Advances in OpenGL and OpenGL ES

Session 513

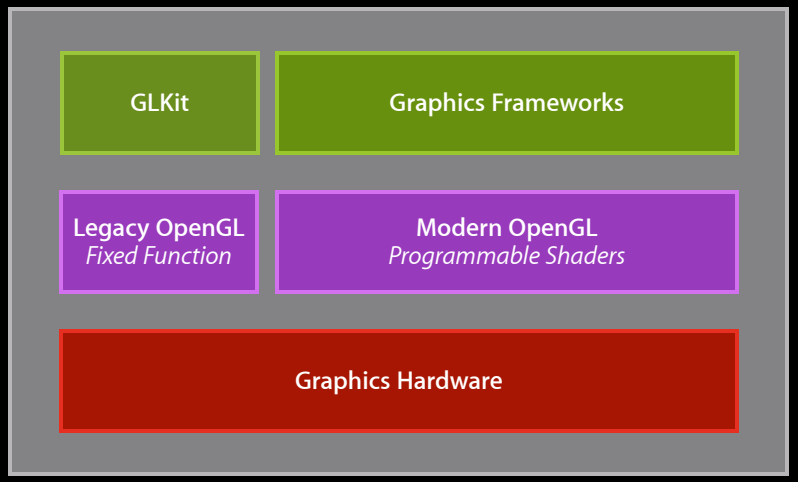
Chris Niederauer
GPU Software

OpenGL and OpenGL ES

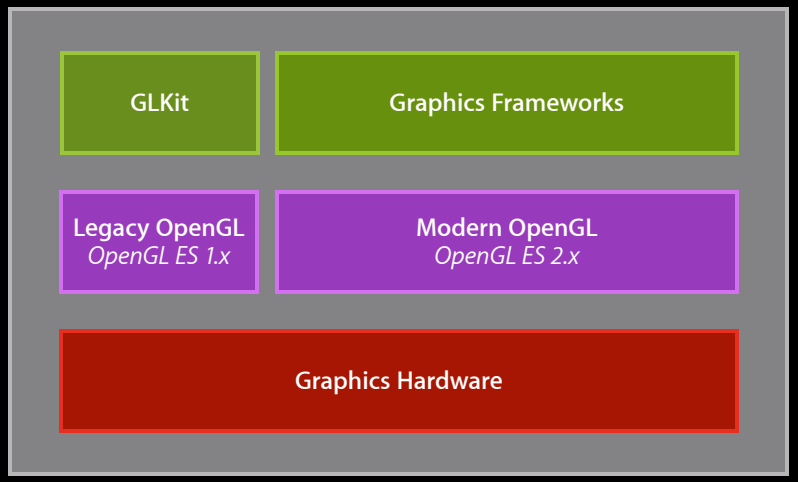
High-performance rendering

- Access to vastly powerful GPUs
- Broadly used for games, visualization, and entertainment
- Foundation of visual technologies in iOS and OS X

OpenGL and OpenGL ES

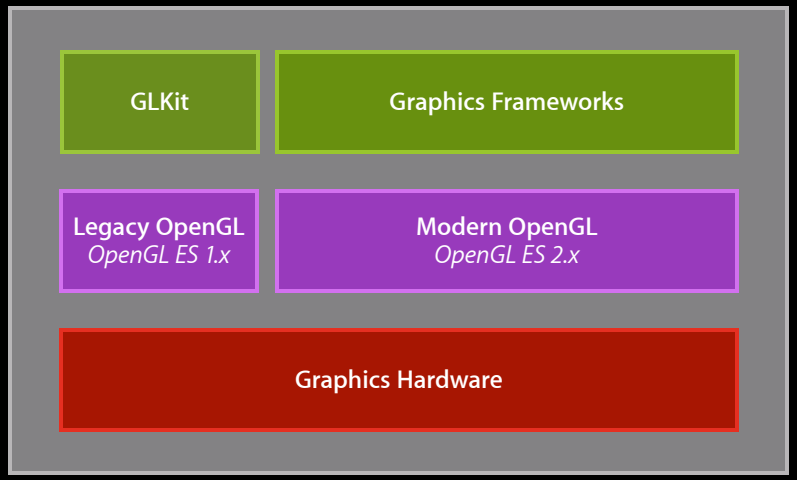


OpenGL and OpenGL ES

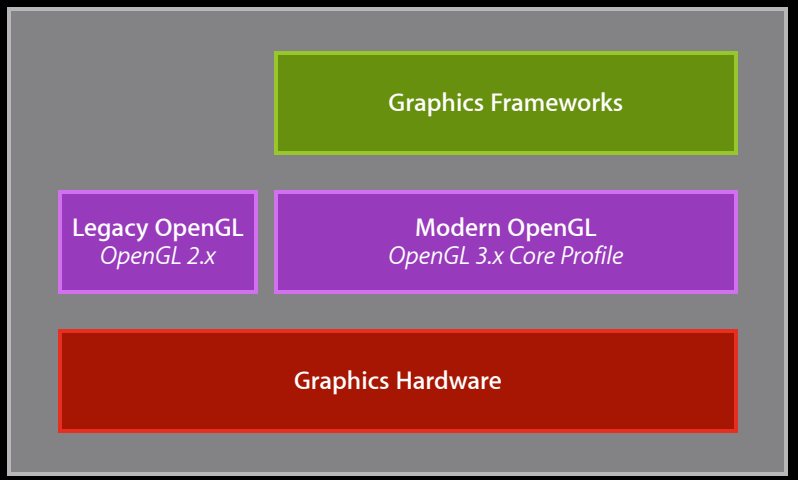


iOS

OpenGL and OpenGL ES

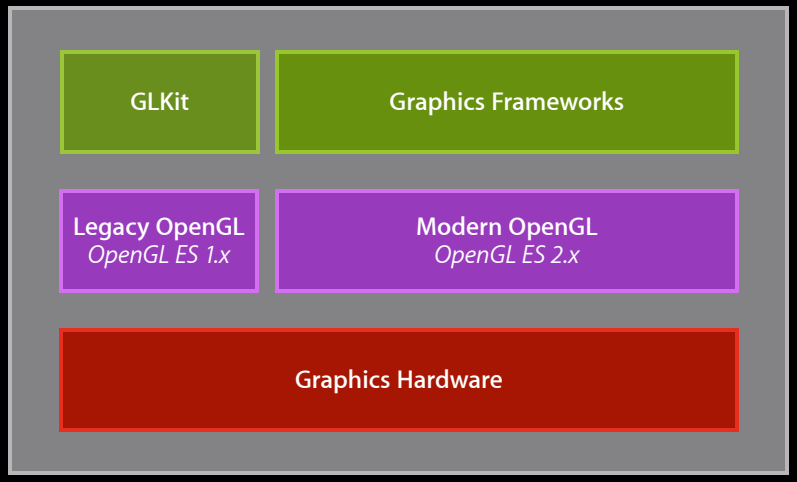


iOS

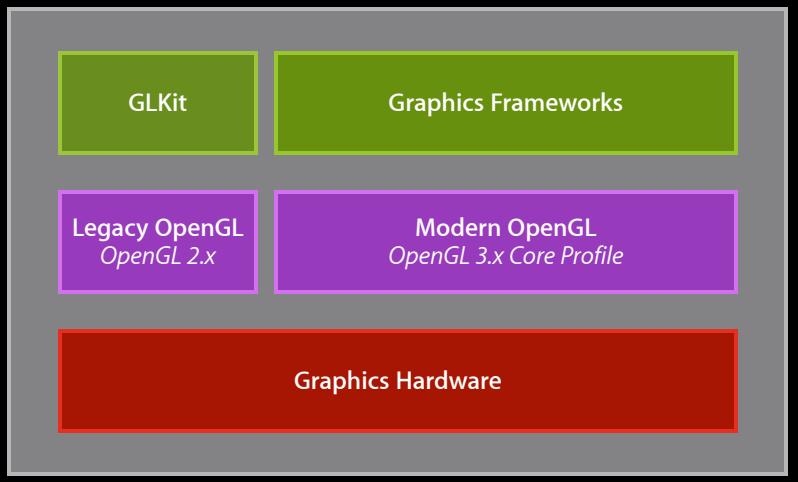


OS X

OpenGL and OpenGL ES



iOS



OS X

Introduction

Agenda

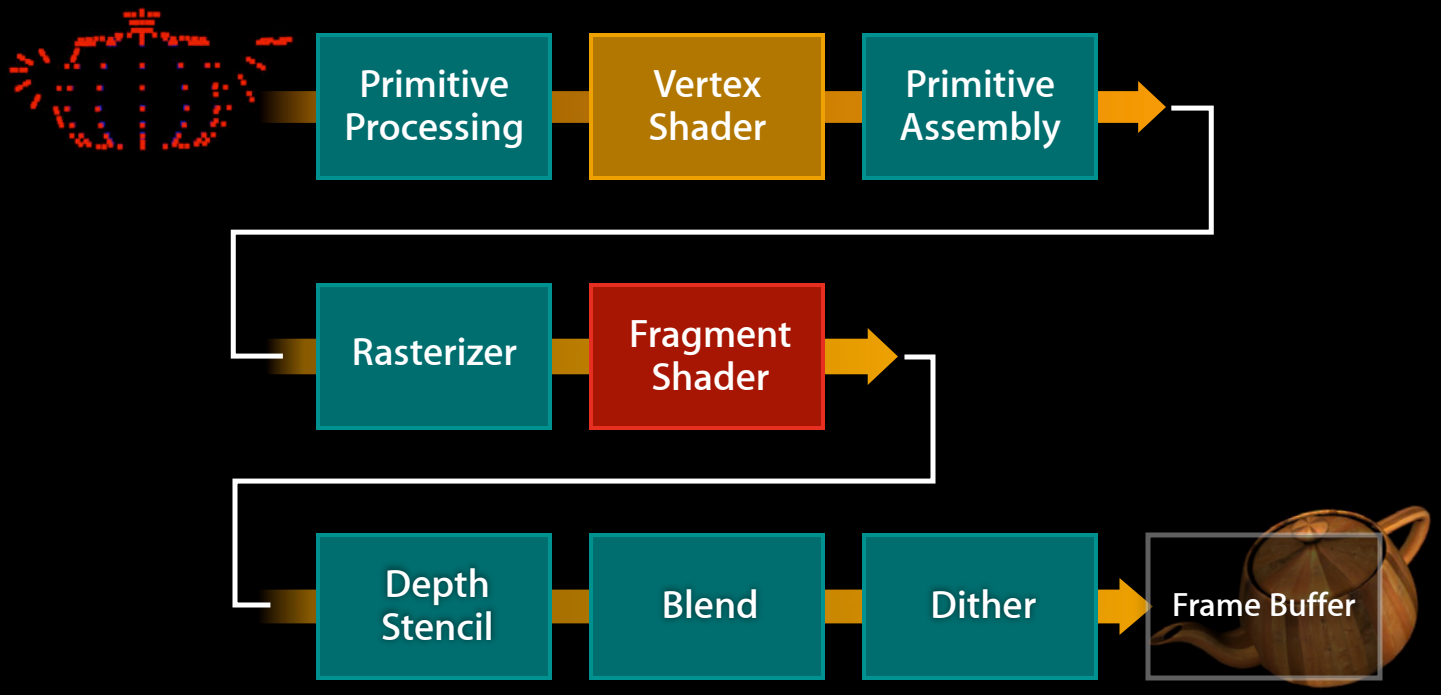
- New extensions for iOS
- GLKit refresher
- High resolution for OS X

