Core Image Techniques

Session 511

David Hayward
Advanced Imaging Team

These are confidential sessions—please refrain from streaming, blogging, or taking pictures

Introduction

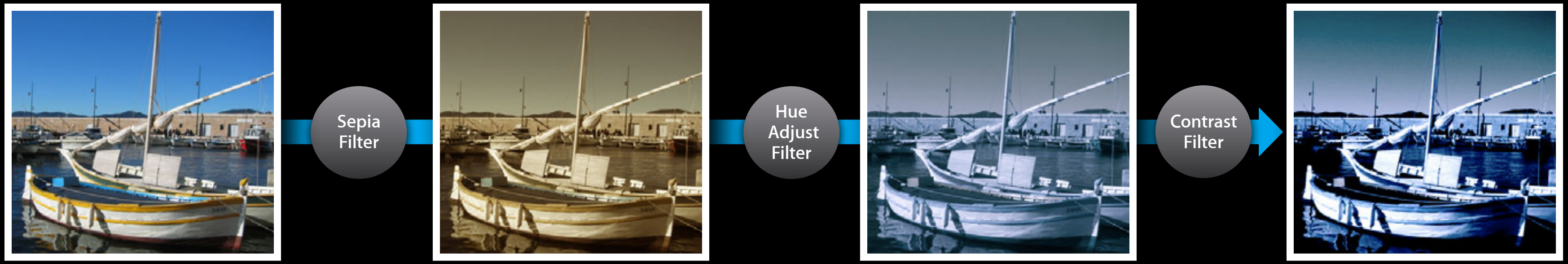
- Quick Introduction to Core Image
- A brief overview of what is new in iOS 6
- How to write a performant real-time camera app
- How to leverage OpenGL ES and Core Image simultaneously
- How to use Core Image to enhance your game

Quick Introduction to Core Image

Key concepts

Basic Concept

Filters can be chained together



Original

Sepia Filter

Hue Adjust Filter

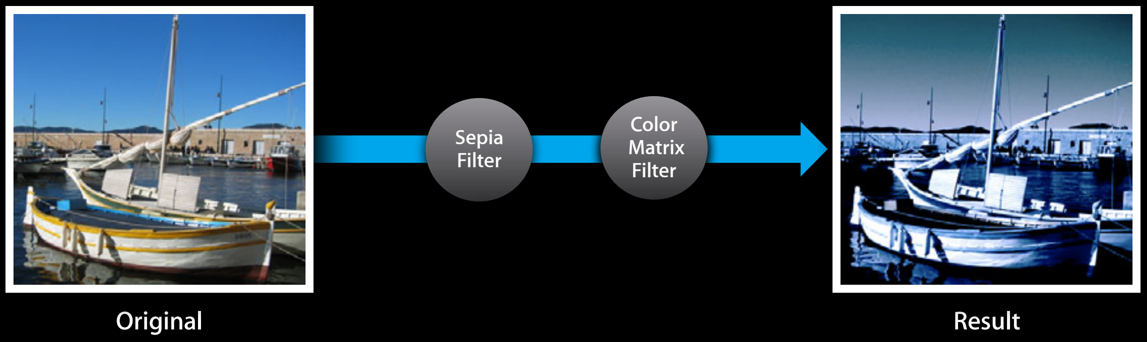
Contrast Filter

Result

This allows for complex effects

Basic Concept

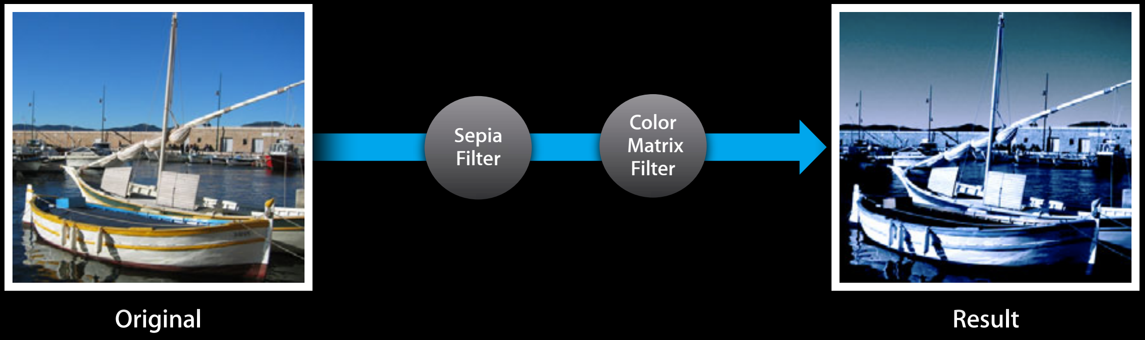
Filter chains are optimized at time of render



