
GameDevelopers Conference
MARCH 20-24
SAN JOSE, CALIFORNIA
WHAT'S NEXT
GDC:06
www.gdconf.com
GAME DEVELOPERS CHOICE AWARDS
INDEPENDENT GAMES FESTIVAL
GDC MOBILE
SERIOUS GAMES SUMMIT
GAME CONNECTION
CMP

OpenGL ES 1.1+ and ES 2.0

Presented by
Aaftab Munshi

GameDevelopers Conference

Agenda

- OpenGL ES 1.1 Specification
- OpenGL ES 1.1 Extension Pack
- OpenGL ES 2.0 Specification

OpenGL ES 1.1 - Features

- Buffer Objects
- Texture Enhancements
- Matrix Palette
- User Clip Planes
- Point Sprites
- State Queries
- Draw Texture

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
- Vertex Buffer objects
  - Allow caching of vertex data
  - Can be used to store
    - vertex array and element index data
  - Limitations
    - Vertex data cannot be read back
    - Cannot map the vertex data into client memory
      - MapBuffer, UnmapBuffer

Using Buffer Objects

- Vertex Array Example

```
void Draw_Arrays()
{
    // Enable client states needed for our vertex format
    glEnableClientState( GL_VERTEX_ARRAY);
    glEnableClientState( GL_TEXTURE_COORD_ARRAY);
    glEnableClientState( GL_COLOR_ARRAY);

    // Set up glPointers
    glVertexPointer( 3, GL_SHORT, sizeof( tVertex),
                     (void*) &my_vertex_data[0].x );

    glTexCoordPointer( 2, GL_SHORT, sizeof( tVertex),
                       (void*) &my_vertex_data[0].u );

    glColorPointer( 4, GL_UNSIGNED_BYTE, sizeof( tVertex),
                    (void*) &my_vertex_data[0].color );

   // Draw the vertex array as an indexed triangle list
   glDrawElements(
       GL_TRIANGLES,
       my_index_count,
       GL_UNSIGNED_SHORT,
       my_index_data
   );
}
```

Using Buffer Objects

- Vertex Buffer Objects Example

```
void Draw_Buffers()
{
    // Bind vertex + index buffers
    glBindBuffer( GL_ARRAY_BUFFER, vertex_buffer);
    glBindBuffer( GL_ELEMENT_ARRAY_BUFFER, index_buffer);

    // Enable client states needed for our vertex format
    glEnableClientState( GL_VERTEX_ARRAY);
    glEnableClientState( GL_TEXTURE_COORD_ARRAY);
    glEnableClientState( GL_COLOR_ARRAY);

    // Set up glPointers
    glVertexPointer( 3, GL_SHORT, sizeof(tVertex), 0);
    glTexCoordPointer( 2, GL_SHORT, sizeof(tVertex), 8);
    glColorPointer( 4, GL_UNSIGNED_BYTE, sizeof( tVertex), 12);

    // Draw the vertex buffer as an indexed triangle list
    glDrawElements(
        GL_TRIANGLES,
        my_index_count,
        GL_UNSIGNED_SHORT,
        0,
    );
}
```

Texture Enhancements

- Minimum of 2 texture units is required
- glTexImage
  - supports 2D textures only
  - "internal format" must match "format"
  - "type" used to indicate texel format
- Addressing Modes
  - repeat, clamp to edge
- Auto mip-map generation
- Texture Combine
  - Support following extensions as core features
    - ARB_texture_env_combine
      - Extends combine functions by including support for
        - ADD_SIGNED, SUBTRACT, and INTERPOLATE
      - Application can program arguments
        - passed in as inputs to combine units
    - ARB_texture_env_dot3
      - Compute a per-pixel dot product
        - Useful for per-pixel lighting & bump mapping

Texture Combine

COMBINE_RGB	Texture Function
REPLACE	Arg0
MODULATE	Arg0 * Arg1
ADD	Arg0 + Arg1
ADD_SIGNED	Arg0 + Arg1 – 0.5
INTERPOLATE	Arg0 * Arg2 + Arg1 * (1 – Arg2)
SUBTRACT	Arg0 – Arg1
DOT3_RGB	$4 * ((Arg0_r - 0.5) * (Arg1_r - 0.5) + (Arg0_g - 0.5) * (Arg1_g - 0.5) + (Arg0_b - 0.5) * (Arg1_b - 0.5))$
DOT3_RGBA	$4 * ((Arg0_r - 0.5) * (Arg1_r - 0.5) + (Arg0_g - 0.5) * (Arg1_g - 0.5) + (Arg0_b - 0.5) * (Arg1_b - 0.5))$