Programmable Blending

Allan Schaffer
Graphics and Game Technologies Evangelist

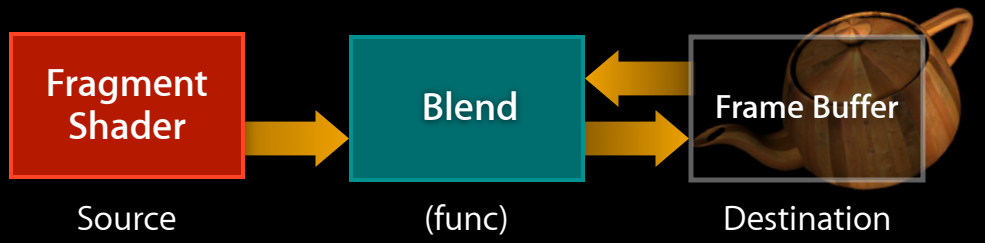
OpenGL ES 2.0

Programmable graphics pipeline



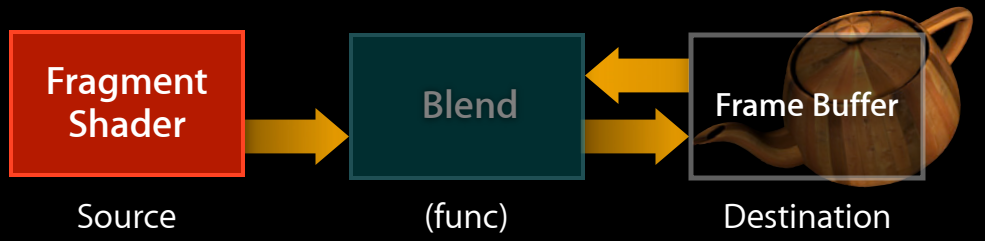
OpenGL ES 2.0

Standard blending



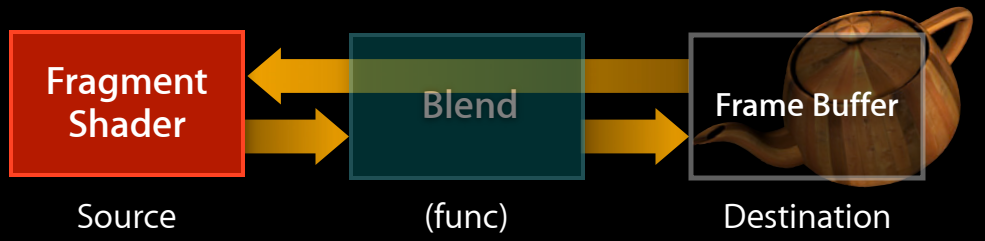
OpenGL ES 2.0

Programmable blending



OpenGL ES 2.0

Programmable blending



Framebuffer Fetch

APPLE_shader_framebuffer_fetch

- gl_LastFragData[0]
  - Provides current framebuffer color in fragment shader
  - Read-only
  - All iOS 6 devices
  - Coexists with built-in blending

Framebuffer Fetch

Example shader

```
#extension GL_APPLE_shader_framebuffer_fetch : require
varying lowp vec4 color;

void main()
{
  // GL_ONE, GL_ONE_MINUS_SRC_ALPHA
  gl_FragColor = color + (gl_LastFragData[0] * (1.0 - color.a));

or..

  // Difference Blend
  gl_FragColor = abs(color - gl_LastFragData[0]);
}
```

Framebuffer Fetch

Example shader

```
#extension GL_APPLE_shader_framebuffer_fetch : require
varying lowp vec4 color;

void main()
{
  // GL_ONE, GL_ONE_MINUS_SRC_ALPHA
  gl_FragColor = color + (gl_LastFragData[0] * (1.0 - color.a));

or..

  // Difference Blend
  gl_FragColor = abs(color - gl_LastFragData[0]);
}
```

Framebuffer Fetch

Example shader

```
#extension GL_APPLE_shader_framebuffer_fetch : require
varying lowp vec4 color;

void main()
{
  // GL_ONE, GL_ONE_MINUS_SRC_ALPHA
  gl_FragColor = color + (gl_LastFragData[0] * (1.0 - color.a));

or..

  // Difference Blend
  gl_FragColor = abs(color - gl_LastFragData[0]);
}
```

Framebuffer Fetch

Example shader

```
#extension GL_APPLE_shader_framebuffer_fetch : require
varying lowp vec4 color;

void main()
{
  // GL_ONE, GL_ONE_MINUS_SRC_ALPHA
  gl_FragColor = color + (gl_LastFragData[0] * (1.0 - color.a));

or..

  // Difference Blend
  gl_FragColor = abs(color - gl_LastFragData[0]);
}
```

Framebuffer Fetch

Hard light shader

```
#extension GL_APPLE_shader_framebuffer_fetch : require
varying lowp vec4 colorVarying;

// “Hard Light” Blend
#define BlendHardLight(color, dest)   (color < 0.5 ?   \
    (2.0 * color * dest) : (1.0 - 2.0 * (1.0 - color) * (1.0 - dest)))


void main()
{
    gl_FragColor.r = BlendHardLight(colorVarying.r, gl_LastFragData[0].r);
    gl_FragColor.g = BlendHardLight(colorVarying.g, gl_LastFragData[0].g);
    gl_FragColor.b = BlendHardLight(colorVarying.b, gl_LastFragData[0].b);
    gl_FragColor.a = colorVarying.a;
}
```

Framebuffer Fetch

Hard light shader

```
#extension GL_APPLE_shader_framebuffer_fetch : require
varying lowp vec4 colorVarying;

// “Hard Light” Blend
#define BlendHardLight(color, dest)   (color < 0.5 ?   \
    (2.0 * color * dest) : (1.0 - 2.0 * (1.0 - color) * (1.0 - dest)))

void main()
{
    gl_FragColor.r = BlendHardLight(colorVarying.r, gl_LastFragData[0].r);
    gl_FragColor.g = BlendHardLight(colorVarying.g, gl_LastFragData[0].g);
    gl_FragColor.b = BlendHardLight(colorVarying.b, gl_LastFragData[0].b);
    gl_FragColor.a = colorVarying.a;
}
```

Framebuffer Fetch

Hard light shader

```
#extension GL_APPLE_shader_framebuffer_fetch : require
varying lowp vec4 colorVarying;

// “Hard Light” Blend
#define BlendHardLight(color, dest)   (color < 0.5 ?   \
    (2.0 * color * dest) : (1.0 - 2.0 * (1.0 - color) * (1.0 - dest)))


void main()
{
    gl_FragColor.r = BlendHardLight(colorVarying.r, gl_LastFragData[0].r);
    gl_FragColor.g = BlendHardLight(colorVarying.g, gl_LastFragData[0].g);
    gl_FragColor.b = BlendHardLight(colorVarying.b, gl_LastFragData[0].b);
    gl_FragColor.a = colorVarying.a;
}
```

Framebuffer Fetch

Local postprocessing effects

- Step one: Draw scene
- Step two: Draw full-screen quad with shader, e.g.:

```
#extension GL_APPLE_shader_framebuffer_fetch : require
void main()
{
  // RGB to grayscale
  mediump float lum = dot(gl_LastFragData[0], vec4(0.30,0.59,0.11,0.0));
  gl_FragColor =  vec4(lum, lum, lum, 1.0);
}
```

- More efficient than render-to-texture

Framebuffer Fetch

Local postprocessing effects

- Step one: Draw scene
- Step two: Draw full-screen quad with shader, e.g.:

```
#extension GL_APPLE_shader_framebuffer_fetch : require
void main()
{
  // RGB to grayscale
  mediump float lum = dot(gl_LastFragData[0], vec4(0.30,0.59,0.11,0.0));
  gl_FragColor =  vec4(lum, lum, lum, 1.0);
}
```

- More efficient than render-to-texture

Programmable Blending

Going further

- Unique blends using extended GLSL operations
  - Advanced blends
  - Custom overlays
  - Lighting effects
- Using framebuffer RGB values to do a texture lookup
  - Color grading/globally modified color
- Blends with noncolor framebuffer data
  - Normals, ambient occlusion, etc.

Fine-Grain Texture Copy

Fine-Grain Texture Copy

Target audience

- Most games: Load all textures
  - Only have a few textures
  - Only load as many as fits in RAM

Fine-Grain Texture Copy

Target audience

- Most games: Load all textures
  - Only have a few textures
  - Only load as many as fits in RAM
- Some games: Dynamic texture loading
  - Load and delete textures between levels, etc.

Fine-Grain Texture Copy

Target audience

- Most games: Load all textures
  - Only have a few textures
  - Only load as many as fits in RAM
- Some games: Dynamic texture loading
  - Load and delete textures between levels, etc.
- Few games: Fully dynamic texture management
  - Keep a runtime texture "budget"
  - Load and delete textures during gameplay
  - Wish to have even finer grain control

Fine-Grain Texture Copy

Concept

- For games with dynamic texture management
  - Quickly create storage and copy texture contents
  - Enables a “texture swap” algorithm
    - Add higher resolution mipmap base level as needed
    - Remove mipmap base level to reduce memory footprint
- Enabled by two new extensions
  - EXT_texture_storage
  - APPLE_texture_copy

Fine-Grain Texture Copy

EXT_texture_storage

- Immutable texture object
  - Defines all texture properties in a single call
  - Memory allocations and completeness checked up front
  - Texture data uploaded via `glTexSubImage2D`
- Supported on all iOS 6 devices

Fine-Grain Texture Copy

EXT_texture_storage usage

```
// Create and bind texture name
glGenTextures(1, &name);
glBindTexture(GL_TEXTURE_2D, name);

// Define & allocate texture storage
glTexStorage2DEXT(GL_TEXTURE_2D, num_levels, GL_RGBA8_OES, width, height);

// Load data
for (int level=0; level<num_levels; level++)
    glTexSubImage2D(GL_TEXTURE_2D, level, ..., data[level]);
```

Fine-Grain Texture Copy

EXT_texture_storage usage

```
// Create and bind texture name
glGenTextures(1, &name);
glBindTexture(GL_TEXTURE_2D, name);

// Define & allocate texture storage
glTexStorage2DEXT(GL_TEXTURE_2D, num_levels, GL_RGBA8_OES, width, height);

// Load data
for (int level=0; level<num_levels; level++)
    glTexSubImage2D(GL_TEXTURE_2D, level, ..., data[level]);
```

Fine-Grain Texture Copy

EXT_texture_storage usage

```
// Create and bind texture name
glGenTextures(1, &name);
glBindTexture(GL_TEXTURE_2D, name);

// Define & allocate texture storage
glTexStorage2DEXT(GL_TEXTURE_2D, num_levels, GL_RGBA8_OES, width, height);

// Load data
for (int level=0; level<num_levels; level++)
    glTexSubImage2D(GL_TEXTURE_2D, level, ..., data[level]);
```