This greatly improves performance

Basic Concept

Filter chains are optimized at time of render



This greatly improves performance

Basic Concept

Flexible inputs

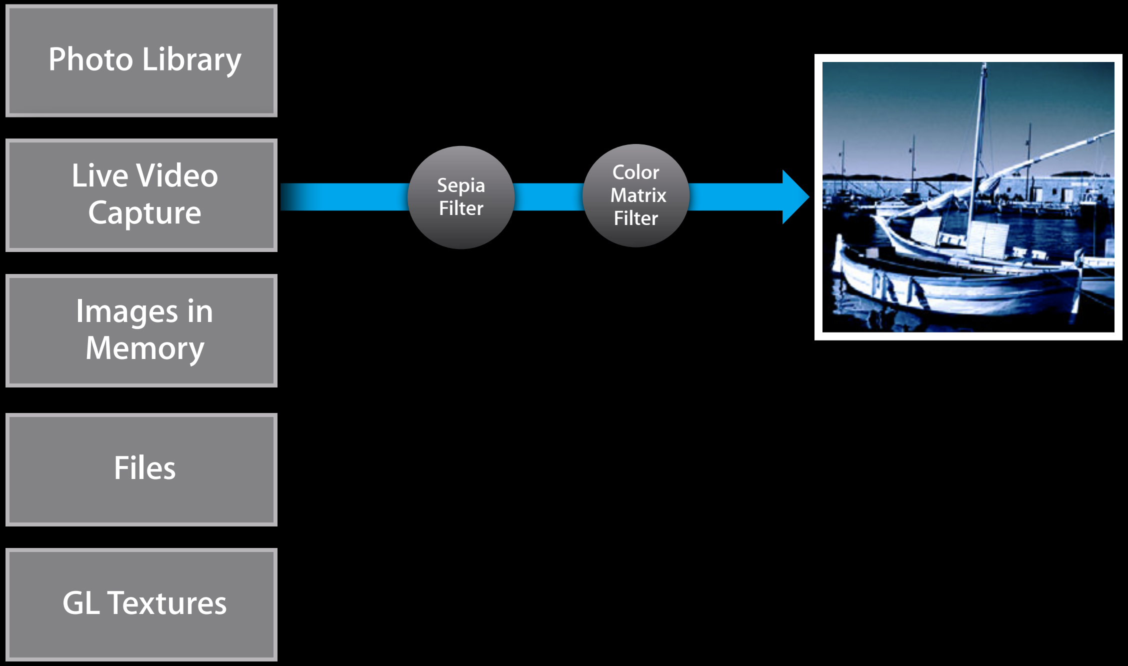


Photo Library

Live Video Capture

Images in Memory

Files

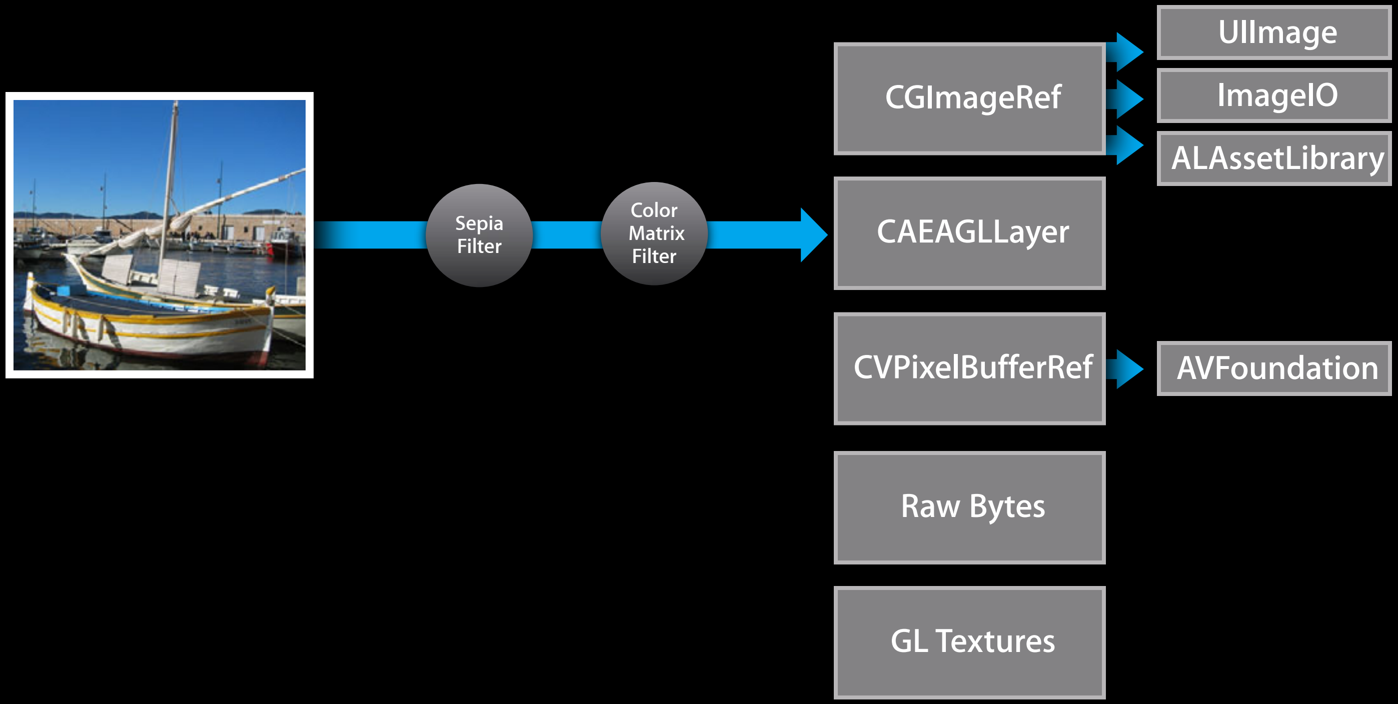
GL Textures

Sepia Filter

Color Matrix Filter

Basic Concept

Flexible outputs



Sepia Filter

Color Matrix Filter

CGImageRef

UIImage

ImageIO

ALAssetLibrary

CAEAGLLayer

CVPixelBufferRef

AVFoundation

Raw Bytes

GL Textures

Core Image Techniques

What’s new in iOS 6

Chendi Zhang
Employee

Core Image on iOS 6

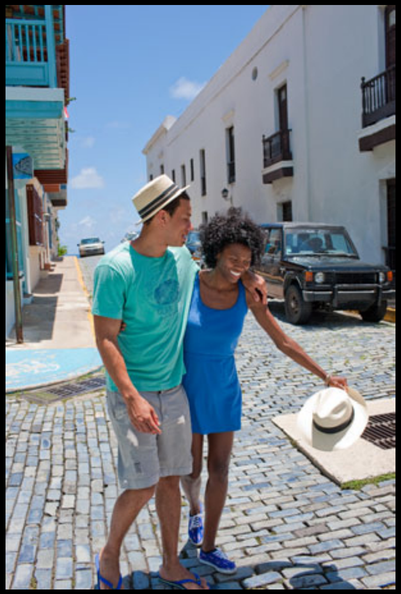
- Improved filter code generation
- Better OpenGL ES integration
- Many new filters
  - Gaussian blurs
  - Lanczos scaling

CIGaussianBlur

- The most requested filter addition
- Multipass filter: Can be quite expensive on large images
- Supports arbitrary blur radius sizes
- Basis of many other filters
    - CIBloom/CIGloom
    - CIUnsharpMask
    - CISharpenLuminance

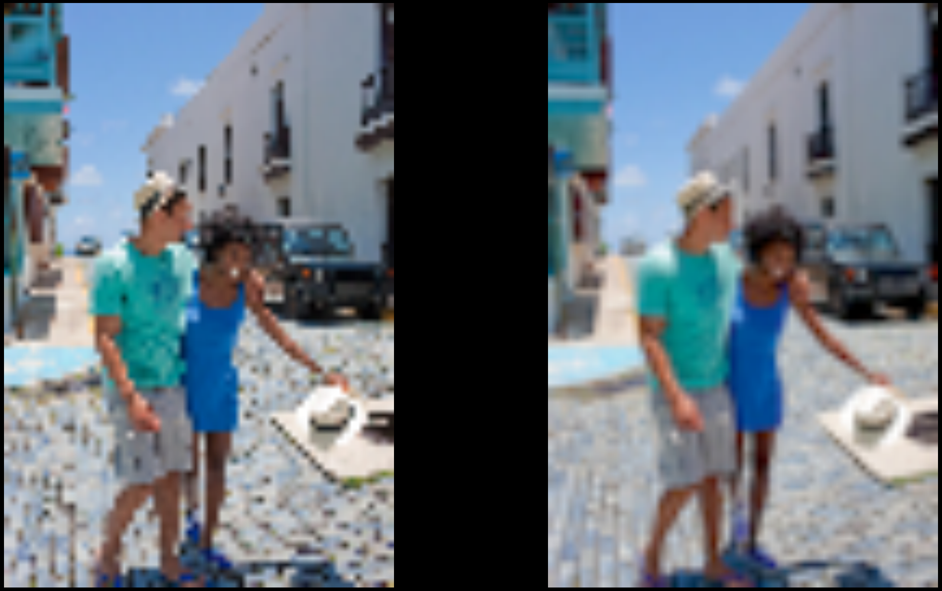
CILanczosScaleTransform

- Higher quality downsampling than OpenGL ES's bilinear
- Comparable with CG high-quality resampling



CILanczosScaleTransform

- Higher quality downsampling than OpenGL ES's bilinear
- Comparable with CG high-quality resampling



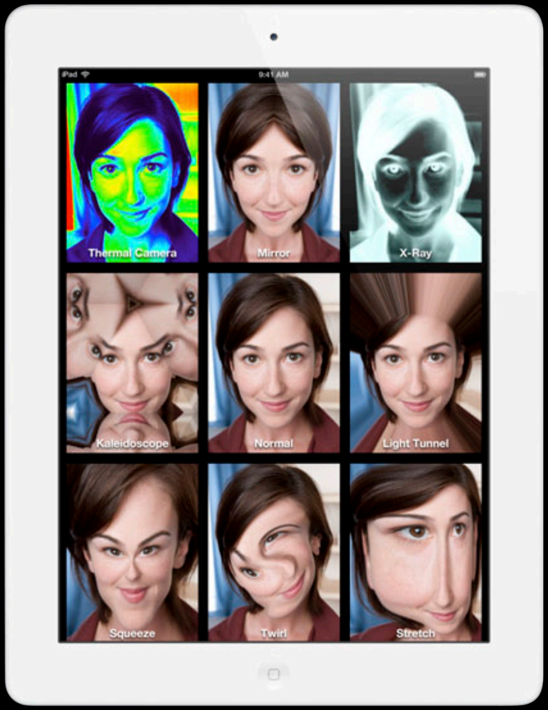
Core Image Techniques

Implementing real-time video effects

Chendi Zhang
Employee

Let’s Create a Photo/Video Effects App

- Photo apps are very popular
- GPUs are now fast enough to do complex real-time effects
- Live video preview is important for photo apps
- Core Image is perfect for this type of live image processing



Let’s Create a Photo/Video Effects App

The Effect Chain

How do we get a “vintage” look?

- Color transformation
- Vignette
- Film scratches
- Add a border

The Effect Chain

How do we get a “vintage” look?

- CIColorCube
- CIVignette
- CILightenBlendMode
- CISourceOverCompositing

The Effect Chain

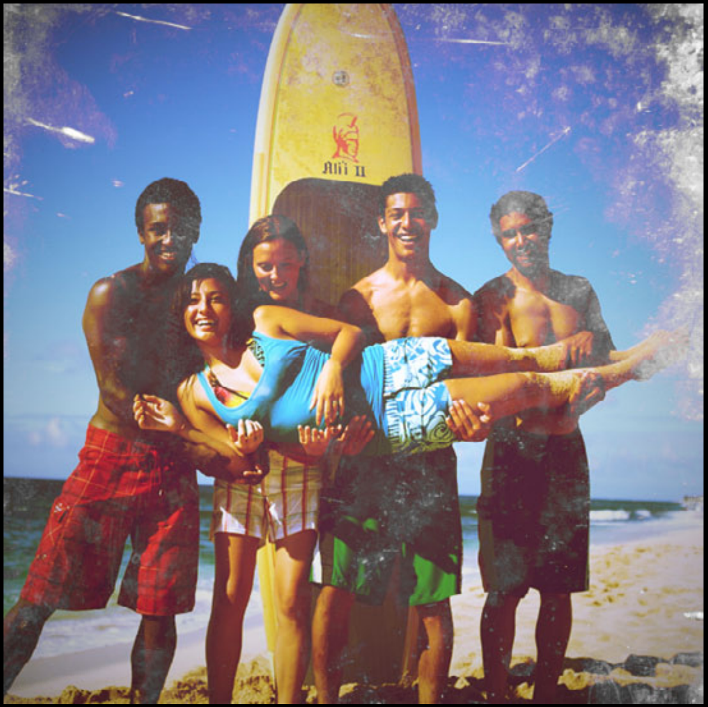
How do we get a “vintage” look?