COMBINE_ALPHA	Texture Function
REPLACE	Arg0
MODULATE	Arg0 * Arg1
ADD	Arg0 + Arg1
ADD_SIGNED	Arg0 + Arg1 – 0.5
INTERPOLATE	Arg0 * Arg2 + Arg1 * (1 – Arg2)
SUBTRACT	Arg0 – Arg1

Texture Combine

SRCn_RGB	OPERANDn_RGB	Argument
TEXTURE	SRC_COLOR	C_s
	ONE_MINUS_SRC_COLOR	$1 - C_s$
	SRC_ALPHA	A_s
	ONE_MINUS_SRC_ALPHA	$1 - A_s$
CONSTANT	SRC_COLOR	C_c
	ONE_MINUS_SRC_COLOR	$1 - C_c$
	SRC_ALPHA	A_c
	ONE_MINUS_SRC_ALPHA	$1 - A_c$
PRIMARY_COLOR	SRC_COLOR	C_f
	ONE_MINUS_SRC_COLOR	$1 - C_f$
	SRC_ALPHA	A_f
	ONE_MINUS_SRC_ALPHA	$1 - A_f$
PREVIOUS	SRC_COLOR	C_p
	ONE_MINUS_SRC_COLOR	$1 - C_p$
	SRC_ALPHA	A_p
	ONE_MINUS_SRC_ALPHA	$1 - A_p$

Table 5.2: Arguments for COMBINE_RGB functions.

SRCn_ALPHA	OPERANDn_ALPHA	Argument
TEXTURE	SRC_ALPHA	A_s
	ONE_MINUS_SRC_ALPHA	$1 - A_s$
CONSTANT	SRC_ALPHA	A_c
	ONE_MINUS_SRC_ALPHA	$1 - A_c$
PRIMARY_COLOR	SRC_ALPHA	A_f
	ONE_MINUS_SRC_ALPHA	$1 - A_f$
PREVIOUS	SRC_ALPHA	A_p
	ONE_MINUS_SRC_ALPHA	$1 - A_p$

Table 5.3: Arguments for COMBINE_ALPHA functions.

OES_matrix_palette

- Used to do vertex skinning in OpenGL ES 1.1
  - Modified version of ARB_matrix_palette
  - A set of matrix indices & weights per vertex.
    - # of matrices / vertex can be queried using glGetIntegerv
    - Minimum # supported
      - 3 matrices / vertex
  - Matrix palette
    - Matrix indices specified per vertex into the palette
    - Size of matrix palette can be queried using glGetIntegerv
    - Minimum supported
      - Palette of 16 matrices.

User Clip Planes

- Useful for portal culling algorithms
- Support a minimum of one user clip plane
  - OpenGL supports a minimum of 6 user planes
- How does it work
  - Clip plane in object coordinates specified by glClipPlane
  - Compute dot product of clip plane & point in eye space
    - If dot product >= 0, point is inside plane

Points

- OES_point_sprite
  - Accelerate rendering of particle effects
  - Render particles as points instead of quads
  - Texture coordinates smoothly interpolated across point from 0.0 to 1.0
    - Application can enable/disable this per texture
  - Point sprites enabled using GL_POINT_SPRITE
- OES_point_size_array
  - Extends how points & point sprites are rendered
  - glPointSizePointerOES
    - Used to specify the point size array
    - Point sizes can also be stored in a buffer object
- Distance attenuation of points
  - Attenuate point size based on distance from eye

State

- OpenGL ES 1.0 supports static state queries
- OpenGL ES 1.1 allows dynamic state queries

  Why ?
  - Implement state save/restore
  - Middle-ware or application can implement state push and pop with an infinite stack depth