Fine-Grain Texture Copy

EXT_texture_storage usage

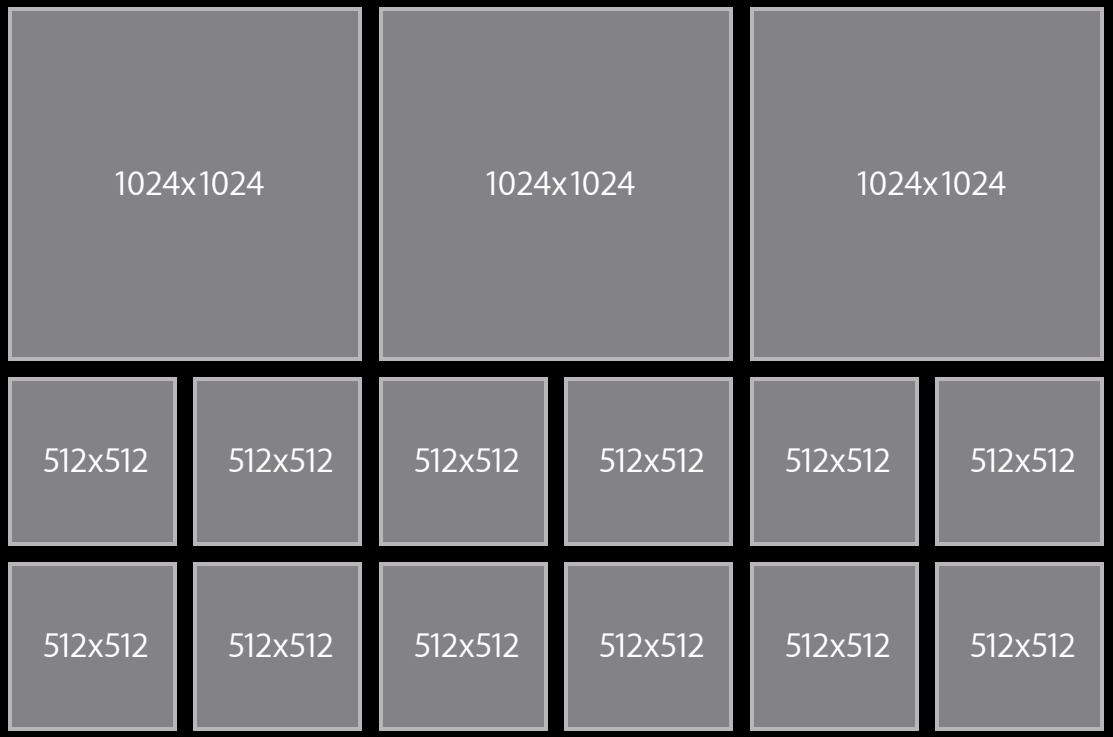
```
// Create and bind texture name
glGenTextures(1, &name);
glBindTexture(GL_TEXTURE_2D, name);

// Define & allocate texture storage
glTexStorage2DEXT(GL_TEXTURE_2D, num_levels, GL_RGBA8_OES, width, height);

// Load data
for (int level=0; level<num_levels; level++)
    glTexSubImage2D(GL_TEXTURE_2D, level, ..., data[level]);
```

Fine-Grain Texture Copy

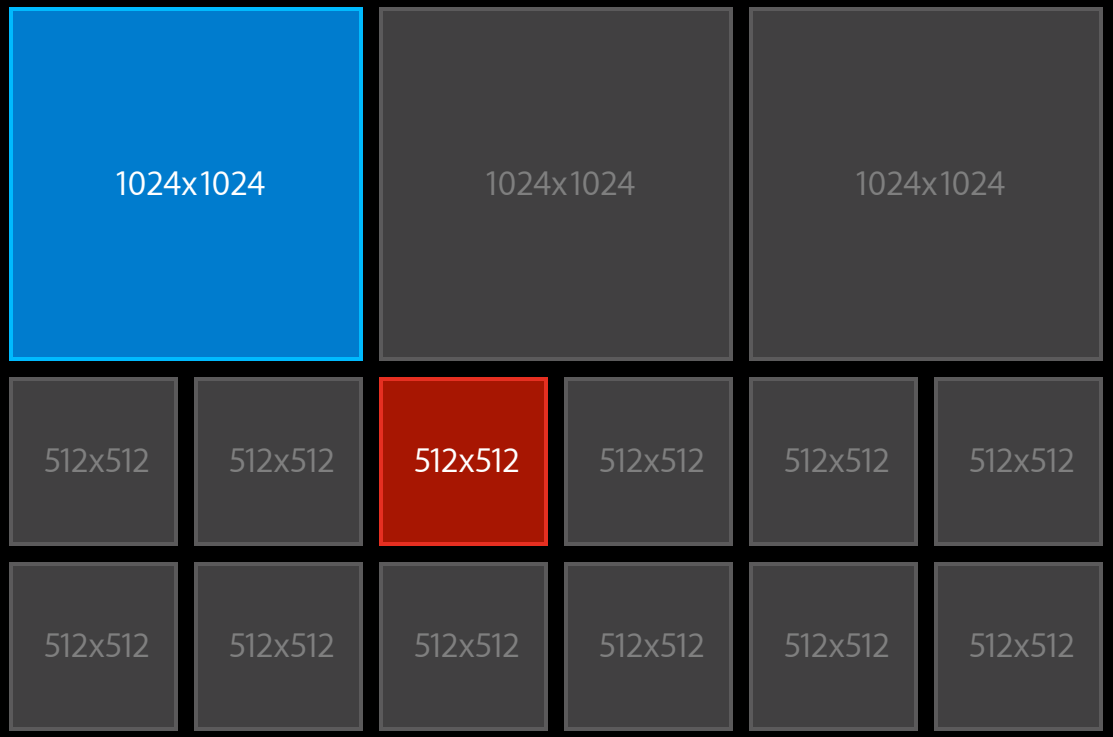
APPLE_copy_texture_levels

- Fast copy of specified mipmap levels between textures
  - Based on dimensions, for example:
    - [1x1] thru [512x512] levels of source copied to
      [1x1] thru [512x512] levels of destination
- Enables fine-grain memory management of levels
- Requires immutable textures (`EXT_texture_storage`)
- Supported on all iOS 6 devices

