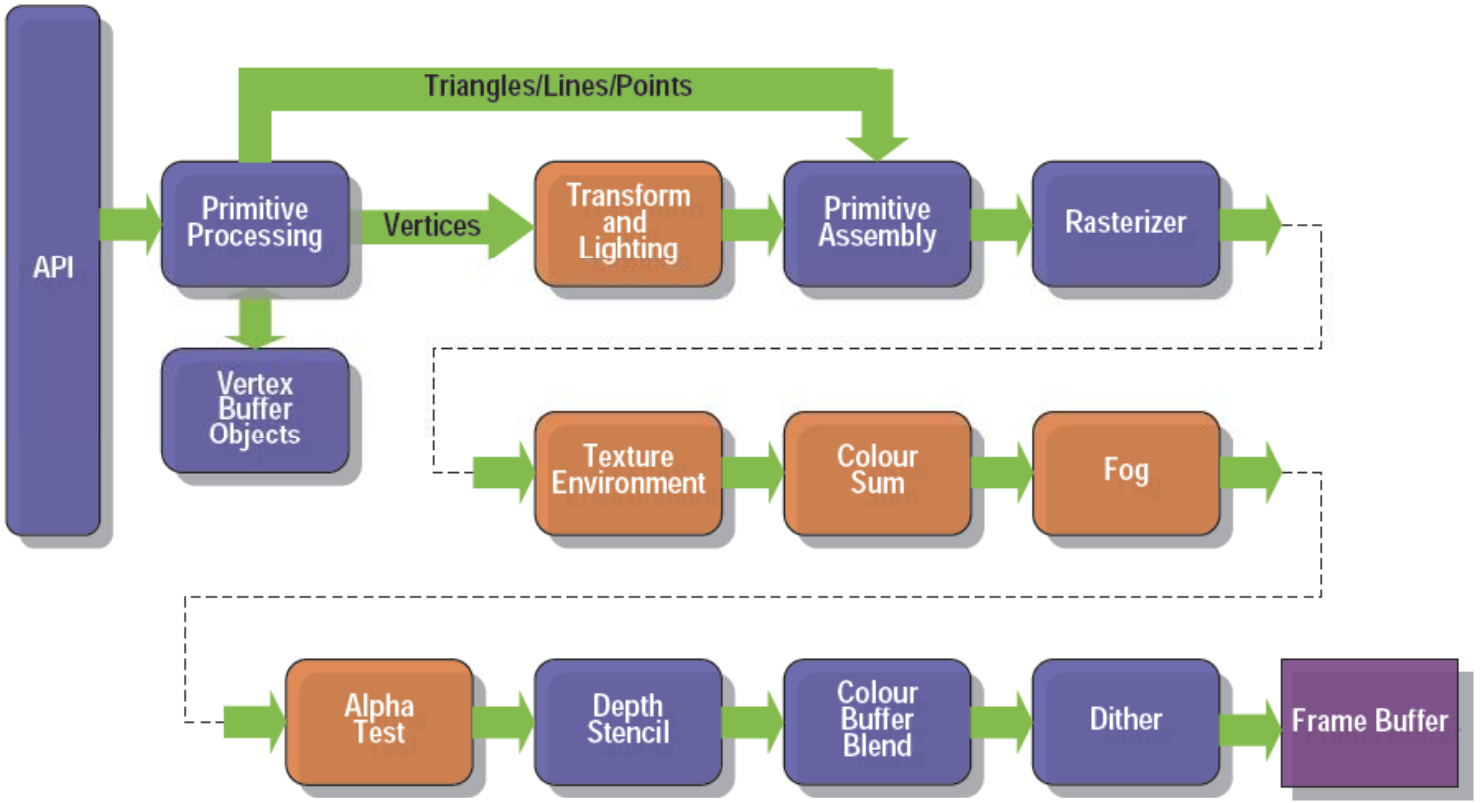
# OpenGL Framework

- The Common Profile
  - Intended for consumer entertainment and related devices
  - Addresses the broadest range of the market including support for platforms with varying capability.
- The Safety Critical Profile
  - Intended for consumer and industrial applications
  - Reliability and certifiability are the primary constraints.
- Extensions
  - Include extensions that add new features to the implementation.
- Native Platform Graphics Interface Layer – EGL
  - Includes a specification of a common platform interface layer, called EGL.
  - This layer is platform independent and may optionally