  State Functions
  - Refer to the OpenGL ES 1.1 spec

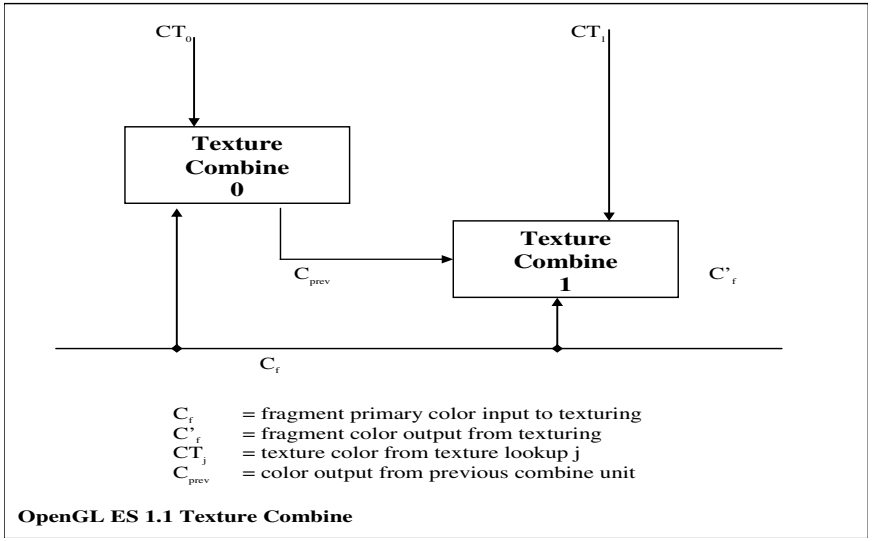
OES_draw_texture

- Render pixel rectangles using texture units
- Why ?
  - Useful for fast rendering of sprites, bitmapped font glyphs, & 2D framing elements in games
- glDrawPixels vs. OES_draw_texture
  - Both are similar in functionality
  - OES_draw_texture uses texture pipe & texture objects
    - glDrawPixels inefficient because pixel data supplied by application cannot be cached.

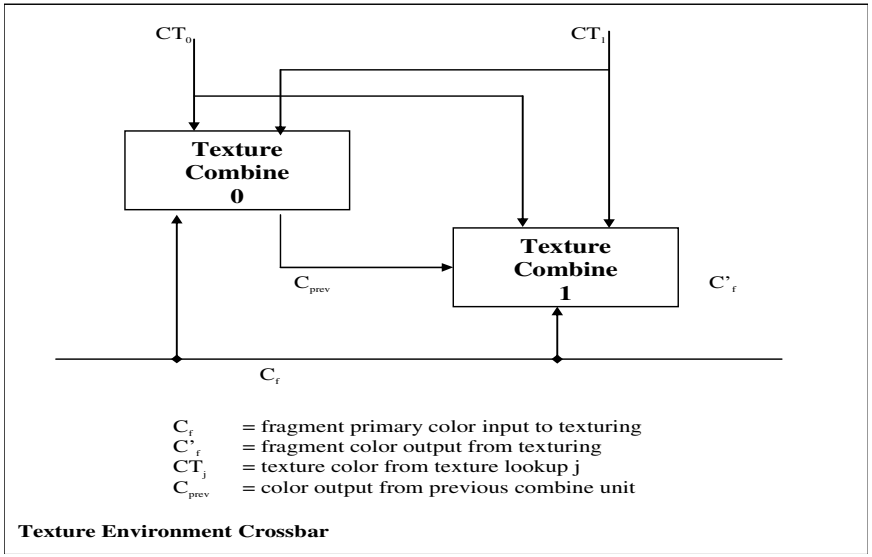
OpenGL ES 1.1 Extension Pack

- Collection of optional extensions
  - Texture Crossbar
  - Texture Cubemap
  - Texture Mirrored Repeat
  - Blending Enhancements
  - Stencil Enhancements
  - Extended Matrix Palette
  - Framebuffer Objects
    - Discussed in OpenGL 2.0 slides