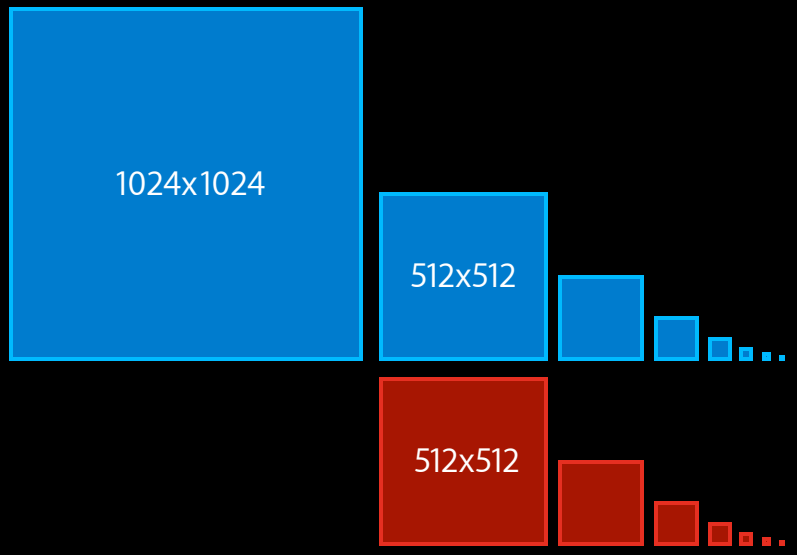
Fine-Grain Texture Copy



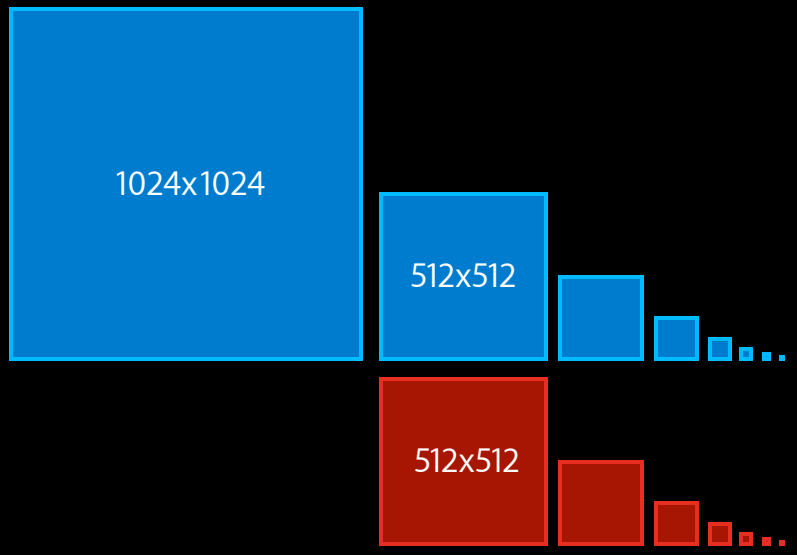
Fine-Grain Texture Copy



Fine-Grain Texture Copy

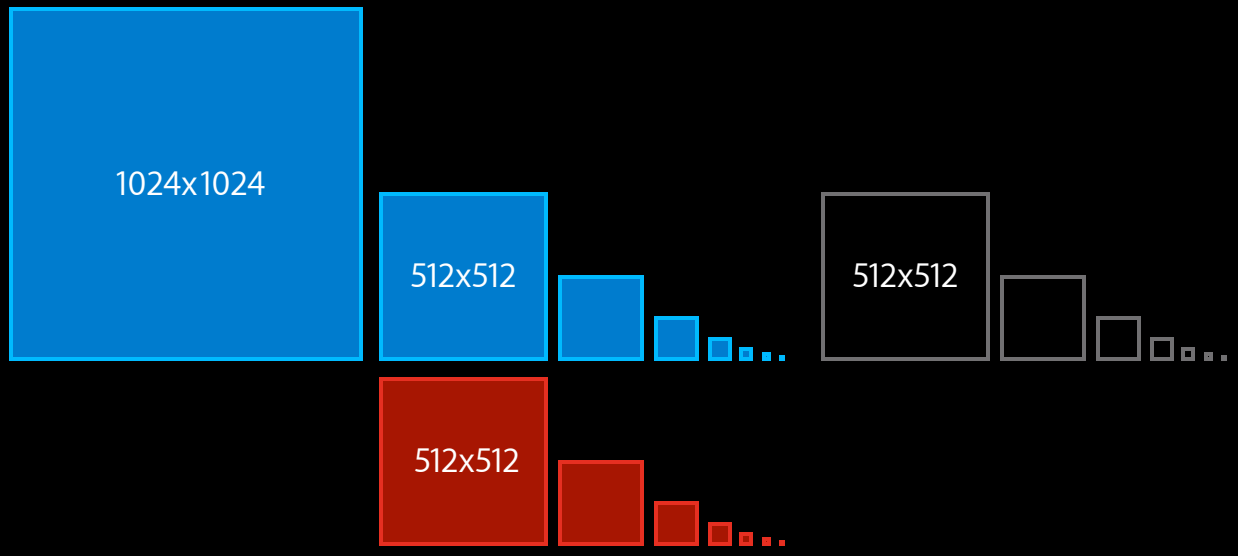


Fine-Grain Texture Copy



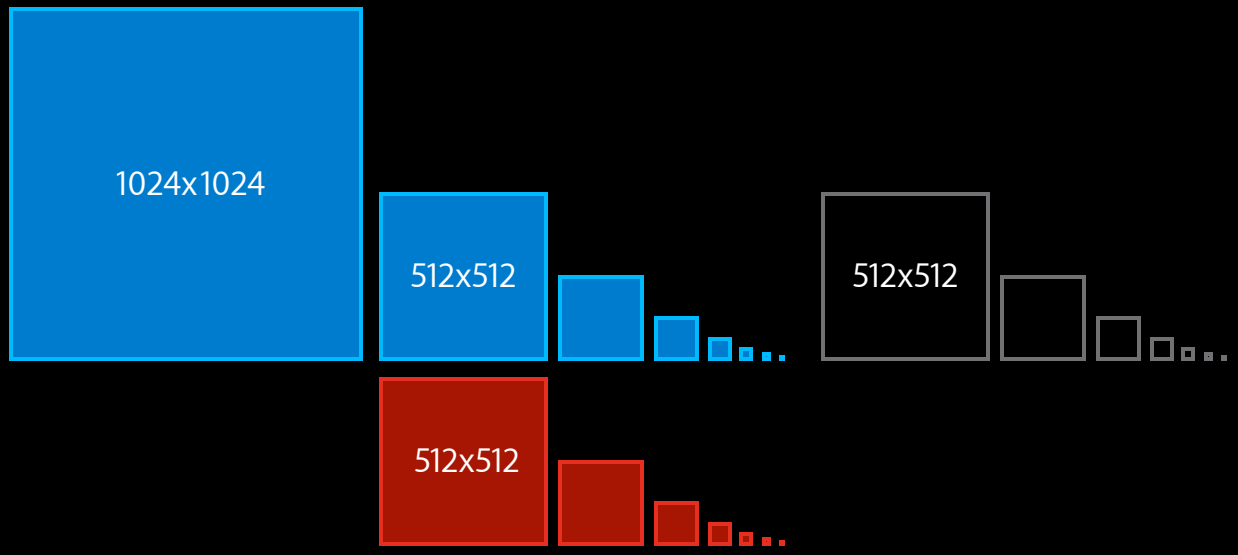

```
// Create temp texture
glGenTextures(1, &temp_texture);
glBindTexture(GL_TEXTURE_2D, temp_texture);
glTexStorage2DEXT(GL_TEXTURE_2D, 10, GL_RGBA8_OES, 512, 512);
```

Fine-Grain Texture Copy



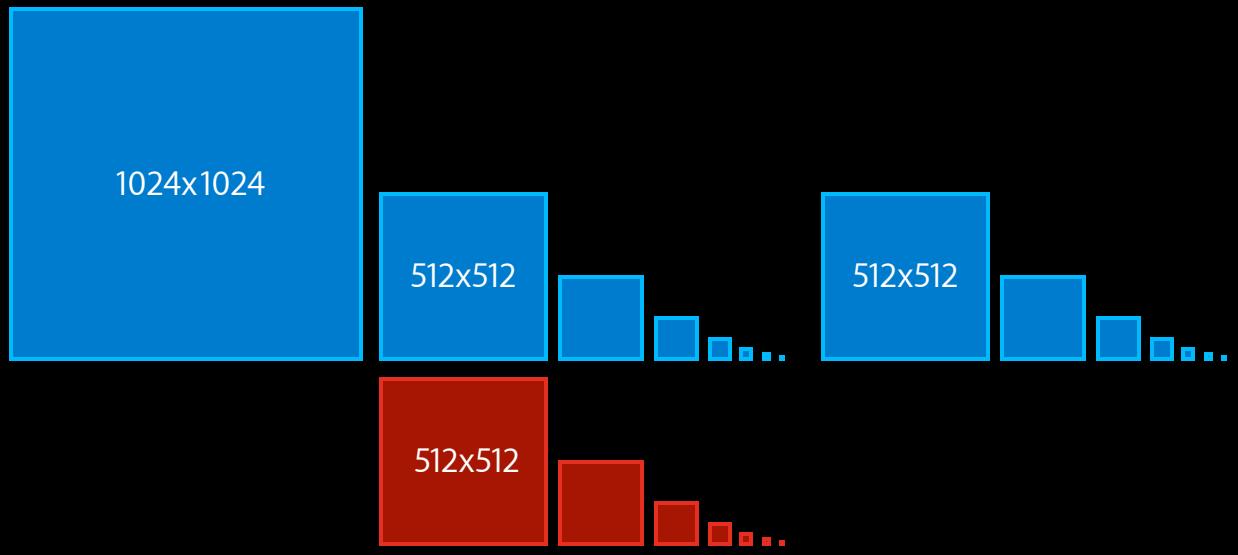

```
// Create temp texture
glGenTextures(1, &temp_texture);
glBindTexture(GL_TEXTURE_2D, temp_texture);
glTexStorage2DEXT(GL_TEXTURE_2D, 10, GL_RGBA8_OES, 512, 512);
```

Fine-Grain Texture Copy



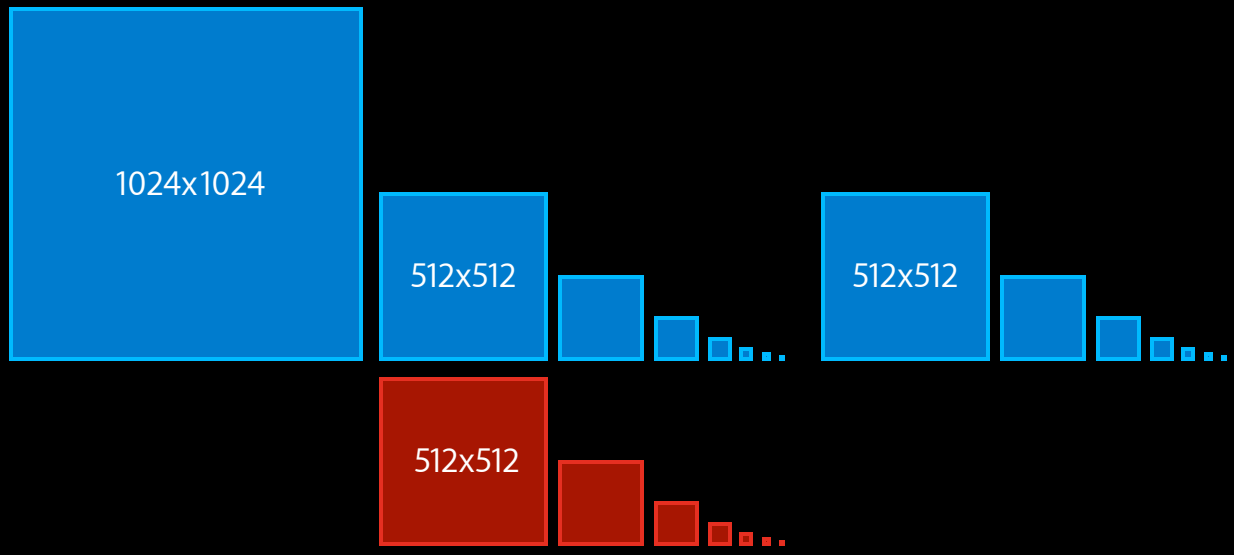

```
// Copy 10 “blue” levels starting at level 1 to temp
glCopyTextureLevelsAPPLE(temp_texture, blue_texture, 1, 10);
```

Fine-Grain Texture Copy



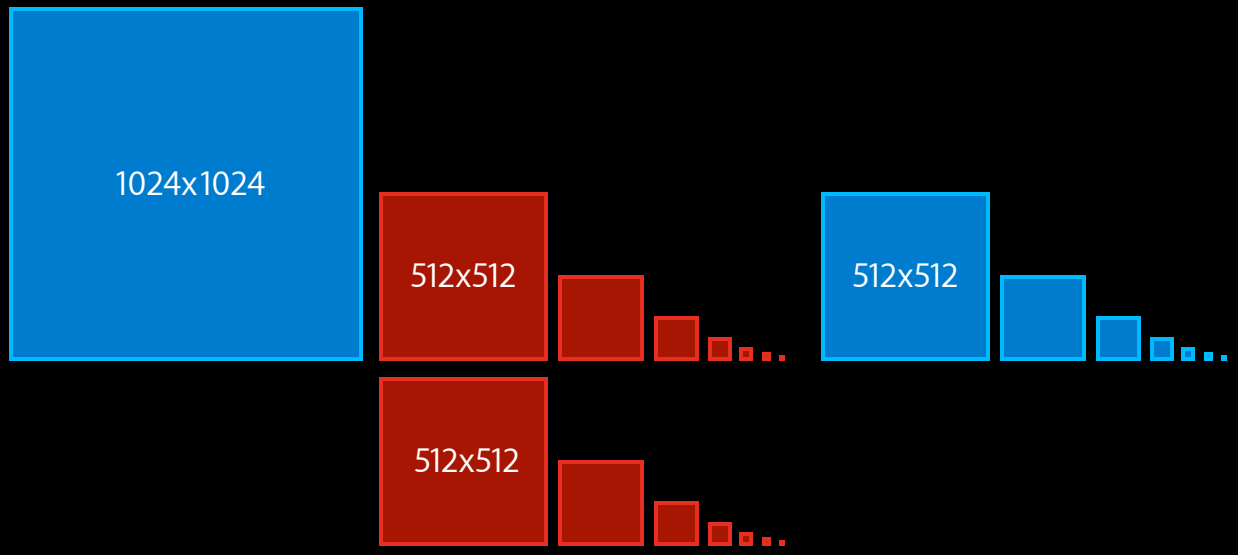

```
// Copy 10 “blue” levels starting at level 1 to temp
glCopyTextureLevelsAPPLE(temp_texture, blue_texture, 1, 10);
```

Fine-Grain Texture Copy



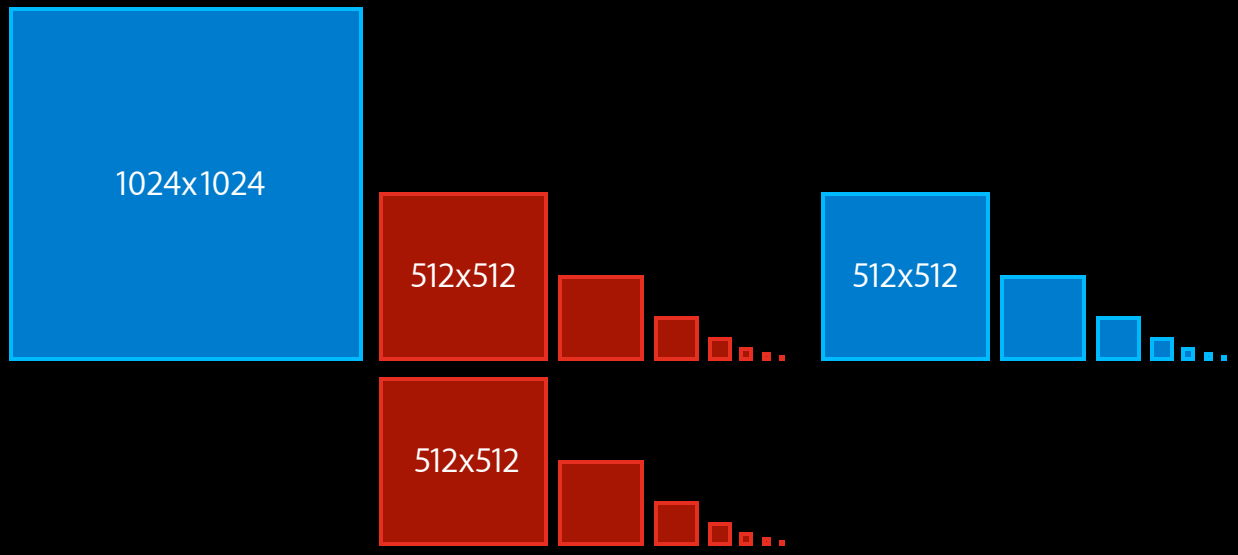

```
// Copy 10 “red” levels starting at level 0 to “blue”
glCopyTextureLevelsAPPLE(blue_texture, red_texture, 0, 10);
```