# OpenGL ES 1.x

- Fixed Function Pipeline



# OpenGL ES 1.x

- Buffer objects
  - Vertex Arrays
  - Used to render primitives in OpenGL ES 1.0
  - Vertex array data stored in client memory
  - Need to allow vertex data to be cached in graphics memory
- Auto mipmap generation
- Enhanced texture processing
  - For effects such as bump-mapping and per-pixel lighting.
  - Supports 2D textures only
- Vertex skinning functionality
  - Using the OES_matix_palette extension
  - A set of matrix indices & weights per vertex.
    - # of matrices / vertex can be queried using glGetIntegerv
    - Minimum 3 matrices / vertex

# OpenGL ES 1.x

- User-defined clip planes
  - Useful for portal culling algorithms
  - Support a minimum of one user clip plane
- Enhanced point sprites and point sprite arrays
  - Accelerate rendering of particle effects
  - Render particles as points instead of quads
- Static and Dynamic state
  - Queries are supported for static and dynamic state explicitly supported in the profile.
  - The supported GL state queries can be categorized into simple queries, enumerated queries, t exture queries, pointer and string queries, and buffer object queries.
- Draw Texture
  - Render pixel rectangles using texture units
  - Useful for fast rendering of sprites, bitmapped font
  - glyphs, & 2D framing elements in games

# OpenGL ES 1.x Extention

- Cube Maps
  - Accurate real-time reflections in handheld 3D games
- Texture Environment Crossbar
  - Adds the capability to use the texture color from other texture units as sources to the COMBIN E environment function.
  - OpenGL ES 1.1
    - Use the color from the current texture unit as a source.
  - Extension
    - Use the color from any texture unit as a source.
- Mirrored Texture Addressing
  - Set of texture wrap modes to include a mode that effectively uses a texture map
  - GL_MIRRORED_REPEAT