OpenGL ES 1.1 Extension Pack



OpenGL ES 1.1 Texture Combine



Texture Environment Crossbar

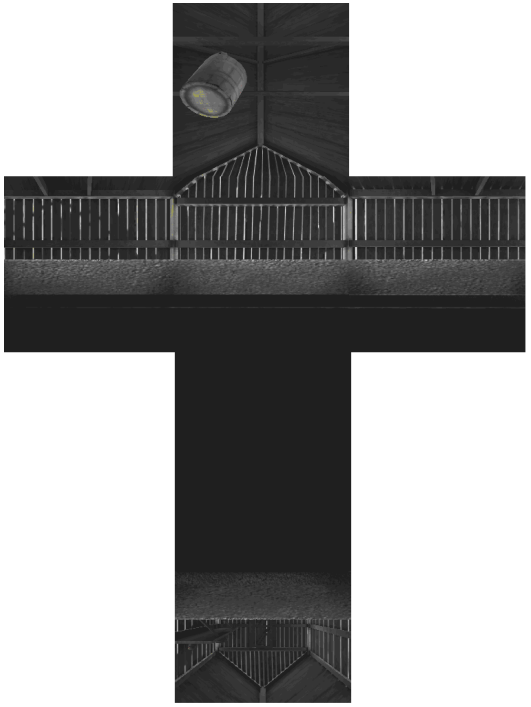
- Texture Crossbar
  - Adds capability to use the texture color from other texture units as sources to the COMBINE unit.

OpenGL ES 1.1 Extension Pack

- Mirrored Texture Addressing
- Blending Extensions
  - Additional Blending Equations
    - GL_FUNC_SUBTRACT
    - GL_FUNC_REVERSE_SUBTRACT
  - glBlendFuncSeparate, glBlendEquationSeparate
    - Separate blending functions and equations for RGB and alpha.
- New StencilOp functions
  - GL_INCR_WRAP and GL_DECR_WRAP

OpenGL ES 1.1 Extension Pack

- Cube Maps
  - Accurate real-time reflections in handheld 3D games
    - Given normal & position, compute reflection vector
    - Use reflection vector as texture coordinate
    - Per-pixel computed reflection vector is used to index one of 6 faces of cube
      - Generate per-pixel reflection value
  - Per-pixel “specular” lighting
    - Per-vertex specular highlight requires very finely tessellated geometry
    - With cube-maps you can compute specular color per pixel
      - And not require lots of geometry



OpenGL ES 1.1 Extension Pack

- Extended Matrix Palette
  - OES_matrix_palette in OpenGL ES 1.1 recommends
    - a minimum matrix palette size of 9 matrices and
    - up to 3 matrices / vertex
  - Problems
    - Requires games to break geometry into smaller primitive lists
      - not efficient for HW
    - Many cases where 4 matrices / vertex is required
  - OES_extended_matrix_palette increases minimums to
    - Matrix palette size = 32
    - # of matrices / vertex = 4