Fine-Grain Texture Copy



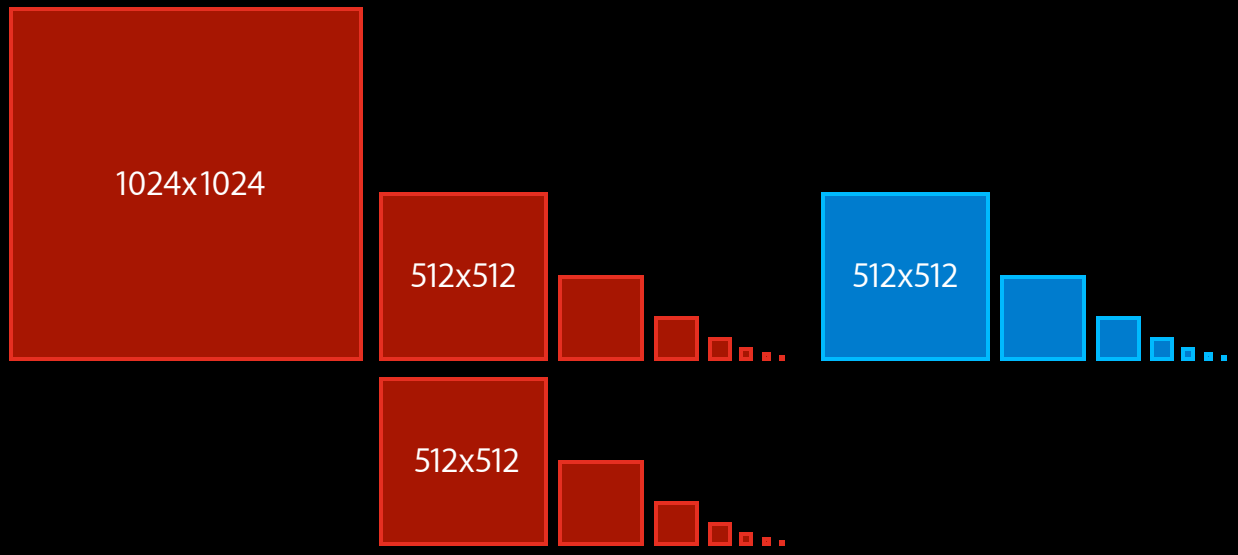

```
// Copy 10 “red” levels starting at level 0 to “blue”
glCopyTextureLevelsAPPLE(blue_texture, red_texture, 0, 10);
```

Fine-Grain Texture Copy



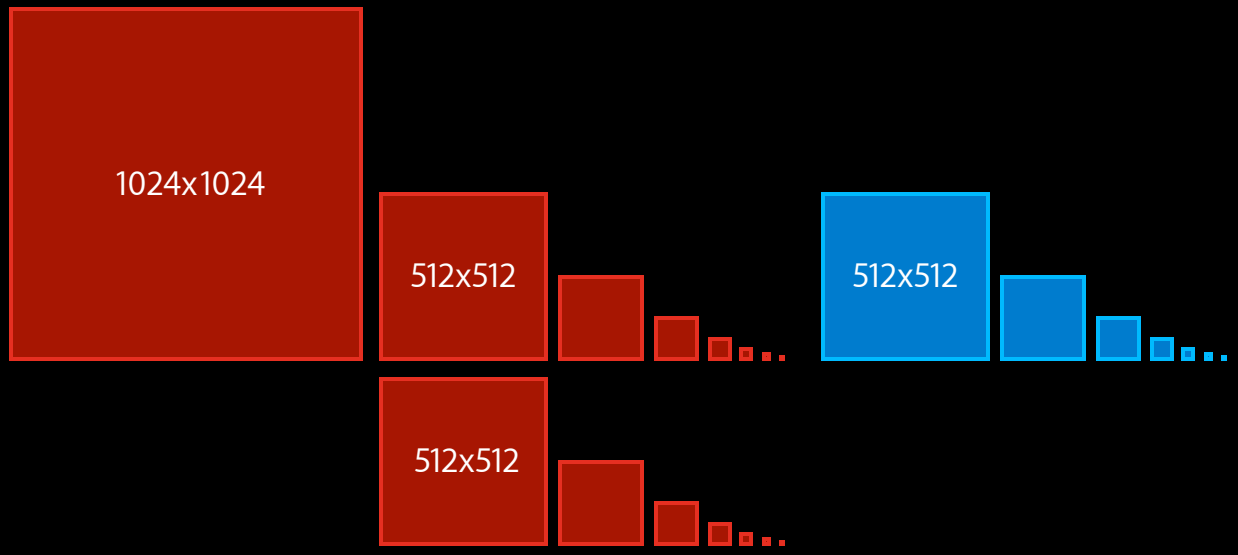

```
// upload new “red” level 0 data into “blue”
glBindTexture(GL_TEXTURE_2D, blue_texture);
glTexSubImage2D(GL_TEXTURE_2D, 0, ... , new_data);
```

Fine-Grain Texture Copy



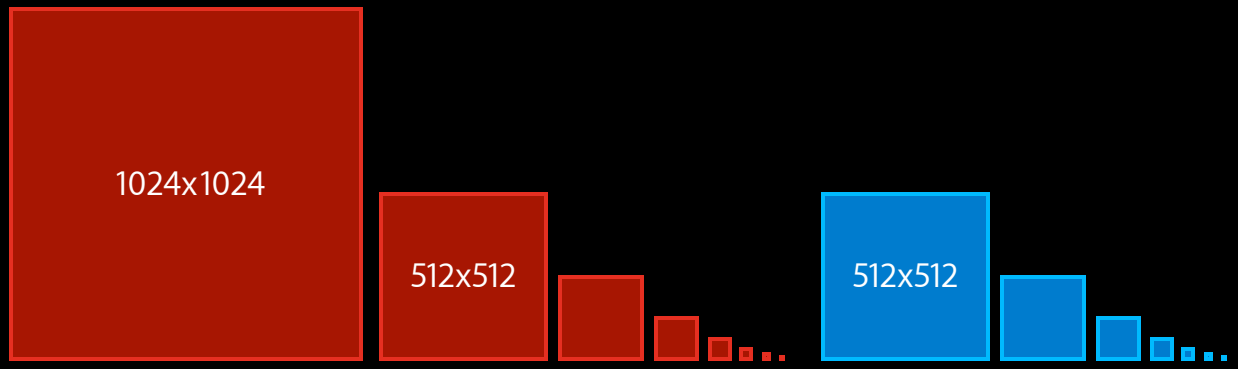

```
// upload new “red” level 0 data into “blue”
glBindTexture(GL_TEXTURE_2D, blue_texture);
glTexSubImage2D(GL_TEXTURE_2D, 0, ... , new_data);
```

Fine-Grain Texture Copy




```
// Delete “red” or keep for later re-use as temp
glDeleteTextures(red_texture);
```

Fine-Grain Texture Copy




```
// Delete “red” or keep for later re-use as temp
glDeleteTextures(red_texture);
```

Fast VBO Updates

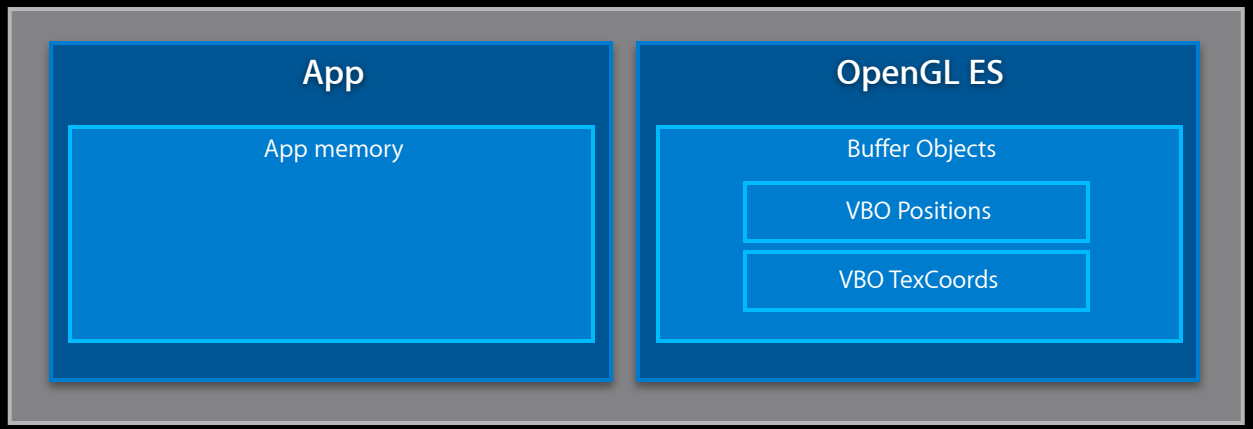
Vertex Buffer Objects

Vertex Buffer Objects are essential

- Fundamental method of defining geometry
  - Vertices, normals, colors, texture coordinates
- High performance
  - Buffer object data directly addressable by GPU
- Supported on all devices
  - Required by Core Profile on OS X

Dynamic VBO Updates

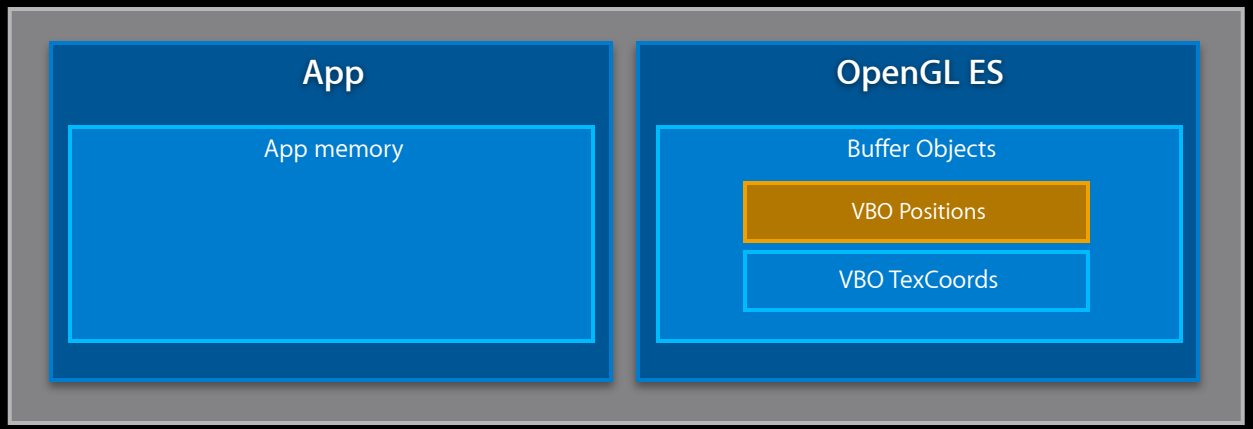
Typical example




```
glBindBuffer(GL_ARRAY_BUFFER, vbo_positions);
void *data = glMapBufferOES(GL_ARRAY_BUFFER, GL_WRITE_ONLY);
memcpy(&data[offset], new_data, length);
success = glUnmapBufferOES(GL_ARRAY_BUFFER);
```