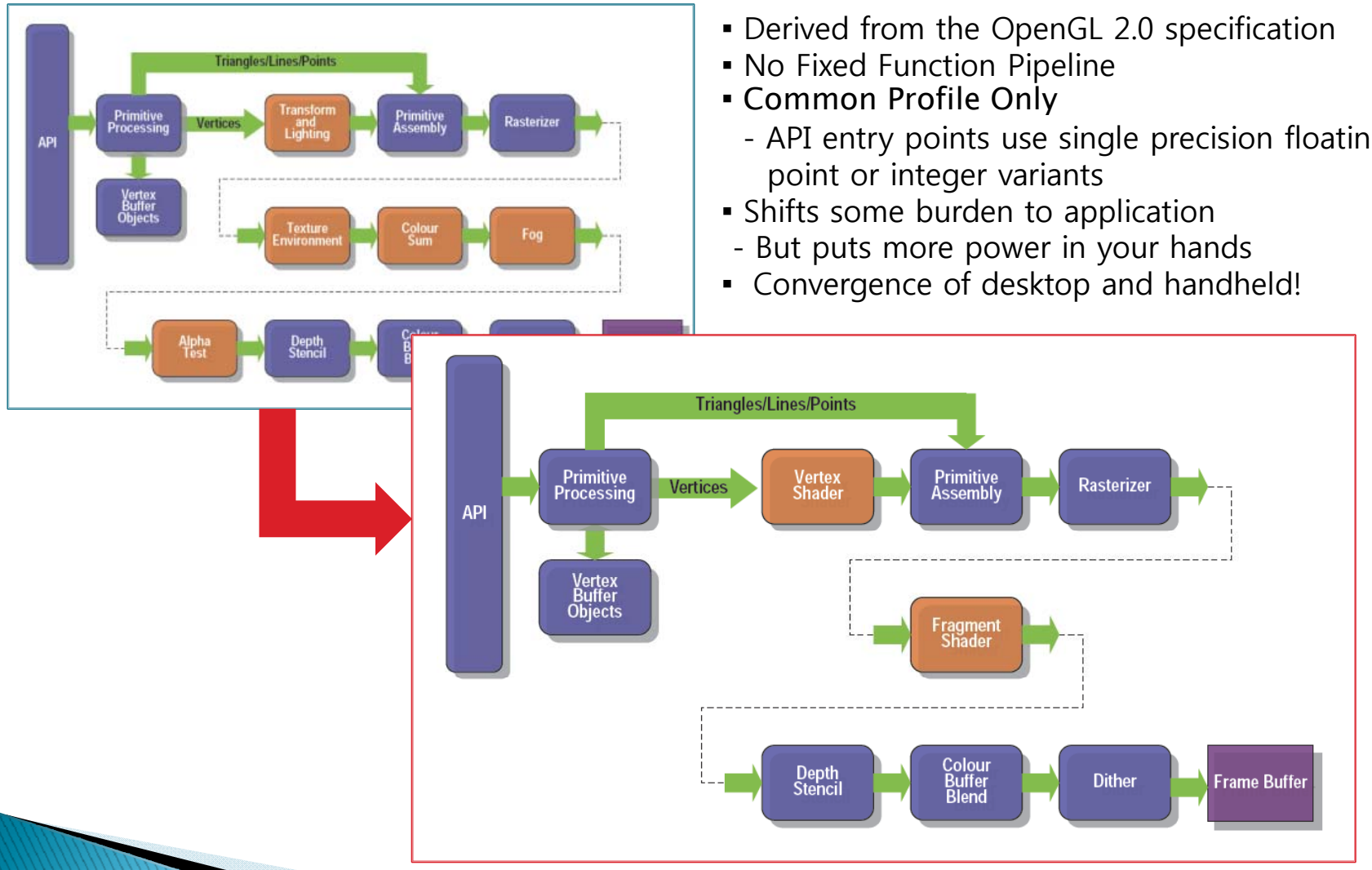
# OpenGL ES 1.1 Extention

- Blending Extensions
  - Independent setting of the RGB and alpha blend factors for blend operations
  - GL_FUNC_SUBTRACT, GL_FUNC_REVERSE_SUBTRACT
- Stencil Extensions
  - GL_INCR_WRAP and GL_DECR_WRAP
- Extended Matrix Palette
  - OpenGL ES 1.1 recommends
    - Matrix palette size = 9 matrices and 3 of matrices / vertex
  - ES_extended_matrix_palettet
    - Matrix palette size = 32 matrices and 4 of matrices / vertex
- Framebuffer Objects