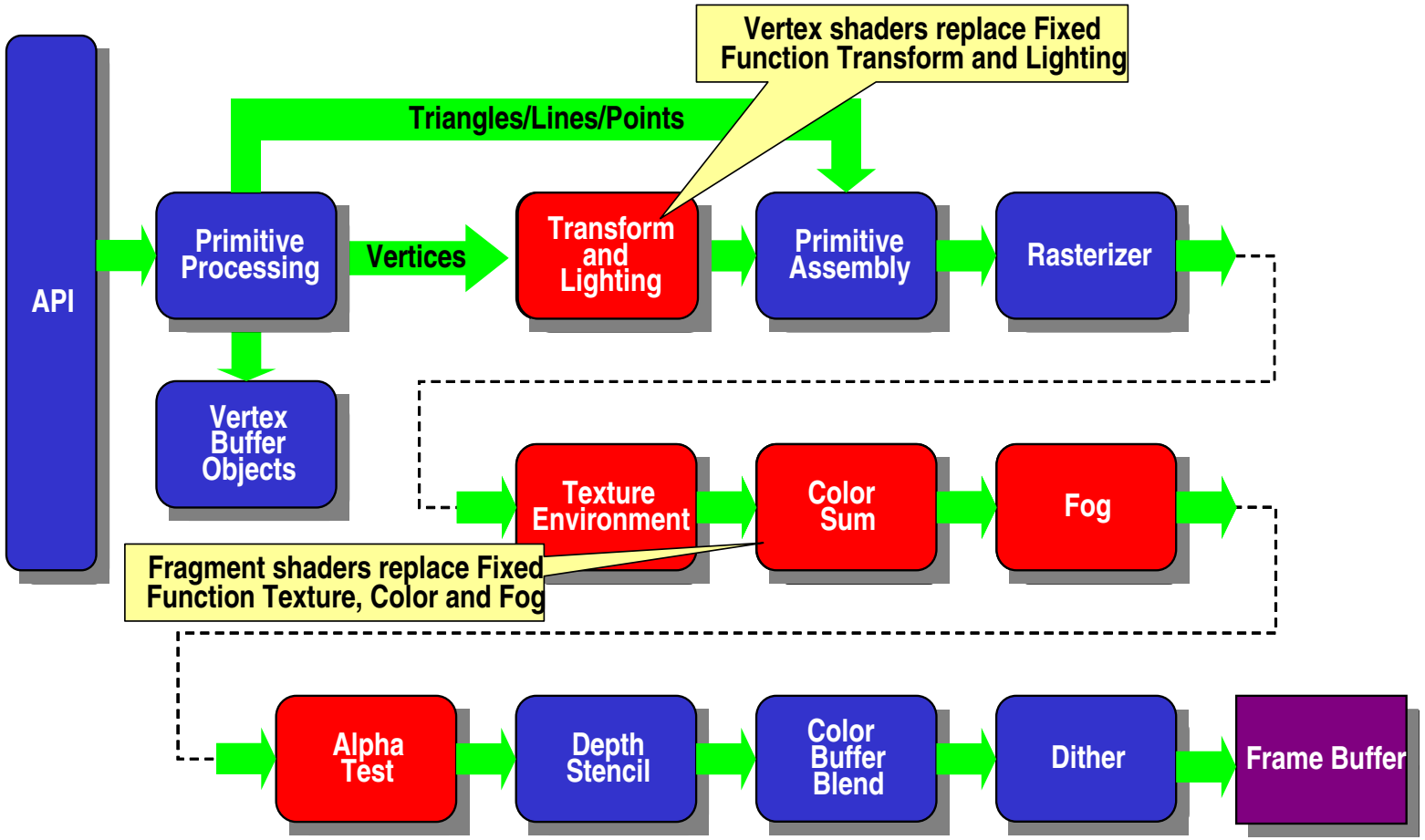
OpenGL ES 2.0

GameDevelopers Conference

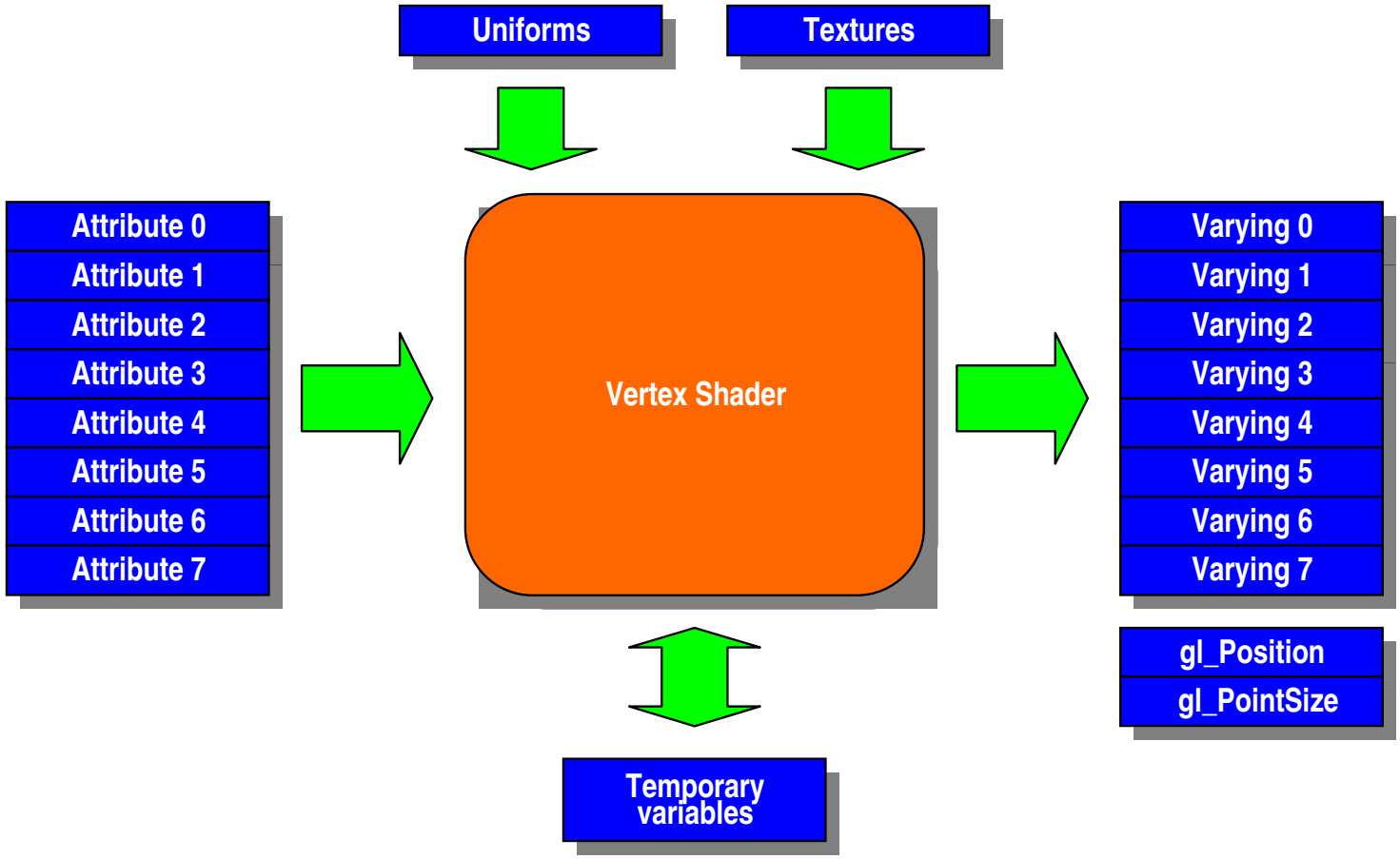
OpenGL ES 2.0 – Overview

- 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 2.0 Programmable Pipeline



OpenGL ES 2.0 – Vertex Shader