- CIColorCube
- CIVignette
- CILightenBlendMode
- CISourceOverCompositing

The Effect Chain

How do we get a “vintage” look?

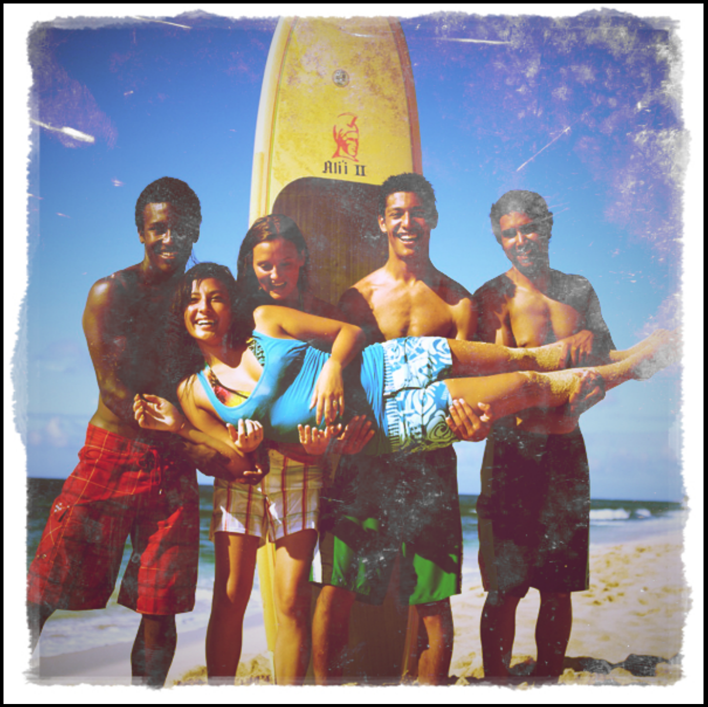
- CIColorCube
- CIVignette
- CILightenBlendMode
- CISourceOverCompositing



The Effect Chain

How do we get a “vintage” look?

- CIColorCube
- CIVignette
- CILightenBlendMode
- CISourceOverCompositing



Ready to Share

CIColorCube

- An extremely flexible filter
- Used for a variety of different color effects
- Often faster than algorithmic filters
- Supports up to a 64x64x64 cube