# OpenGL ES 2.x

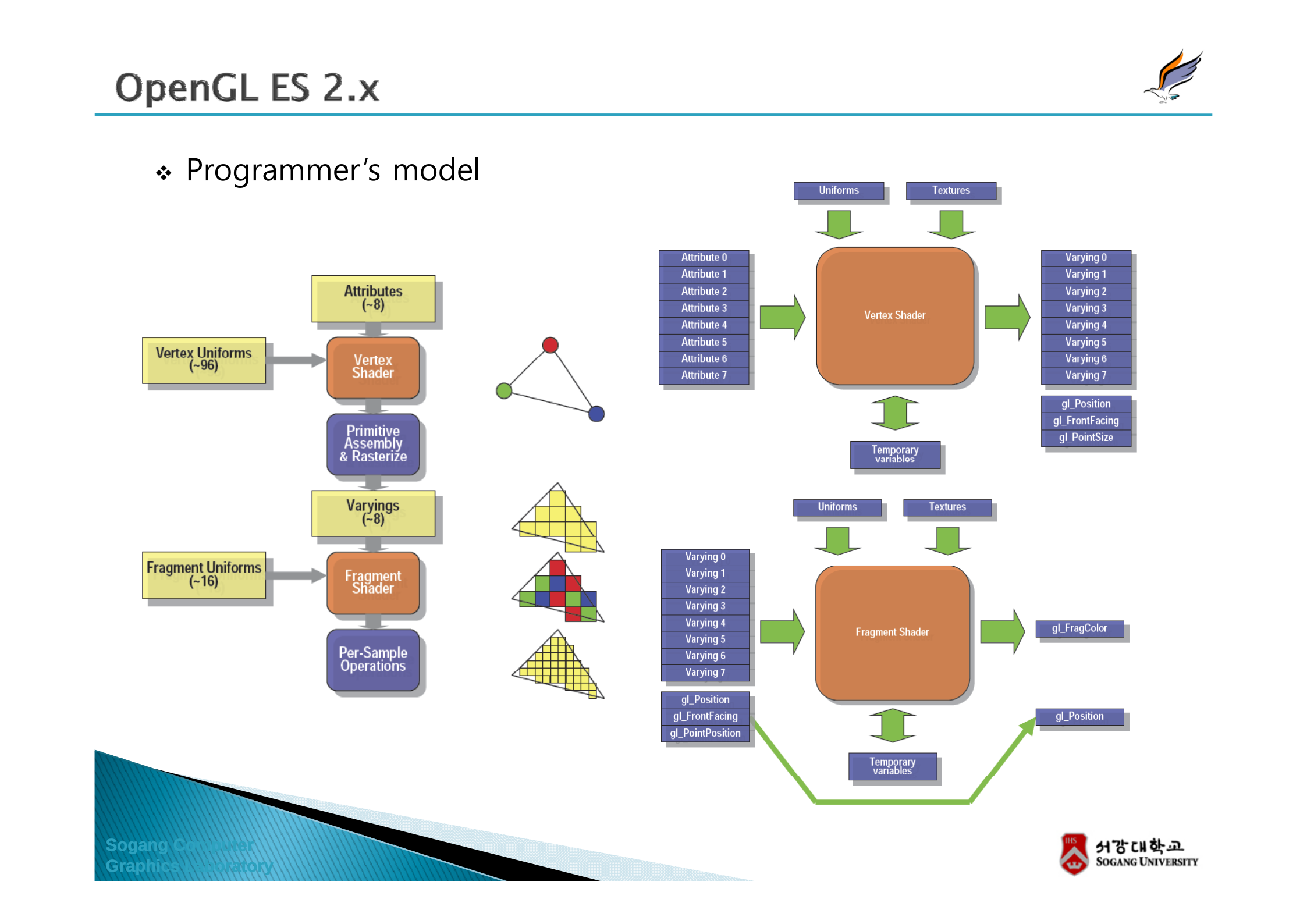


❖ Programmable Pipeline




- Derived from the OpenGL 2.0 specification
- No Fixed Function Pipeline
- **Common Profile Only**
  - API entry points use single precision floating point or integer variants
- Shifts some burden to application
  - But puts more power in your hands
- Convergence of desktop and handheld!

# OpenGL ES 2.x



- Programmer's model

# OpenGL ES 2.x : GLSL ES

## Vertex Shader functions

- The vertex shader can do:
  - Transformation of position using model-view and projection matrices
  - Transformation of normals, including renormalization
  - Texture coordinate generation and transformation
  - Per-vertex lighting
  - Calculation of values for lighting per pixel
- The vertex shader cannot do:
  - Anything that requires information from more than one vertex
  - Anything that depends on connectivity.
  - Any triangle operations (e.g. clipping, culling)
  - Access color buffer

## Fragment Shader Functions

- The fragment shader can do:
  - Texture blending,  Fog, Alpha testing, Dependent textures
  - Pixel discard, Bump and environment mapping
- The fragment shader cannot do:
  - Blending with color buffer, ROP operations
  - Depth or stencil tests, Write depth

# OpenGL ES 2.x – What's in

- Vertex Data : Point, Line, Triangle.
  - Specified using glVertexAttribPointer
  - Vertex Data Formats
    - All base GL data types, Fixed, and Half float
  - Vertex Buffer Objects
- Texturing
  - Addressing modes
    - Repeat, clamp to edge, mirrored repeat
  - Half-float and float texture formats
  - Cube-maps
- Per-Fragment Operations
  - Depth, Stencil tests same as OpenGL 2.0
  - Blending similar except
    - GL_MIN, GL_MAX functions are not supported
- State Queries
  - Exhaustive set of static and dynamic state can be queried.