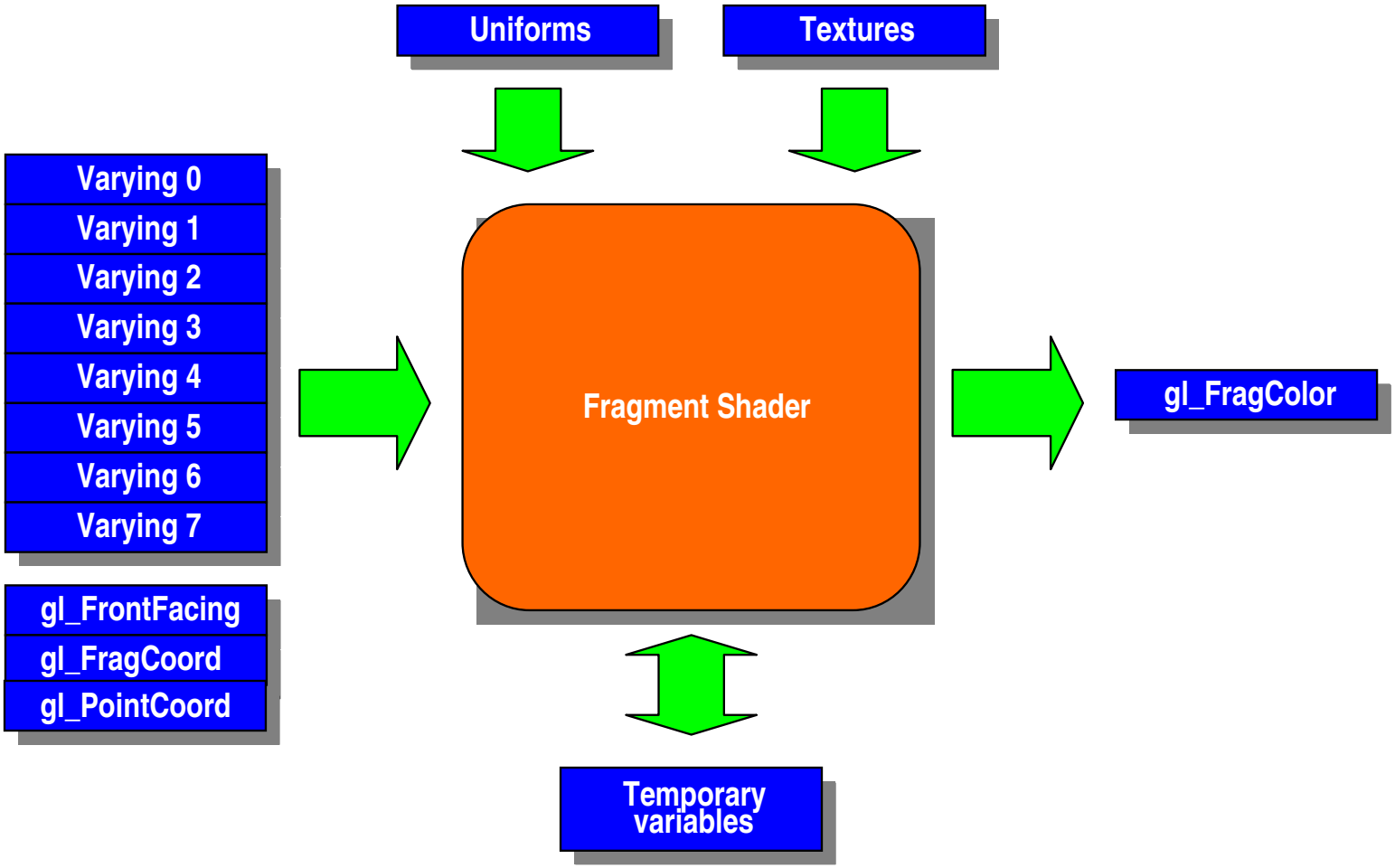
OpenGL ES 2.0 – Example Vertex Shader

```
#version 110
attribute vec4 Vertex;
attribute vec2 VertexSt;
uniform mat4 Transform;

varying vec2 TexCoord;
invariant gl_Position;

void main()
{
   gl_Position = Transform * Vertex;
   TexCoord = VertexSt;
}
```