CIColorCube

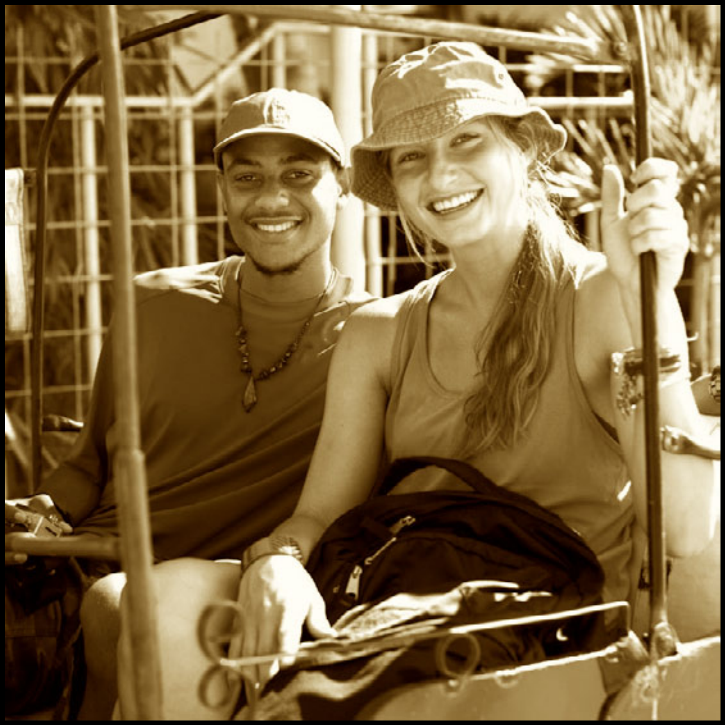
Approximating CISepiaTone

CISepia Tone

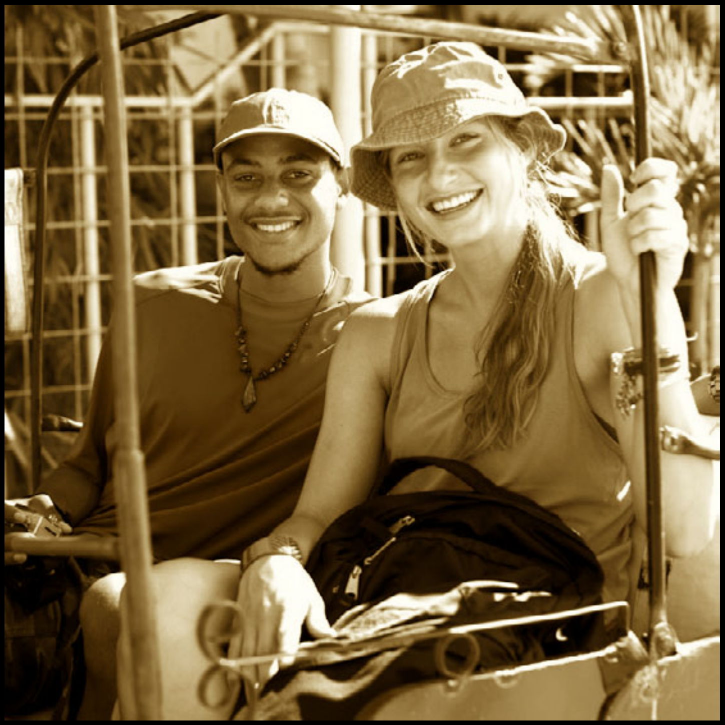
64x64x64

CIColorCube

Approximating CISepiaTone



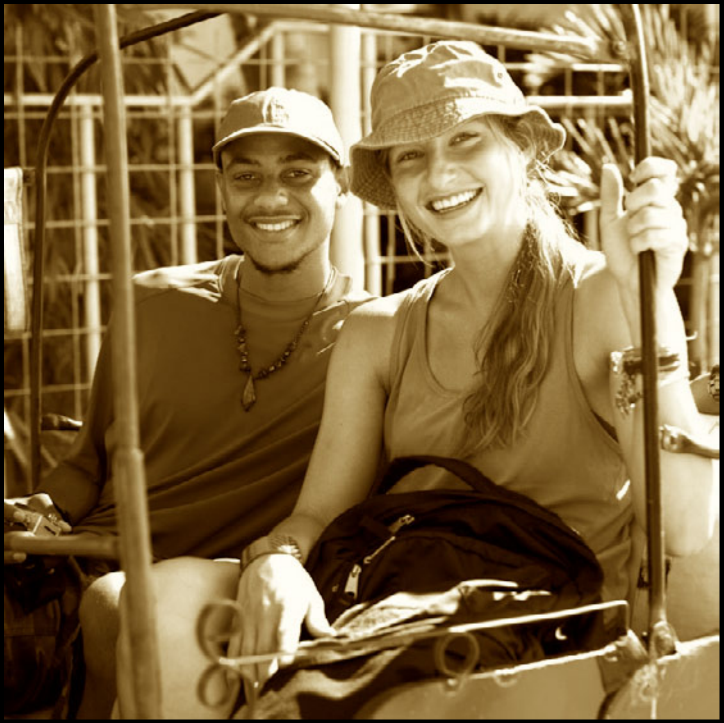
CISepia Tone



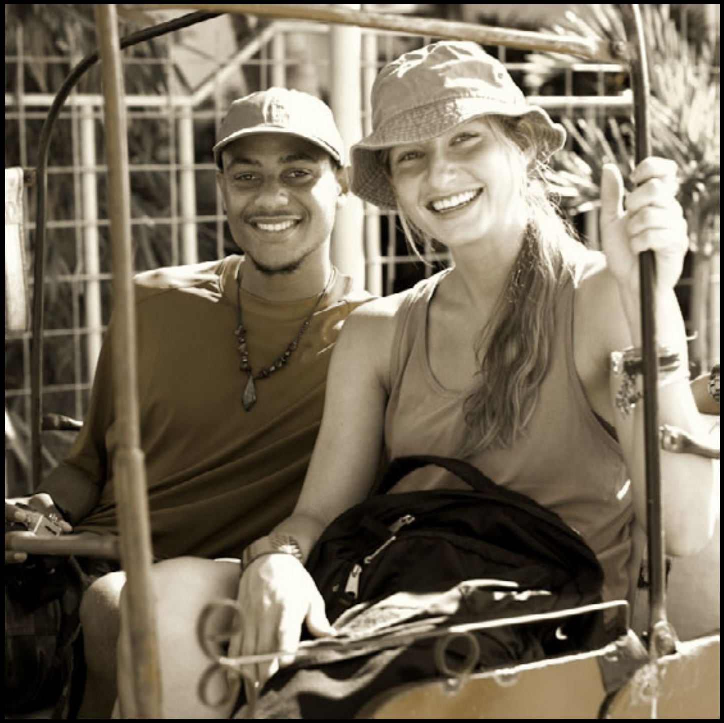
8x8x8

CIColorCube

Approximating CISepiaTone



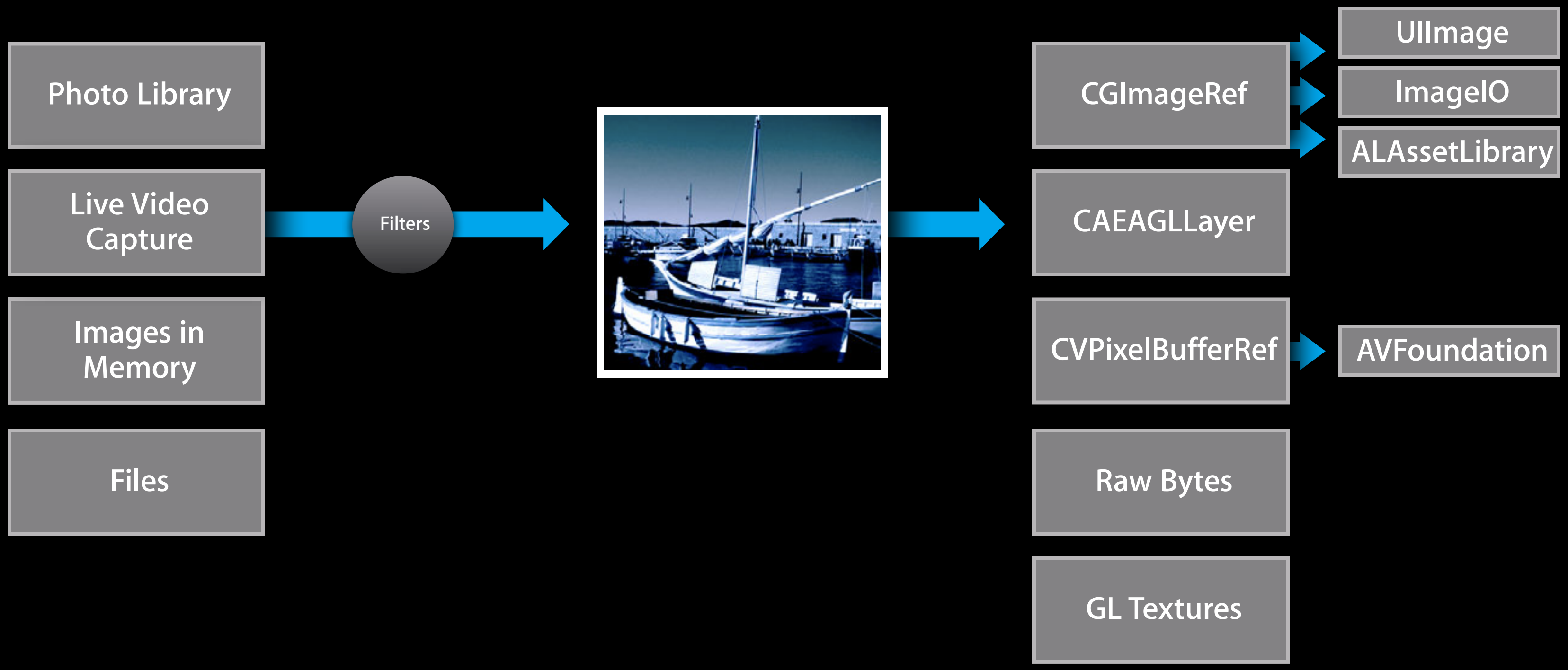
CISepia Tone



2x2x2

Creating a Real-Time Camera Effects App

Attempt 0



Creating a Real-Time Camera Effects App

Attempt 0

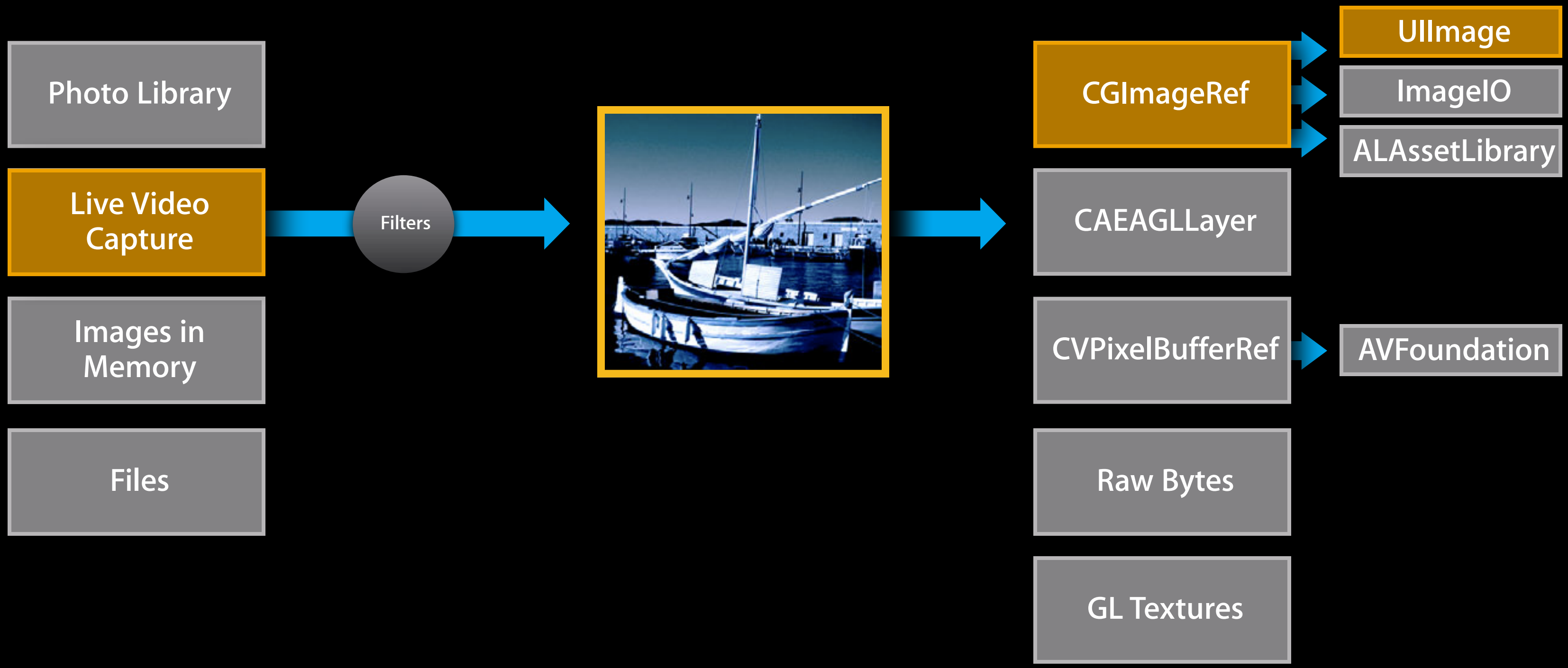


Photo Library

Live Video Capture

Images in Memory

Files

Filters

CGImageRef

CAEAGLLayer

CVPixelBufferRef

Raw Bytes

GL Textures

UIImage

ImageIO

ALAssetLibrary

AVFoundation

Demo

Why Was That Slow?

- UIImageView works best with static images
- Avoid creating a CIContext for each render
- Use lower-level APIs for performance-sensitive work