# OpenGL ES 2.x – What's in

- Shaders - Two models supported
  - Online compile – OES_shader_source
    - Shaders compiled using glCompileShader
    - New call "glReleaseShaderCompilerOES" added to allow application to tell the GL that the shader compiler resources can be released
  - Offline compile – OES_shader_binary
    - Binaries loaded using glShaderBinaryOES
    - Load individual shader binaries or a binary that contains an optimized vertex / fragment shader pair
  - No default model specified
    - Application must query to determine which method is supported.

- Shader Precision Formats
  - Vertex Shader
    - Must support single precision FP
  - Fragment Shader
    - No default precision specified
    - Must support a minimum of 16 bit FP(5 bits of exponent, 10 bits of mantissa)
  - Precision Qualifiers
    - Used to specify precision of data : lowp, mediump, highp
  - precision <precision-qualifer> <type> statement

# OpenGL ES 2.x

- Precision Qualifiers
  - Lowp float
    - Typically implemented by fixed point sign + 1.8 fixed point.
    - Range is -2.0 < x < 2.0
    - Resolution 1/256
    - Use for simple colour blending
  - Mediumpfloat
    - Typically implemented by sign + 5.10 floating point
    - -16384 < x < 16384
    - Resolution 1 part in 1024
    - Use for HDR blending, some texture coordinate calculations
  - Highpfloat
    - Typically implemented by 24 bit float (16 bits of mantissa)
    - Use of texture coordinate calculation : environment mapping
  - single precision
    - Not explicit in GLSL but usually available in the vertex shader

# OpenGL ES 2.x – What's in

- OpenGL ES Shading Language
  - Built-in minimum constants
    - gl_MaxVertexAttribs = 8
    - gl_MaxVertexUniformComponents = 384 floats
    - gl_MaxVaryingFloats = 32
    - gl_MaxVertexTextureImageUnits = 0
    - gl_MaxCombinedTextureImageUnits = 2
    - gl_MaxTextureImageUnits = 2
    - gl_MaxFragmentUniformComponents = 64 floats
    - gl_MaxDrawBuffers = 1

# OpenGL ES 2.x – What's out

- Enable/Disable(MULTISAMPLE)
  - Selected using appropriate EGLconfig
- Anti-aliased lines
- Points and anti-aliased points
  - Only point sprites supported
- Coordinate Transforms & Matrix Stack
- User Clip Planes
- Depth texture formats and comparison mode
- Occlusion queries