Dynamic VBO Updates

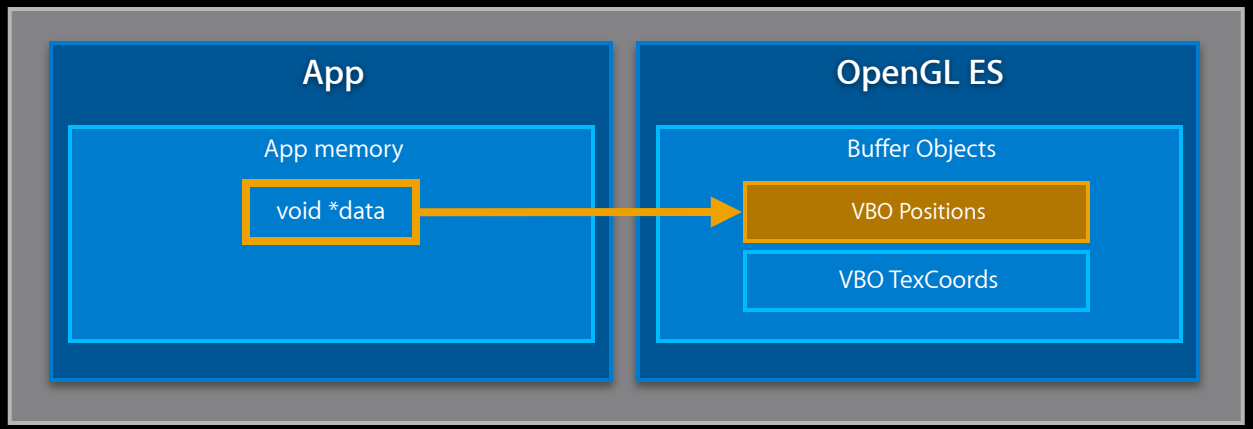
Typical example




```
glBindBuffer(GL_ARRAY_BUFFER, vbo_positions);
void *data = glMapBufferOES(GL_ARRAY_BUFFER, GL_WRITE_ONLY);
memcpy(&data[offset], new_data, length);
success = glUnmapBufferOES(GL_ARRAY_BUFFER);
```

Dynamic VBO Updates

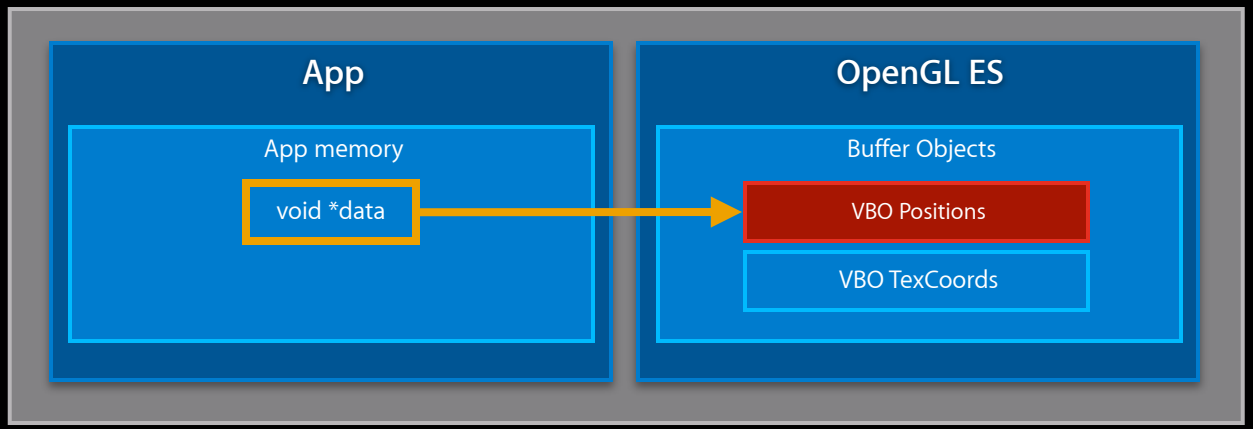
Typical example




```
glBindBuffer(GL_ARRAY_BUFFER, vbo_positions);
void *data = glMapBufferOES(GL_ARRAY_BUFFER, GL_WRITE_ONLY);
memcpy(&data[offset], new_data, length);
success = glUnmapBufferOES(GL_ARRAY_BUFFER);
```

Dynamic VBO Updates

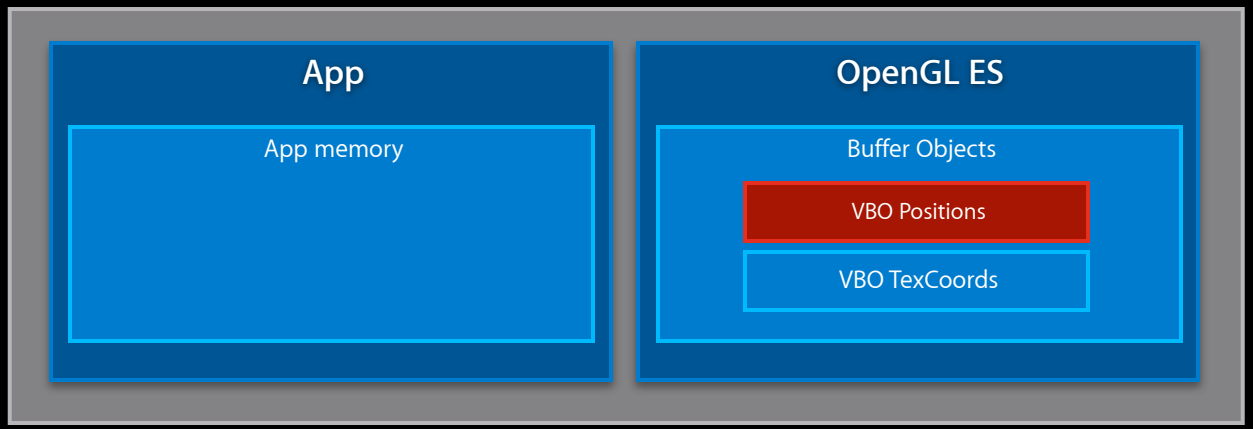
Typical example




```
glBindBuffer(GL_ARRAY_BUFFER, vbo_positions);
void *data = glMapBufferOES(GL_ARRAY_BUFFER, GL_WRITE_ONLY);
memcpy(&data[offset], new_data, length);
success = glUnmapBufferOES(GL_ARRAY_BUFFER);
```

Dynamic VBO Updates

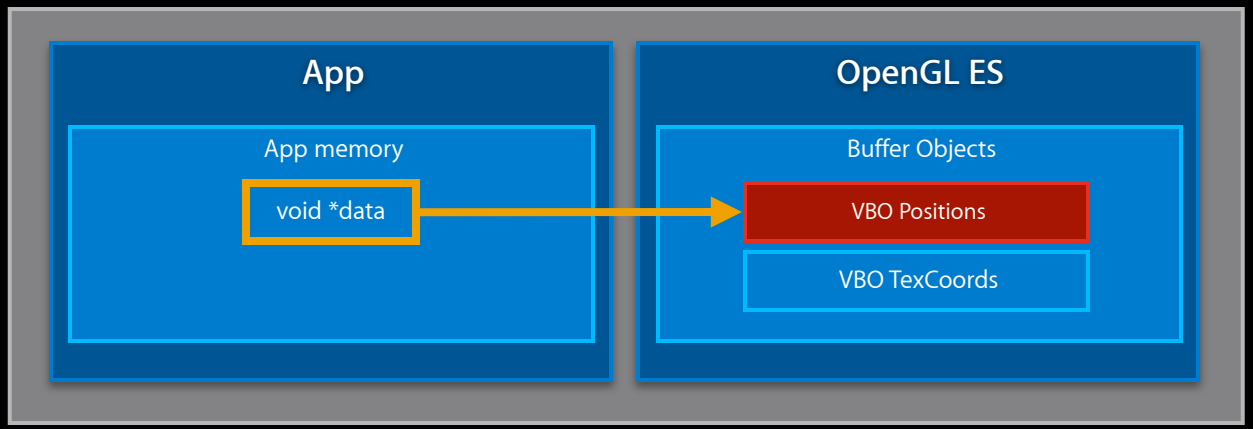
Typical example




```
glBindBuffer(GL_ARRAY_BUFFER, vbo_positions);
void *data = glMapBufferOES(GL_ARRAY_BUFFER, GL_WRITE_ONLY);
memcpy(&data[offset], new_data, length);
success = glUnmapBufferOES(GL_ARRAY_BUFFER);
```

Dynamic VBO Updates

Two issues



- Map can force GPU to sync with CPU
  - CPU waits for draw to finish before update
- Unmap requires CPU memory caches to be flushed
  - To ensure all modifications visible to GPU

Fast VBO Updates

APPLE_map_buffer_range and APPLE_sync

- APPLE_map_buffer_range
  - Provides explicit sub-range flushing
    - Specify subrange that's been modified
    - Identifies data to be flushed
  - Enables asynchronous buffer modification
- APPLE_sync
  - Provides sync object
  - Know when commands are completed
- Supported on all iOS 6 devices

Fast VBO Updates

APPLE_map_buffer_range example

```
// Bind vertex positions buffer
glBindBuffer(GL_ARRAY_BUFFER, vbo_positions);

// Get pointer to exact sub-range to modify
void *data = glMapBufferRangeAPPLE(GL_ARRAY_BUFFER, offset, length,
    GL_MAP_WRITE_BIT_APPLE | GL_MAP_FLUSH_EXPLICIT_BIT_APPLE );

// Copy new data
memcpy(data, new_data, length);

// Flush mapped sub-range and un-map
glFlushMappedBufferRangeAPPLE(GL_ARRAY_BUFFER, 0, length);
success = glUnmapBufferOES(GL_ARRAY_BUFFER);
```

Fast VBO Updates

APPLE_map_buffer_range example

```
// Bind vertex positions buffer
glBindBuffer(GL_ARRAY_BUFFER, vbo_positions);

// Get pointer to exact sub-range to modify
void *data = glMapBufferRangeAPPLE(GL_ARRAY_BUFFER, offset, length,
    GL_MAP_WRITE_BIT_APPLE | GL_MAP_FLUSH_EXPLICIT_BIT_APPLE );

// Copy new data
memcpy(data, new_data, length);

// Flush mapped sub-range and un-map
glFlushMappedBufferRangeAPPLE(GL_ARRAY_BUFFER, 0, length);
success = glUnmapBufferOES(GL_ARRAY_BUFFER);
```

Fast VBO Updates

APPLE_map_buffer_range example

```
// Bind vertex positions buffer
glBindBuffer(GL_ARRAY_BUFFER, vbo_positions);

// Get pointer to exact sub-range to modify
void *data = glMapBufferRangeAPPLE(GL_ARRAY_BUFFER, offset, length,
    GL_MAP_WRITE_BIT_APPLE | GL_MAP_FLUSH_EXPLICIT_BIT_APPLE );

// Copy new data
memcpy(data, new_data, length);

// Flush mapped sub-range and un-map
glFlushMappedBufferRangeAPPLE(GL_ARRAY_BUFFER, 0, length);
success = glUnmapBufferOES(GL_ARRAY_BUFFER);
```

Fast VBO Updates

APPLE_map_buffer_range example

```
// Bind vertex positions buffer
glBindBuffer(GL_ARRAY_BUFFER, vbo_positions);

// Get pointer to exact sub-range to modify
void *data = glMapBufferRangeAPPLE(GL_ARRAY_BUFFER, offset, length,
    GL_MAP_WRITE_BIT_APPLE | GL_MAP_FLUSH_EXPLICIT_BIT_APPLE );

// Copy new data
memcpy(data, new_data, length);

// Flush mapped sub-range and un-map
glFlushMappedBufferRangeAPPLE(GL_ARRAY_BUFFER, 0, length);
success = glUnmapBufferOES(GL_ARRAY_BUFFER);
```