OpenGL ES 2.0 – Fragment Shader



OpenGL ES 2.0 – Example Fragment Shader

```
#version 110

precision highp float; // set default precision

uniform sampler2D Sampler;

varying vec2 TexCoord;

void main()

{

   vec4 color = texture2D(Sampler, TexCoord);

   color *= 0.5;

   gl_FragColor = color;

}
```

OpenGL ES 2.0 – What's In?

- Vertex Data
  - 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

OpenGL ES 2.0 - What's In

- 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 2.0 - 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.

OpenGL ES 2.0 - What's In

- 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
      - eg. highp mat4 m;
    - precision <precision-qualifer> <type> statement
- glGetShaderPrecisionFormatOES
  - Returns range and precision for formats supported

OpenGL ES 2.0 - 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 2.0 - What's In

- The Invariant Qualifier

  To ensure that a particular output variable is invariant,

  - invariant gl_Position; // make existing gl_Position be invariant

  The invariant qualifier must appear before any storage qualifiers (**varying**) when combined with a declaration

  Only variables that are output from a shader can be declared as invariant

  To guarantee invariance of a particular output variable in two shaders, the following must also be true:

  - The output variable is declared as invariant in both shaders.
  - The same values must be input to all shader input variables consumed by expressions and flow control contributing to the value assigned to the output variable.
  - The texture formats, texel values, and texture filtering are set the same way for any texture function calls contributing to the value of the output variable.

OpenGL ES 2.0 – What's In

- Frame-buffer Objects

  Simplified version of EXT_framebuffer_object

  Efficient way of doing Render to Texture

  Advantages:

  - No additional GL contexts are needed
    - Switching between framebuffers is faster than doing a context switch (eglMakeCurrent)
  - Format of framebuffer is determined by texture or pixel format
  - Images (renderbuffers / textures) can be shared across framebuffers

OpenGL ES 2.0 - What's In

- Frame-buffer Objects contd.
  - No support for accum buffers or multiple render targets
  - Support for 3D textures is optional
  - Support for rendering into a mip-level is optional
  - Adds a new 16-bit RGB565 internal format to RenderBufferStorage
  - 8-bit stencil buffer is required

Frame-buffer Objects - Sample

```
// Create the depth buffer
glGenRenderbuffersOES(1, &m_nReflectDepthId);
glBindRenderbufferOES(GL_RENDERBUFFER_OES, m_nReflectDepthId);
glRenderbufferStorageOES(GL_RENDERBUFFER_OES,
                         GL_DEPTH_COMPONENT16, m_nReflectSize, m_nReflectSize);

// Create the FBO
glGenFramebuffersOES(1, &m_nReflectFramebufferId);
glBindFramebufferOES(GL_FRAMEBUFFER_OES, m_nReflectFramebufferId);

// Bind the texture and the depth buffer
glFramebufferTexture2DOES(GL_FRAMEBUFFER_OES,
                                    GL_COLOR_ATTACHMENT0_OES,
                                    GL_TEXTURE_2D, m_nReflectTexId, 0);
glFramebufferRenderbufferOES(GL_FRAMEBUFFER_OES,
                                      GL_DEPTH_ATTACHMENT_OES,
                                      GL_RENDERBUFFER_OES, m_nReflectDepthId);

CHECK_FRAMEBUFFER_STATUS();
```