Creating a Real-Time Camera Effects App

Attempt 1

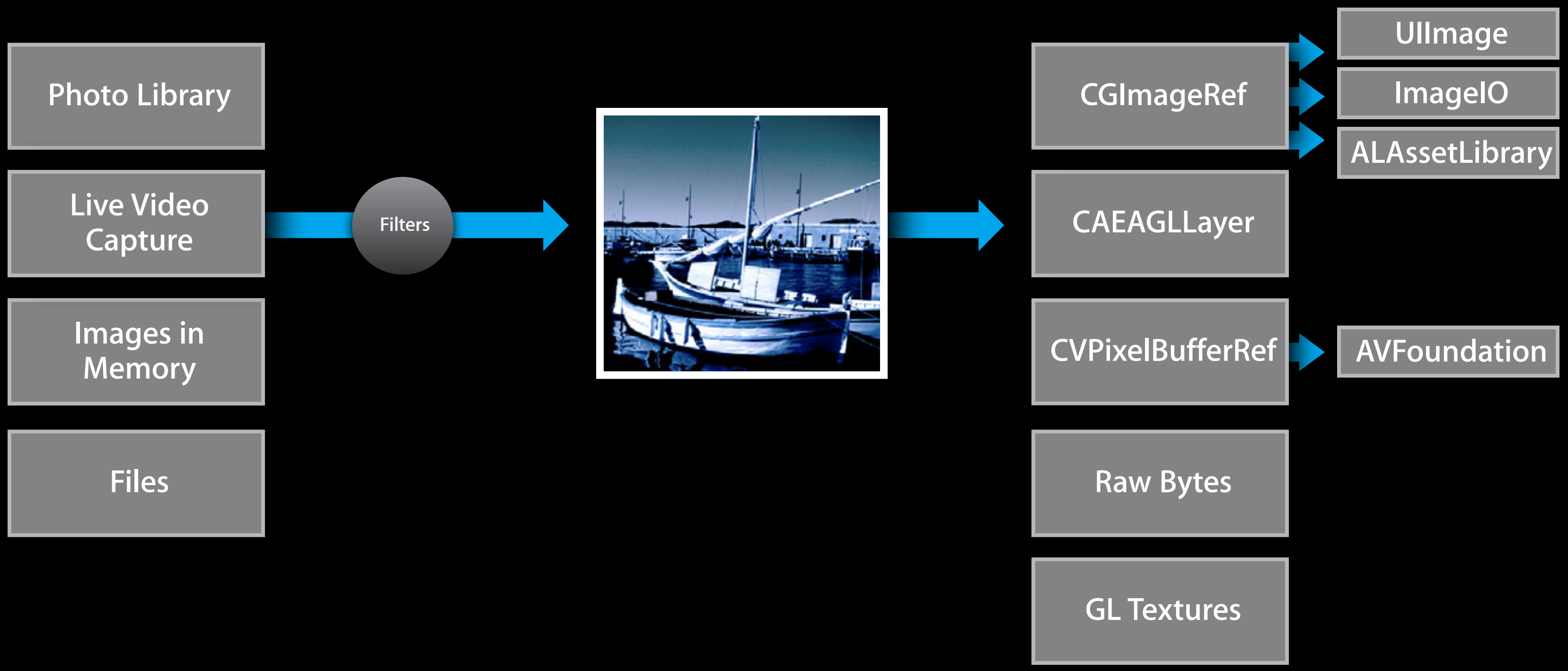


Photo Library

Live Video Capture

Images in Memory

Files

Filters

CGImageRef

CAEAGLLayer

CVPixelBufferRef

Raw Bytes

GL Textures

UIImage

ImageIO

ALAssetLibrary

AVFoundation

Creating a Real-Time Camera Effects App

Attempt 1

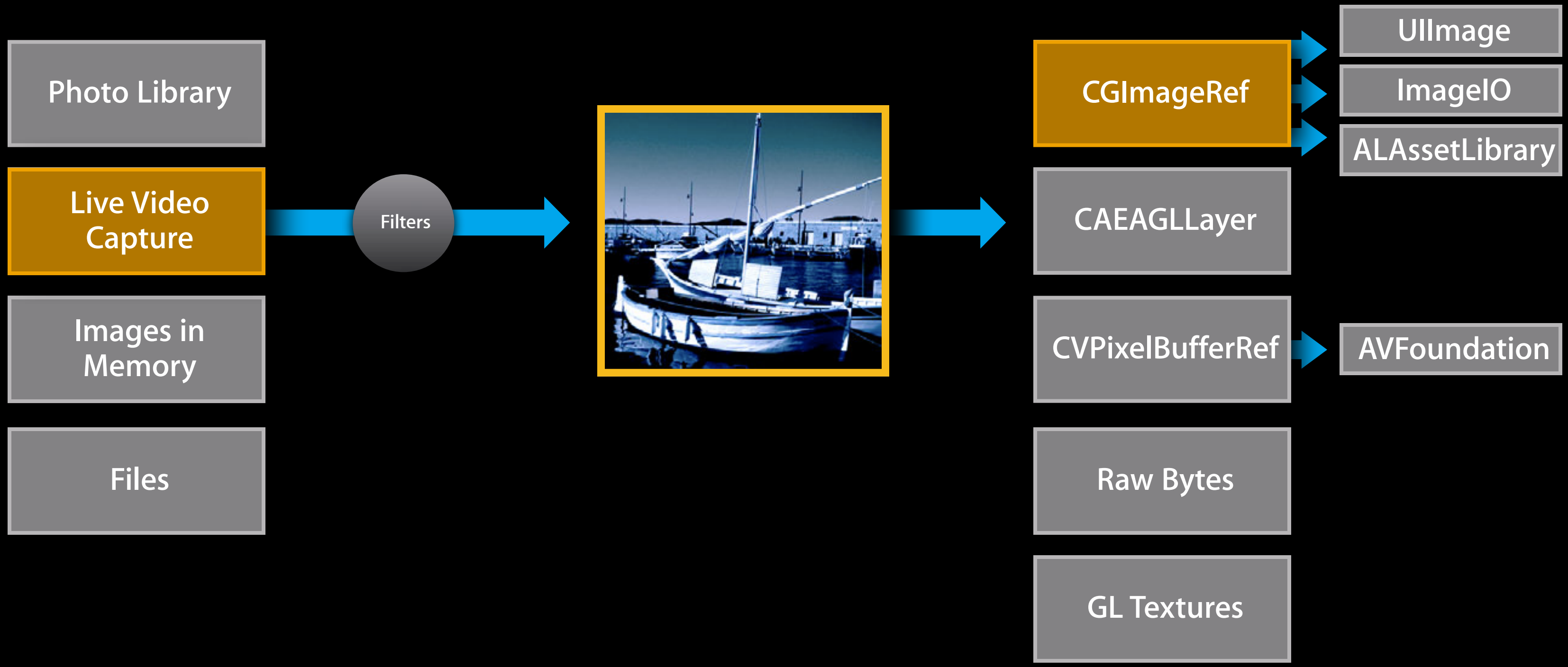


Photo Library