Fast VBO Updates

APPLE_map_buffer_range example

```
// Bind vertex positions buffer
glBindBuffer(GL_ARRAY_BUFFER, vbo_positions);

// Get pointer to exact sub-range to modify
void *data = glMapBufferRangeAPPLE(GL_ARRAY_BUFFER, offset, length,
    GL_MAP_WRITE_BIT_APPLE | GL_MAP_FLUSH_EXPLICIT_BIT_APPLE );

// Copy new data
memcpy(data, new_data, length);

// Flush mapped sub-range and un-map
glFlushMappedBufferRangeAPPLE(GL_ARRAY_BUFFER, 0, length);
success = glUnmapBufferOES(GL_ARRAY_BUFFER);
```

Fast VBO Updates

```
// Bind and Map buffer
glBindBuffer(GL_ARRAY_BUFFER, this_vbo);
void *data = glMapBufferRangeAPPLE(GL_ARRAY_BUFFER, offset, length,
    GL_MAP_WRITE_BIT_APPLE | GL_MAP_FLUSH_EXPLICIT_BIT_APPLE );
```

```
// Modify buffer, flush, unmap
memcpy(data, new_data, length);
glFlushMappedBufferRangeAPPLE(GL_ARRAY_BUFFER, 0, length);
success = glUnmapBufferOES(GL_ARRAY_BUFFER);

// Issue draw commands
drawThisVBO(this_vbo);
```

Fast VBO Updates

```
// Bind and Map buffer
glBindBuffer(GL_ARRAY_BUFFER, this_vbo);
void *data = glMapBufferRangeAPPLE(GL_ARRAY_BUFFER, offset, length,
    GL_MAP_WRITE_BIT_APPLE | GL_MAP_FLUSH_EXPLICIT_BIT_APPLE );
```

```
// Modify buffer, flush, unmap
memcpy(data, new_data, length);
glFlushMappedBufferRangeAPPLE(GL_ARRAY_BUFFER, 0, length);
success = glUnmapBufferOES(GL_ARRAY_BUFFER);

// Issue draw commands
drawThisVBO(this_vbo);
```

Fast VBO Updates

```
// Bind and Map buffer
glBindBuffer(GL_ARRAY_BUFFER, this_vbo);
void *data = glMapBufferRangeAPPLE(GL_ARRAY_BUFFER, offset, length,
    GL_MAP_WRITE_BIT_APPLE | GL_MAP_FLUSH_EXPLICIT_BIT_APPLE |
    GL_MAP_UNSYNCHRONIZED_BIT_APPLE );
```

```
// Modify buffer, flush, unmap
memcpy(data, new_data, length);
glFlushMappedBufferRangeAPPLE(GL_ARRAY_BUFFER, 0, length);
success = glUnmapBufferOES(GL_ARRAY_BUFFER);

// Issue draw commands
drawThisVBO(this_vbo);
```

Fast VBO Updates

```
// Bind and Map buffer
glBindBuffer(GL_ARRAY_BUFFER, this_vbo);
void *data = glMapBufferRangeAPPLE(GL_ARRAY_BUFFER, offset, length,
    GL_MAP_WRITE_BIT_APPLE | GL_MAP_FLUSH_EXPLICIT_BIT_APPLE |
    GL_MAP_UNSYNCHRONIZED_BIT_APPLE );

// Wait for fence (set below) before modifying buffer
glClientWaitSyncAPPLE(fence, GL_SYNC_FLUSH_COMMANDS_BIT_APPLE,
     GL_TIMEOUT_IGNORED_APPLE);

// Modify buffer, flush, unmap
memcpy(data, new_data, length);
glFlushMappedBufferRangeAPPLE(GL_ARRAY_BUFFER, 0, length);
success = glUnmapBufferOES(GL_ARRAY_BUFFER);

// Issue draw commands, then insert fence
drawThisVBO(this_vbo);
fence = glFenceSyncAPPLE(GL_SYNC_GPU_COMMANDS_COMPLETE_APPLE, 0);
```

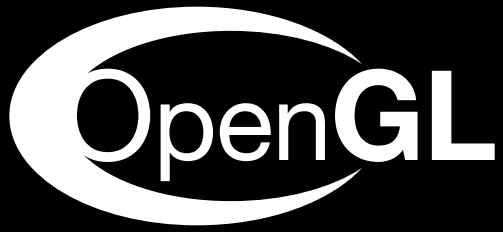
Going Further

Fast nonblocking VBO updates

- Tuning OpenGL ES Games
  - iOS Tech Talk 2011
  - Creating combined arrays and flattening transformations in 2D games
    - http://developer.apple.com/videos/iOS

GLKit


OpenGL|ES


OpenGL


GL Kit

- Create modern GL apps easily
- Move to the shader pipeline
- Gateway to more complex effects
- Provided for iOS and OS X

GLKEffects

Great visual effects with minimal effort

- GLKBaseEffect
  - Bridge from fixed-function vertex and fragment processing
    - Mimics fixed-function lighting, textures, transformations, etc.
    - Uses ES 2.0 shader pipeline on iOS
    - Uses Core Profile on OS X
- GLKReflectionMapEffect
  - Cube map reflection class
- GLKSkyboxEffect
  - Textured backdrop enclosing scene

GLKView and GLKViewController

Overview

- GLKView
  - Provides OpenGL ES compatible view
  - Easy renderbuffer setup
  - Straightforward draw methods
- GLKViewController
  - Provides render loop synchronized to display updates
  - Handles device orientation changes
  - Pause when transitioning to background
  - Frame timing information
- OS X: NSOpenGLView

GLKTextureLoader

Overview

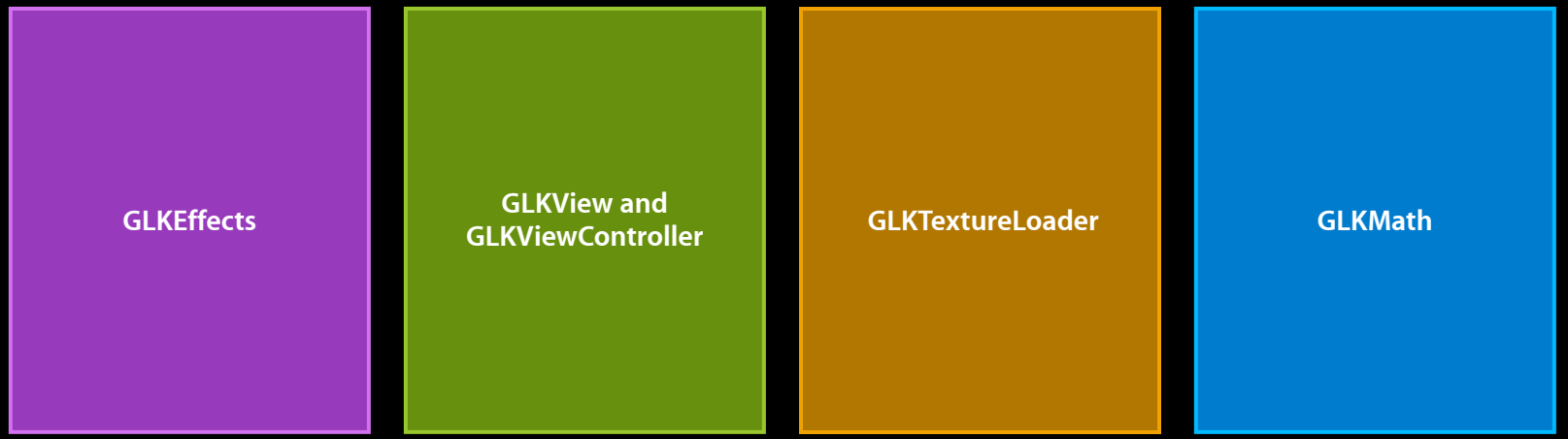
- Texture loading made simple
  - Many common image formats: PNG, JPEG, TIFF, etc.
- Flexible
  - Load textures from file, URL, NSData, CGImageRef
  - 2D or cubemap textures
  - Synchronous or asynchronous loading
- Convenient loading options
  - Generate mipmaps
  - Flip origin
  - Premultiply alpha

GLKMath

3D graphics math library

- Over 175 math functions
  - 2, 3, and 4 component vectors
  - 4x4 and 3x3 matrix type
  - Quaternions
- High performance, C-based, inline
  - iOS: Scalar and NEON implementations
  - OS X: Intel SSE-optimized
- Matrix stack library
  - Easily maintain projection and modelview matrix stack
  - Simplify migration

Going Further



Harnessing GLKit and OpenGL ES

- http://developer.apple.com/videos/ios/

Supporting Retina Displays

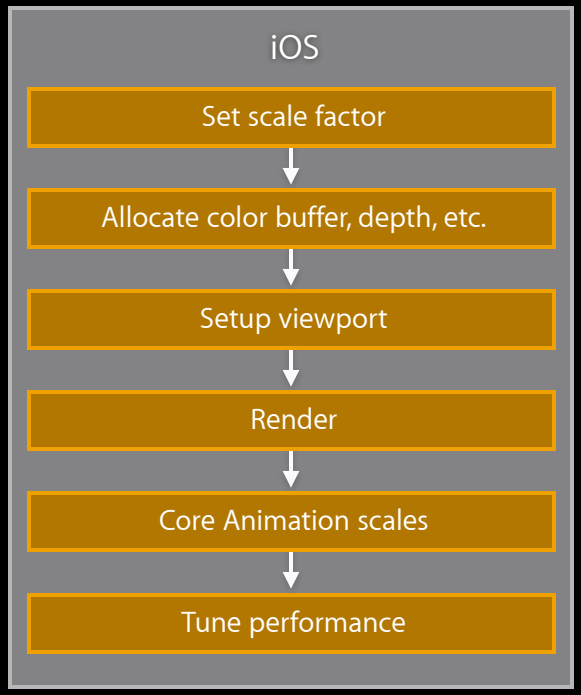
Supporting Retina Displays

Supporting Retina Displays

Very similar approaches

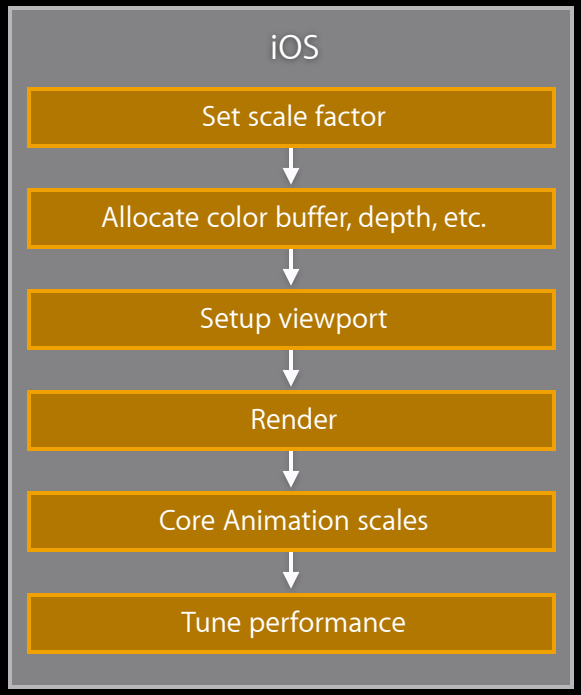
Supporting Retina Displays

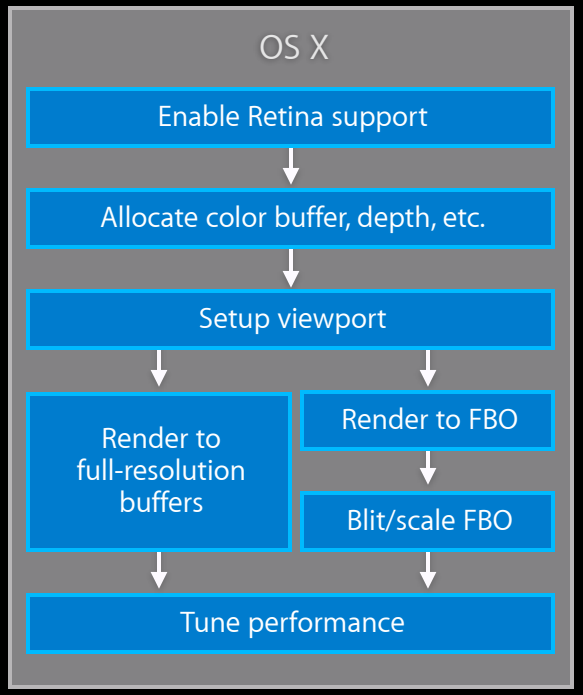
Very similar approaches



Supporting Retina Displays

Very similar approaches





Supporting Retina Displays

Enable Retina support

```
// NSOpenGLView subclass
- (id) initWithFrame:(NSRect)rect
{
    // Enable Retina support
    [self setWantsBestResolutionOpenGLSurface:YES];

    ...
    return self;
}
```