OpenGL ES 2.0 – 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 2.0 – 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 2.0 – Vertex Arrays

In

- glVertexAttribPointer
- glEnableVertexAttribArray
- glDrawArrays
- glDrawElements
- Triangles, tri strips, tri fans, line, line strips, line loop, point sprites

Out

- glVertexPointer
- glTexCoordPointer
- glColorPointer
- glNormalPointer
- glSecondaryColorPointer
- glFogCoordPointer
- glEdgeFlagPointer
- glEnableClientState, glDisableClientState
- glArrayElement, glMultiDrawArrays, glDrawRangeElements
- Quads, quad strips, polygons and points

OpenGL ES 2.0 – Vertex Specification

In

- glVertexAttrib{1234}f[v]

Out

- Immediate Mode
  - glBegin/glEnd
  - glVertex
  - All other glVertexAttrib* variants
- All other per-primitive attributes
  - glMultiTexCoord, glNormal, glColor, glFogCoord, glSecondaryColor, etc.
- Color index mode

OpenGL ES 2.0 – Buffer Objects

In

- glBindBuffer
- glDeleteBuffers
- glGenBuffers
- glBufferData
- glBufferSubData

Optional

- glMapBuffer
- glUnmapBuffer

OpenGL ES 2.0 – Transformation

In

- glViewport

Out

- Everything else!
  - glMatrixMode
  - glLoadMatrix
  - glPush/PopMatrix
  - glTranslate/glRotate/glScale
  - glTexGen
  - glFrustum/glOrtho

OpenGL ES 2.0 – Colors and Coloring

In

- glFrontFace

Out

- glMaterial
- glLight
- glLightModel
- glColorMaterial
- glShadeModel

OpenGL ES 2.0 – AA, Points, Lines, and Polygons

In

- Multisampling
- glLineWidth
- glPointSize
- Culling
- glPolygonOffset

Out

- Point and line smooth
- glPointParameters
  - Now done in the vertex shader
- Line and polygon stippling
- GL_POLYGON_SMOOTH
- glPolygonMode
  - No point or line mode

OpenGL ES 2.0 – Pixels and Bitmaps

In

- glPixelStorei
  - For loading textures and reading from the screen
  - Only supports GL_PACK_ALIGNMENT and GL_UNPACK_ALIGNMENT
- glReadPixels
  - Limited number of formats

Out

- Imaging subset (filters, histograms, minmax)
- glDrawPixels
- glCopyPixels
- glPixelZoom
- glBitmap
- glRasterPos

OpenGL ES 2.0 – Textures

In

- Most common formats
  - GL_RGB, GL_RGBA, GL_LUMINANCE, GL_ALPHA, GL_LUMINANCE_ALPHA
- 2D/3D/Cubemaps
  - glTexImage/glTexSubImage
  - glTexImage3D/glTexSubImage3D
  - glCopyTexImage2D/
  - glCopyTexSubImage{2D|3D}
- Compressed texture entry points
- Texture parameters
  - All filtering modes
  - Clamp-to-edge, repeat, and mirror-repeat wrap modes

Out

- Texture Environment
  - No fixed function blending!
- 1D Textures
- Texture Parameters
  - No LOD control
  - No texture border
  - Thus, no clamp-to-border or clamp wrap modes
  - Generate mipmaps
- Texture Priorities
  - glPrioritizeTextures
  - glAreTexturesResident
- Dynamic texture state queries
  - glGetTexImage, glGetTexParameter
- Fog

OpenGL ES 2.0 – Per-Fragment Operations

In

- Stencil Test
- Scissor Test
- Sample Coverage
- Alpha Blending
  - Add, Subtract, ReverseSubtract
  - glBlendEquationSeparate
  - glBlendFuncSeparate
- Depth Test
- Dithering

Out

- Occlusion Queries
- Alpha Test
- Some alpha blend modes
  - MIN, MAX, LOGIC_OP
- glDrawBuffer and glDrawBuffers
  - No MRTs
- Accumulation Buffer

Questions?