Live Video Capture

Images in Memory

Files

Filters

CGImageRef

CAEAGLLayer

CVPixelBufferRef

Raw Bytes

GL Textures

UIImage

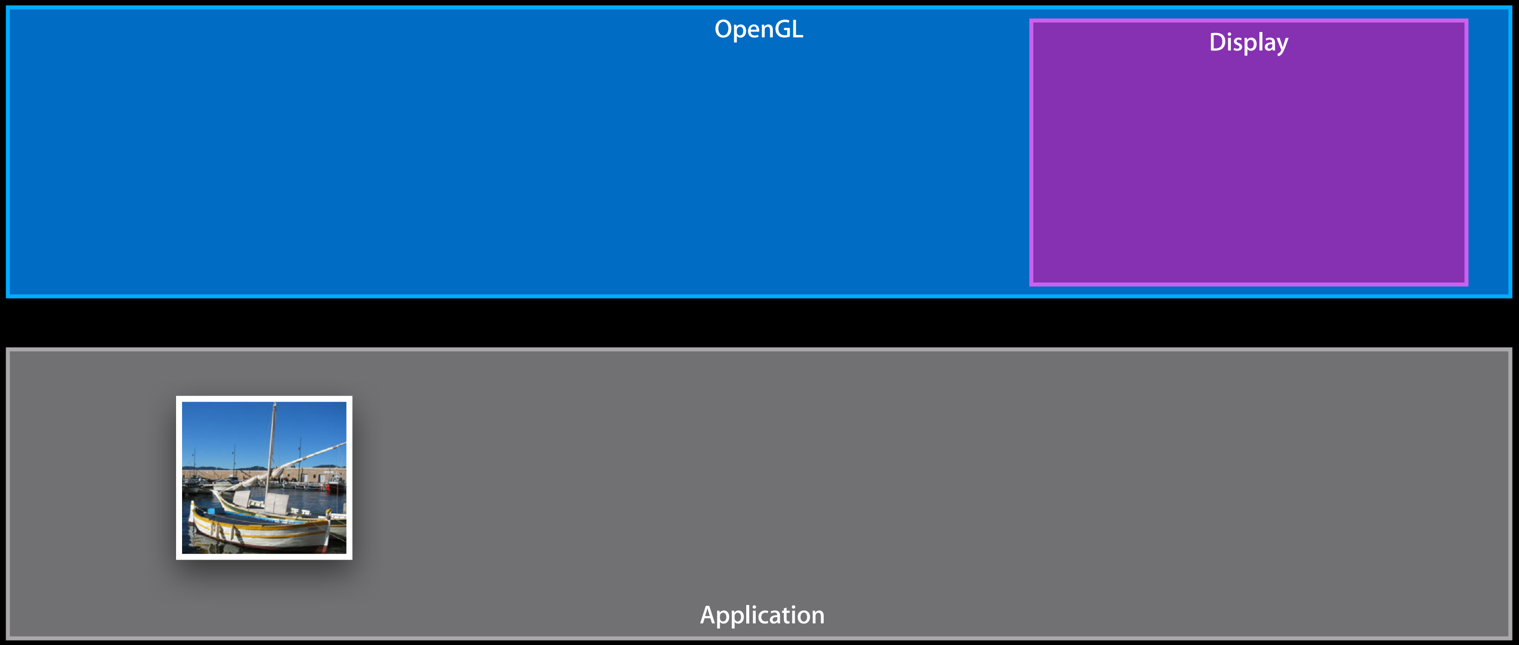
ImageIO

ALAssetLibrary

AVFoundation

Demo

Performance Is Still Non-Ideal

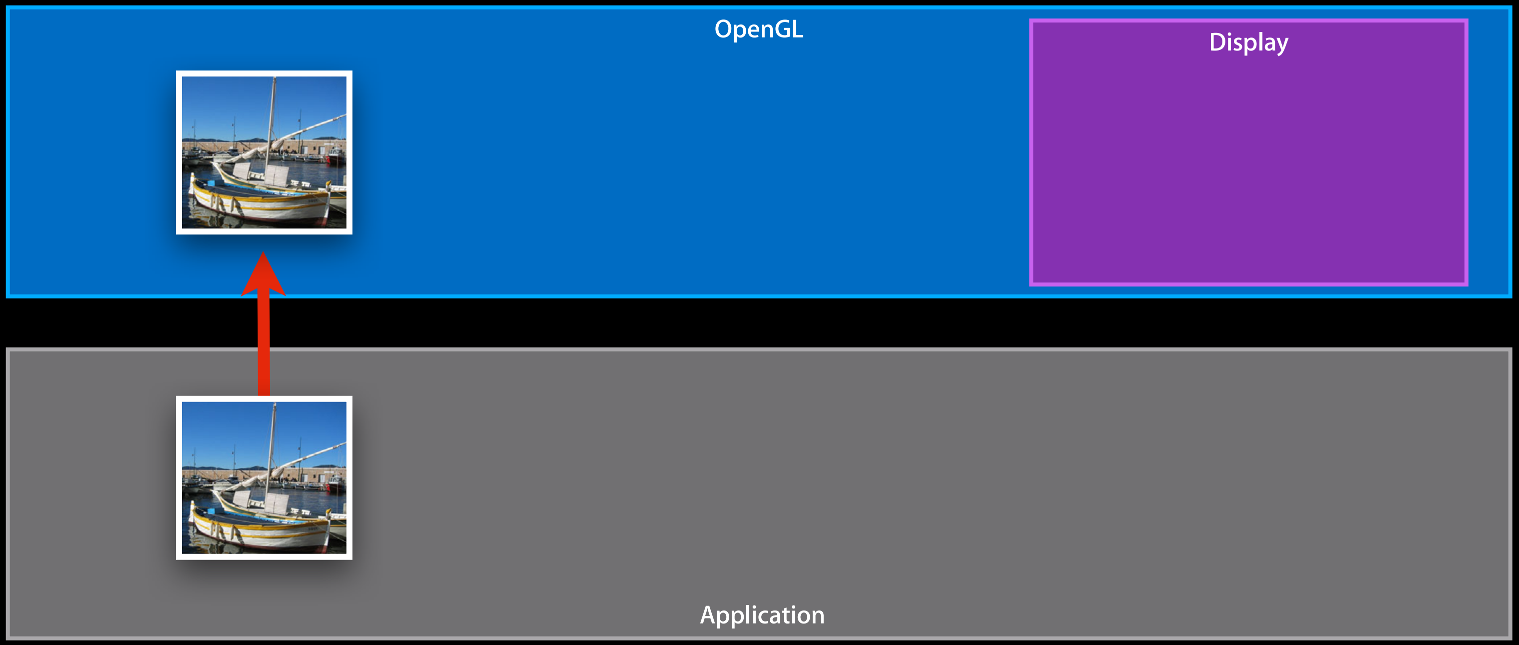


OpenGL

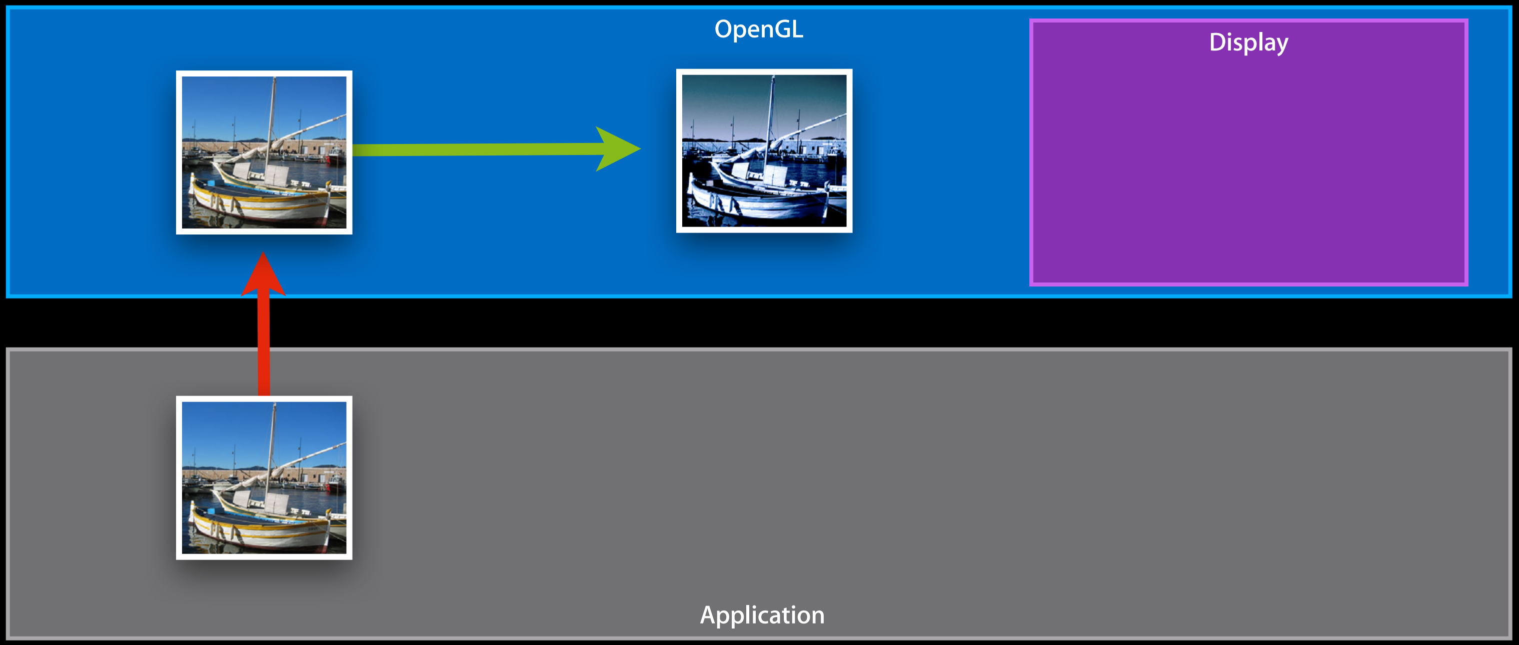
Display

Application