Supporting Retina Displays

Enable Retina support

```
// NSOpenGLView subclass
- (id) initWithFrame:(NSRect)rect
{
    // Enable Retina support
    [self setWantsBestResolutionOpenGLSurface:YES];

    ...
    return self;
}
```

Supporting Retina Displays

Set up the viewport and draw

```
- (void)drawRect:(NSRect)rect   // NSOpenGLView subclass
{
    // Get view dimensions in pixels
    NSRect backingBounds = [self convertRectToBacking:[self bounds]];

    GLsizei backingPixelWidth  = (GLsizei)(backingBounds.size.width),
            backingPixelHeight = (GLsizei)(backingBounds.size.height);

    // Set viewport
    glViewport(0, 0, backingPixelWidth, backingPixelHeight);

    // draw…
}
```

Supporting Retina Displays

Set up the viewport and draw

```
- (void)drawRect:(NSRect)rect   // NSOpenGLView subclass
{
    // Get view dimensions in pixels
    NSRect backingBounds = [self convertRectToBacking:[self bounds]];

    GLsizei backingPixelWidth  = (GLsizei)(backingBounds.size.width),
            backingPixelHeight = (GLsizei)(backingBounds.size.height);

    // Set viewport
    glViewport(0, 0, backingPixelWidth, backingPixelHeight);

    // draw…
}
```

Supporting Retina Displays

Set up the viewport and draw

```
- (void)drawRect:(NSRect)rect   // NSOpenGLView subclass
{
    // Get view dimensions in pixels
    NSRect backingBounds = [self convertRectToBacking:[self bounds]];

    GLsizei backingPixelWidth  = (GLsizei)(backingBounds.size.width),
            backingPixelHeight = (GLsizei)(backingBounds.size.height);

    // Set viewport
    glViewport(0, 0, backingPixelWidth, backingPixelHeight);

    // draw…
}
```

Supporting Retina Displays

Set up the viewport and draw

- Check for calls defined in pixel dimensions

```
glViewport (GLint x, GLint y, GLsizei width, GLsizei height)
glScissor (GLint x, GLint y, GLsizei width, GLsizei height)
glReadPixels (GLint x, GLint y, GLsizei width, GLsizei height, ...)
glLineWidth (GLfloat width)
glRenderbufferStorage (..., GLsizei width, GLsizei height)
```

- Use higher resolution assets

```
glTexImage2D (..., GLsizei width, GLsizei height, ...)
```

Supporting Retina Displays

Blit FBO to backing buffer

```
-(void)presentFBO:(GLint)fbo width:(GLint)fbWidth height:(GLint)fbHeight {

    // Drawing finished, bind FBO for reading and back buffer for writing
    glBindFramebuffer(GL_READ_FRAMEBUFFER, fboName);
    glReadBuffer(GL_COLOR_ATTACHMENT0);     // Source from 1st attachment

    glBindFramebuffer(GL_DRAW_FRAMEBUFFER, 0);
    glDrawBuffer(GL_BACK);                  // Copy into drawable backing

    glBlitFramebuffer( 0, 0, fbWidth,       fbHeight,
                       0, 0, drawableWidth, drawableHeight,
                       GL_COLOR_BUFFER_BIT, GL_LINEAR );

    [[oglView openGLContext] flushBuffer];  // Swap onscreen
    glDrawBuffer(GL_COLOR_ATTACHMENT0);     // Optional
}
```

Supporting Retina Displays

Blit FBO to backing buffer

```
-(void)presentFBO:(GLint)fbo width:(GLint)fbWidth height:(GLint)fbHeight {

    // Drawing finished, bind FBO for reading and back buffer for writing
    glBindFramebuffer(GL_READ_FRAMEBUFFER, fboName);
    glReadBuffer(GL_COLOR_ATTACHMENT0);     // Source from 1st attachment

    glBindFramebuffer(GL_DRAW_FRAMEBUFFER, 0);
    glDrawBuffer(GL_BACK);                  // Copy into drawable backing

    glBlitFramebuffer( 0, 0, fbWidth,       fbHeight,
                       0, 0, drawableWidth, drawableHeight,
                       GL_COLOR_BUFFER_BIT, GL_LINEAR );

    [[oglView openGLContext] flushBuffer];  // Swap onscreen
    glDrawBuffer(GL_COLOR_ATTACHMENT0);     // Optional
}
```

Supporting Retina Displays

Blit FBO to backing buffer

```
-(void)presentFBO:(GLint)fbo width:(GLint)fbWidth height:(GLint)fbHeight {

    // Drawing finished, bind FBO for reading and back buffer for writing
    glBindFramebuffer(GL_READ_FRAMEBUFFER, fboName);
    glReadBuffer(GL_COLOR_ATTACHMENT0);     // Source from 1st attachment

    glBindFramebuffer(GL_DRAW_FRAMEBUFFER, 0);
    glDrawBuffer(GL_BACK);                  // Copy into drawable backing

    glBlitFramebuffer( 0, 0, fbWidth,       fbHeight,
                       0, 0, drawableWidth, drawableHeight,
                       GL_COLOR_BUFFER_BIT, GL_LINEAR );

    [[oglView openGLContext] flushBuffer];  // Swap onscreen
    glDrawBuffer(GL_COLOR_ATTACHMENT0);     // Optional
}
```

Supporting Retina Displays

Blit FBO to backing buffer

```
-(void)presentFBO:(GLint)fbo width:(GLint)fbWidth height:(GLint)fbHeight {

    // Drawing finished, bind FBO for reading and back buffer for writing
    glBindFramebuffer(GL_READ_FRAMEBUFFER, fboName);
    glReadBuffer(GL_COLOR_ATTACHMENT0);     // Source from 1st attachment

    glBindFramebuffer(GL_DRAW_FRAMEBUFFER, 0);
    glDrawBuffer(GL_BACK);                  // Copy into drawable backing

    glBlitFramebuffer( 0, 0, fbWidth,       fbHeight,
                       0, 0, drawableWidth, drawableHeight,
                       GL_COLOR_BUFFER_BIT, GL_LINEAR );

    [[oglView openGLContext] flushBuffer];  // Swap onscreen
    glDrawBuffer(GL_COLOR_ATTACHMENT0);     // Optional
}
```

Supporting Retina Displays

Blit FBO to backing buffer

```
-(void)presentFBO:(GLint)fbo width:(GLint)fbWidth height:(GLint)fbHeight {

    // Drawing finished, bind FBO for reading and back buffer for writing
    glBindFramebuffer(GL_READ_FRAMEBUFFER, fboName);
    glReadBuffer(GL_COLOR_ATTACHMENT0);     // Source from 1st attachment

    glBindFramebuffer(GL_DRAW_FRAMEBUFFER, 0);
    glDrawBuffer(GL_BACK);                  // Copy into drawable backing

    glBlitFramebuffer( 0, 0, fbWidth,       fbHeight,
                       0, 0, drawableWidth, drawableHeight,
                       GL_COLOR_BUFFER_BIT, GL_LINEAR );

    [[oglView openGLContext] flushBuffer];  // Swap onscreen
    glDrawBuffer(GL_COLOR_ATTACHMENT0);     // Optional
}
```

Supporting Retina Displays

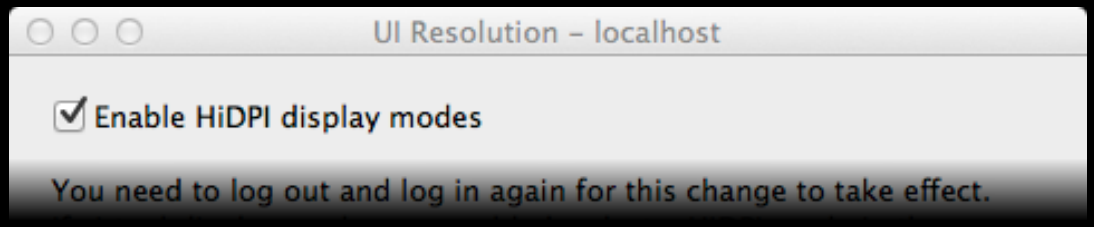
Tune performance

- Strategies for handling increased fragment workload
  - Tune: Render at 2x, optimize fragment shaders
  - Experiment: Render at 1x and enable anti-aliasing
  - Iterate: Try fractional sizes between 1x and 2x while optimizing shaders

Supporting Retina Displays

Going further

- Advanced topics
  - Scaling anti-aliased FBO
  - Handling multiple displays
  - Responding to resolution changes
- See GLFullScreen sample
- Get started today with `/Applications/Graphics Tools/Quartz Debug.app`



Wrap Up

Summary

- APPLE_shader_framebuffer_fetch
- EXT_texture_storage
- APPLE_copy_texture_levels
- APPLE_map_buffer_range
- APPLE_sync
- GLKit for iOS and OS X
- Supporting Retina displays

More Information

Allan Schaffer
Graphics and Game Technologies Evangelist
aschaffer@apple.com

Apple Developer Forums
http://devforums.apple.com/

Tuning OpenGL ES Games
http://developer.apple.com/videos/ios/

Harnessing GLKit and OpenGL ES
http://developer.apple.com/videos/ios/

GLFullScreen
http://developer.apple.com/library/mac/

Related Sessions

OpenGL ES Tools and Techniques	Pacific Heights Wednesday 3:15PM
Adopting OpenCL in Your Application	Nob Hill Thursday 4:30PM
Advanced Tips and Tricks for High Resolution on OS X	Mission Friday 10:15AM

Labs

OpenGL ES Lab	Graphics, Media, & Games Lab A Thursday 9:00AM
OpenGL Lab	Graphics, Media, & Games Lab B Thursday 9:00AM


WWDC2012

The last 3 slides after the logo are intentionally left blank for all presentations.

The last 3 slides after the logo are intentionally left blank for all presentations.

The last 3 slides after the logo are intentionally left blank for all presentations.