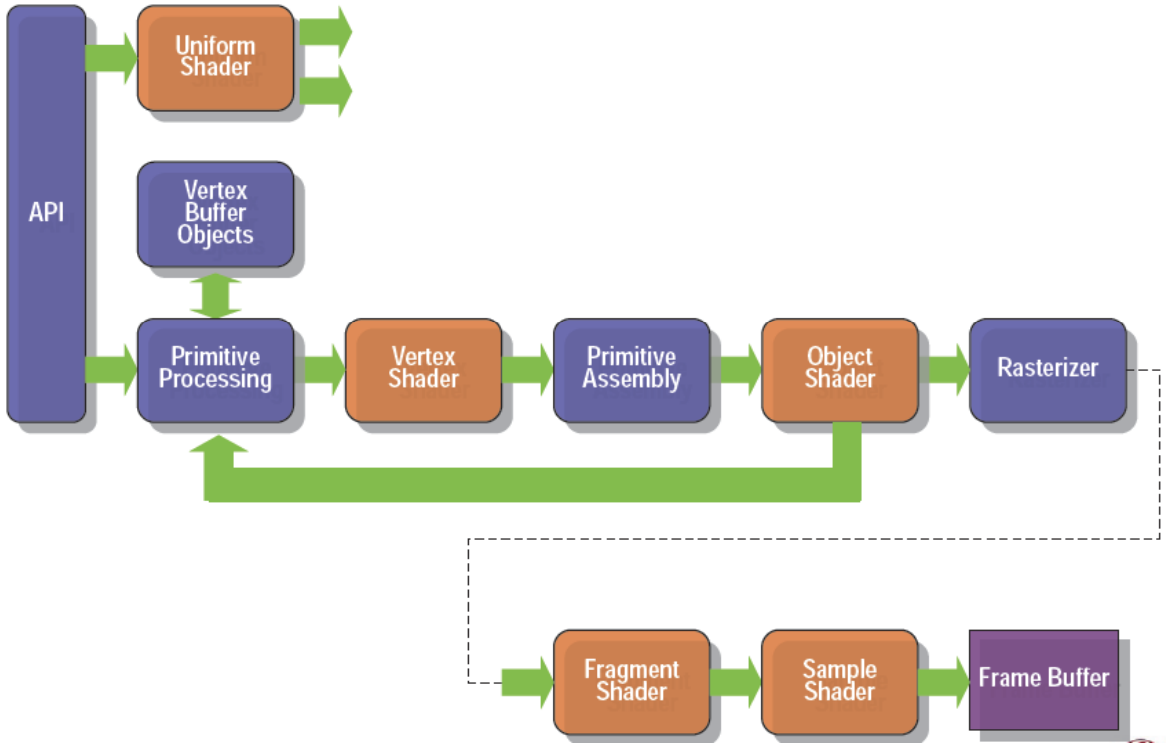
# OpenGL ES 2.x – What's optional

- MapBuffer/UnmapBuffer
- 3D textures
- Non-power of 2 textures
- With support for all addressing modes
- With mip-mapping
- FP16 vertex attribute data
- FP16 and FP32 textures

# OpenGL ES 2.x On going.

- Uniform Shaders
- Object (Geometry) Shaders
  - Programmable tessellation
  - Higher order surfaces
  - Procedural geometry
  - Possibility of accelerating many more algorithms e.g. shadows, occlusion culling

- Future ES Pipeline



# Example OpenGL ES

- The RSX® Graphics Processor : Sony PS3
  - Based on a high end NVidia chip
    - Fully programmable pipeline: shader model 3.0
    - Floating point render targets
    - Hardware anti-aliasing ( 2x, 4x )
    - 256 MB of dedicated video memory
  - PULL from the main memory at 20 GB/s
  - HD Ready (720p/1080p)
    - 720p = 921 600 pixels
    - 1080p = 2 073 600 pixels

    ➔ a high end GPU adapted to work with the Cell Processor and HD displays

# Example OpenGL ES

- PSGL : the high level graphics API, modern GPU extensions
  - Needed a standard : practical and extensible
    → the choice was OpenGL ES 1.0
  - Why not a subset of OpenGL ?
    - Mainly needed conformance tests
  - Benefits :
    - pipeline state management, Vertex arrays, Texture management
    - Bonus: Fixed pipeline : Only ~20 entry points for fixed pipeline
    - Fog, light, material, texenv
  - Inconvenience:
    - Fixed point functions
    - No shaders : needed to be added
  - OpenGL ES 1.1 : VBO, FBO, PBO, Cube Map, texgen
  - Primitives : Quads, Quads_strips, primitive restart, Instancing
  - Queries and Conditional Rendering
  - More data types : half_float
  - Textures
    - Floating point textures, DXT, 3D, non power of 2,
    - Anisotropic filtering, Min/Max LOD, LOD Bias
    - Depth textures,
    - Gamma correction,
    - Vertex Texture

- PSGL: PS3 specific extensions
  - Synchronizations:
    - Wait on or check GPU progress
    - Make the GPU wait on another GPU event or on PPU
    - Provide sync APIs for PPU and for SPU
  - Memory usage hints
    - For texture, VBO, PBO, render-targets
  - PPU specific extensions:
    - Embedded system : PPU usage needs to be limited, some extensions are added to decrease the PPU load for some existing features:
      - Ex: Attribute set
  - CG : high level shader language
    - Support Cg 1.5
    - PS3 specific compiler
    - Mostly compatible with other languages like HLSL
    - Tools : FX composer for PS3
  - CG : runtime
    - Direct access to shader engine registers or via CG parameter
    - Shared and unshared parameters
    - CG FX runtime : techniques, render states, textures