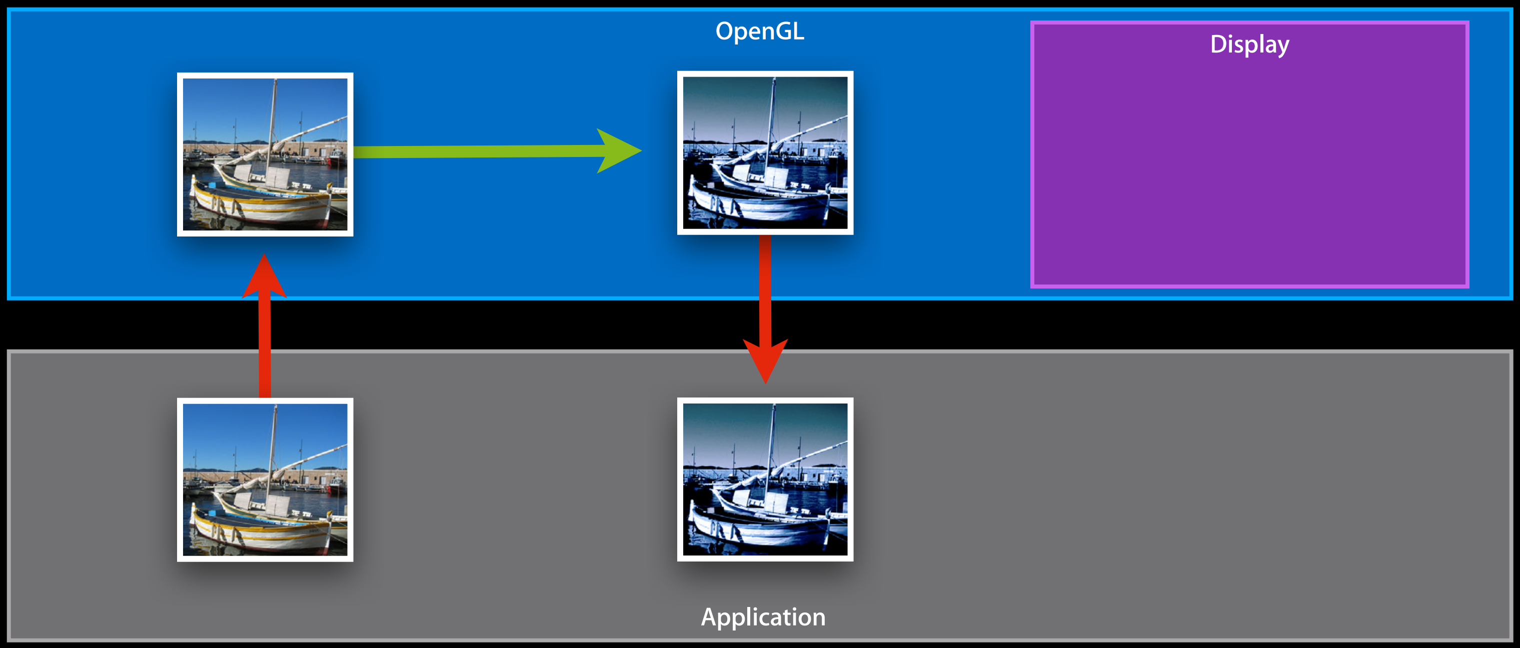
Performance Is Still Non-Ideal



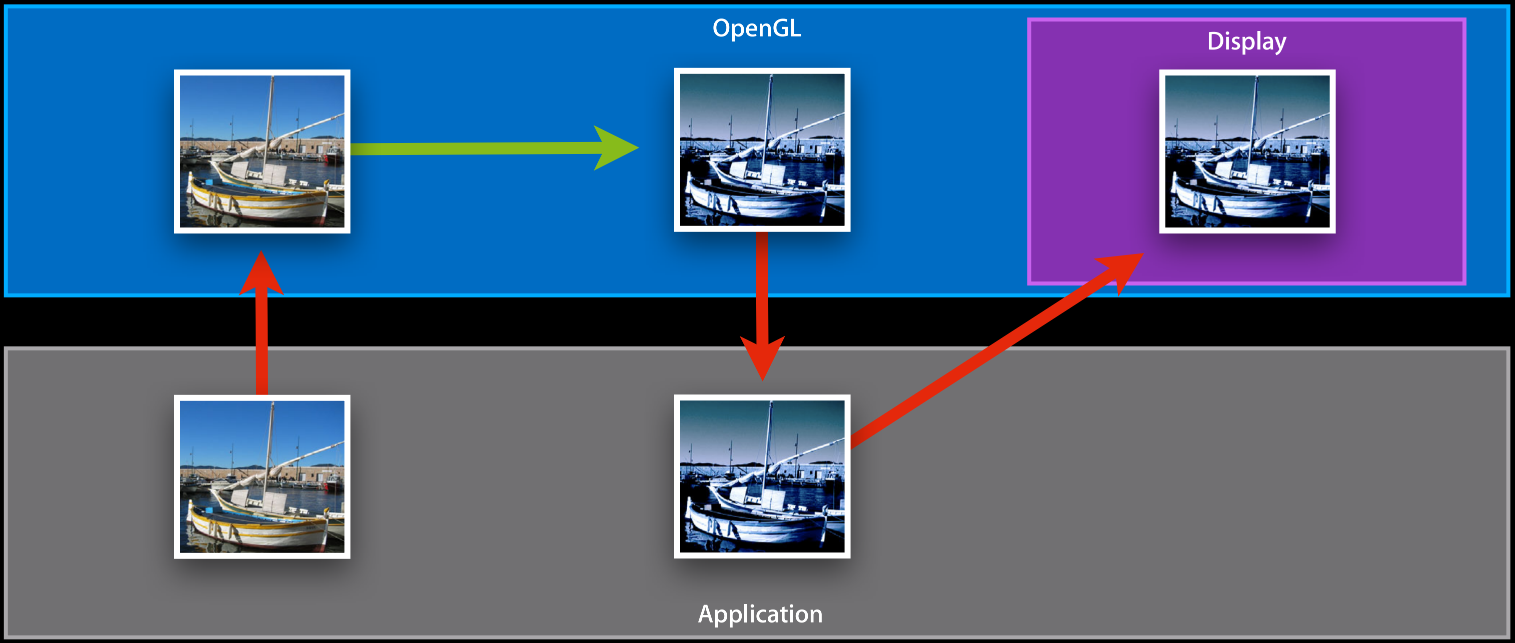
Performance Is Still Non-Ideal



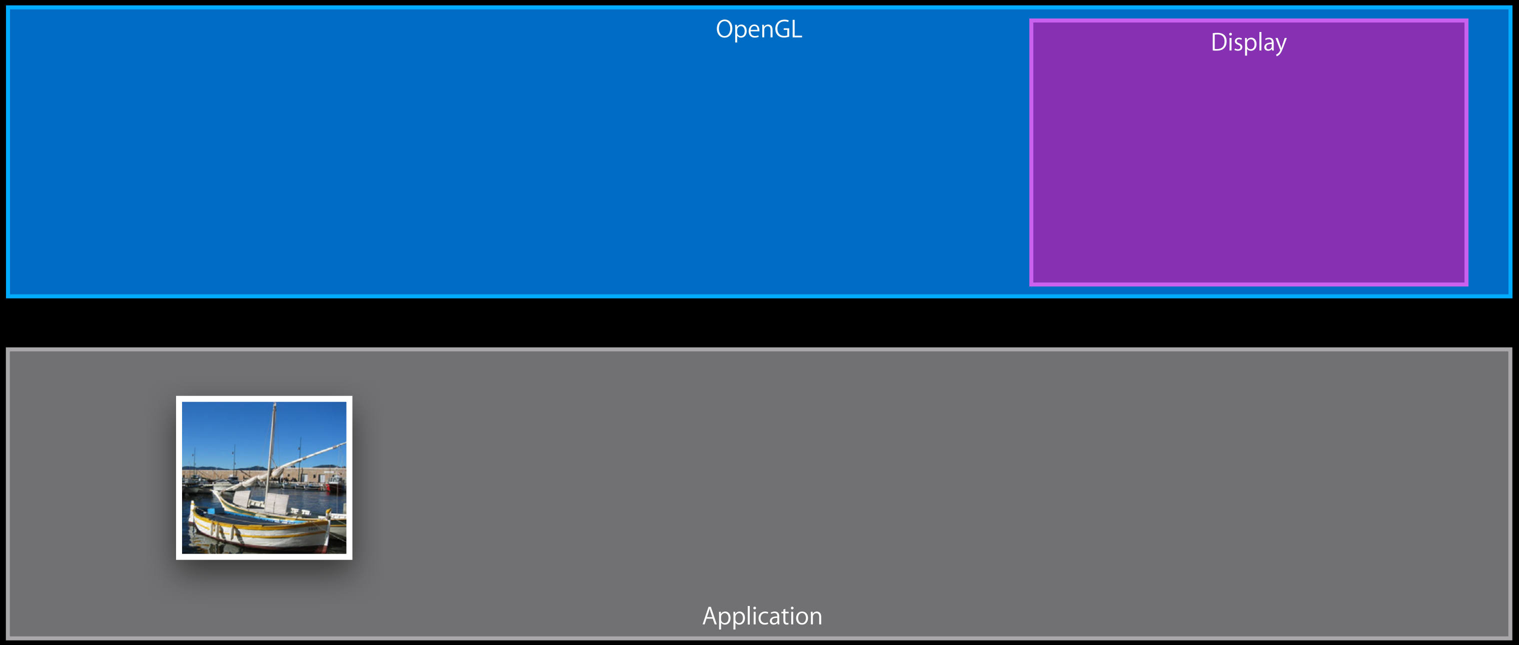
Performance Is Still Non-Ideal



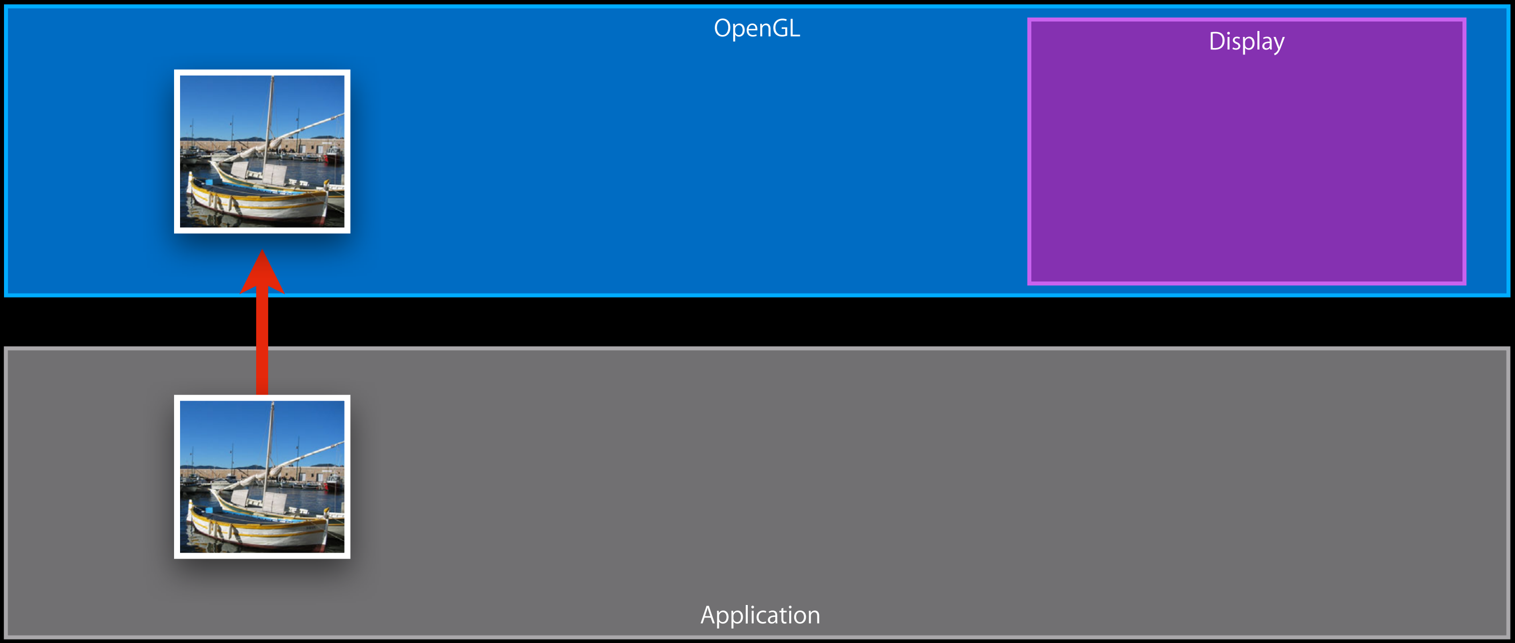
Performance Is Still Non-Ideal



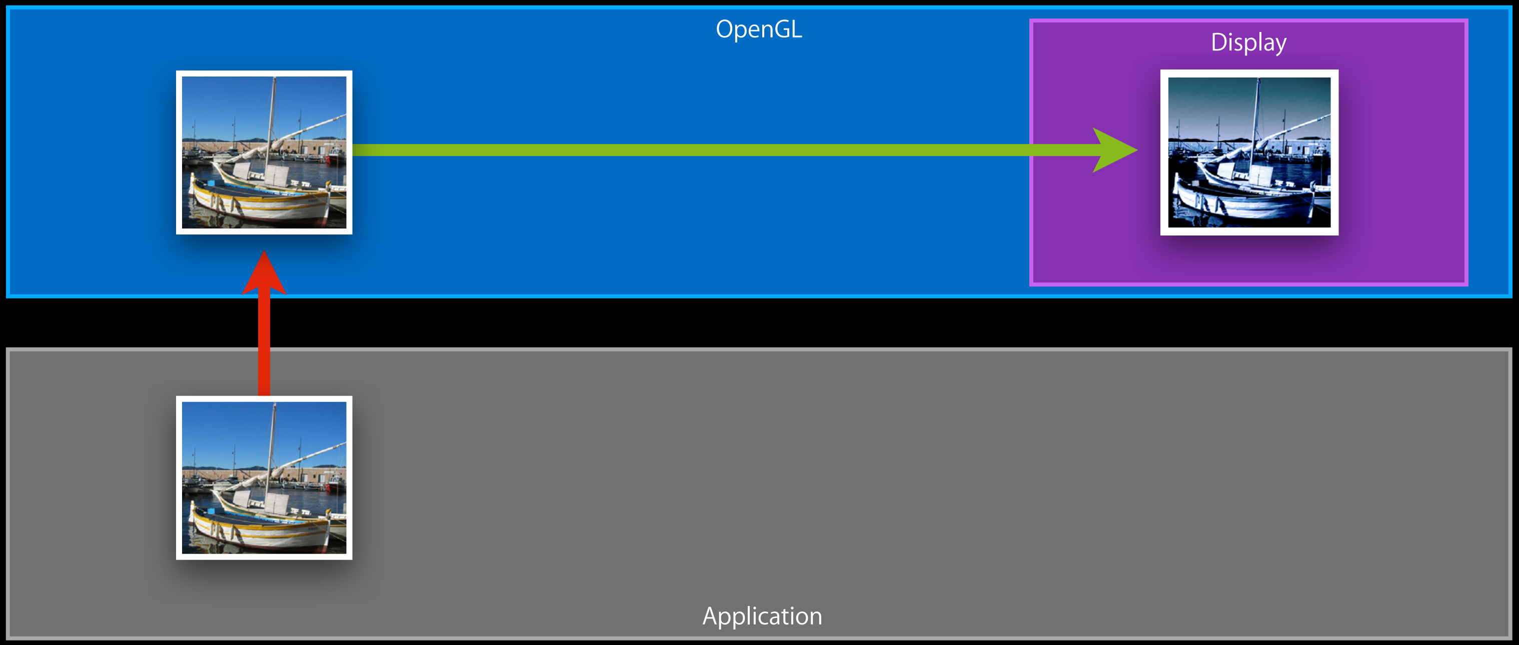
Avoid Unnecessary Texture Transfers



Avoid Unnecessary Texture Transfers



Avoid Unnecessary Texture Transfers



Creating a Real-Time Camera Effects App

Attempt 2

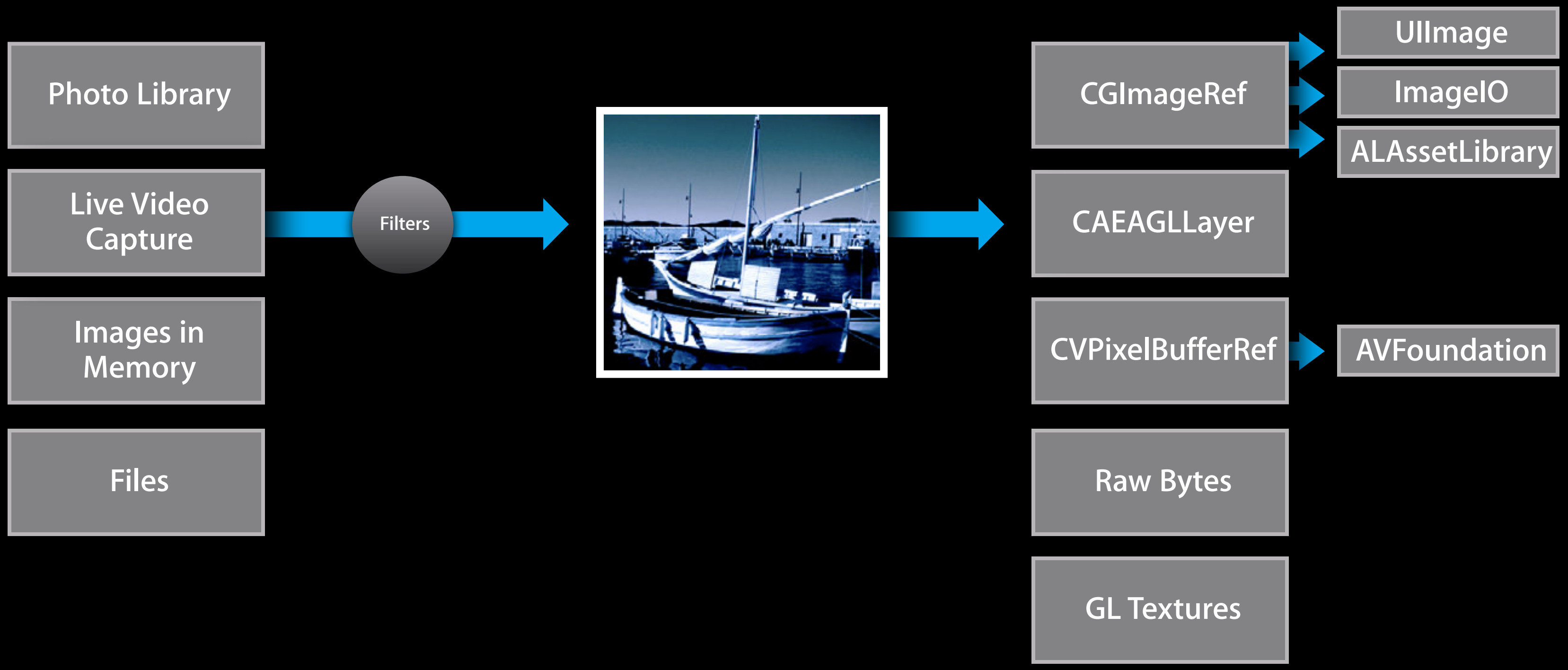


Photo Library

Live Video Capture

Images in Memory

Files

Filters

CGImageRef

UIImage

ImageIO

ALAssetLibrary

CAEAGLLayer

CVPixelBufferRef

AVFoundation

Raw Bytes

GL Textures

Creating a Real-Time Camera Effects App

Attempt 2

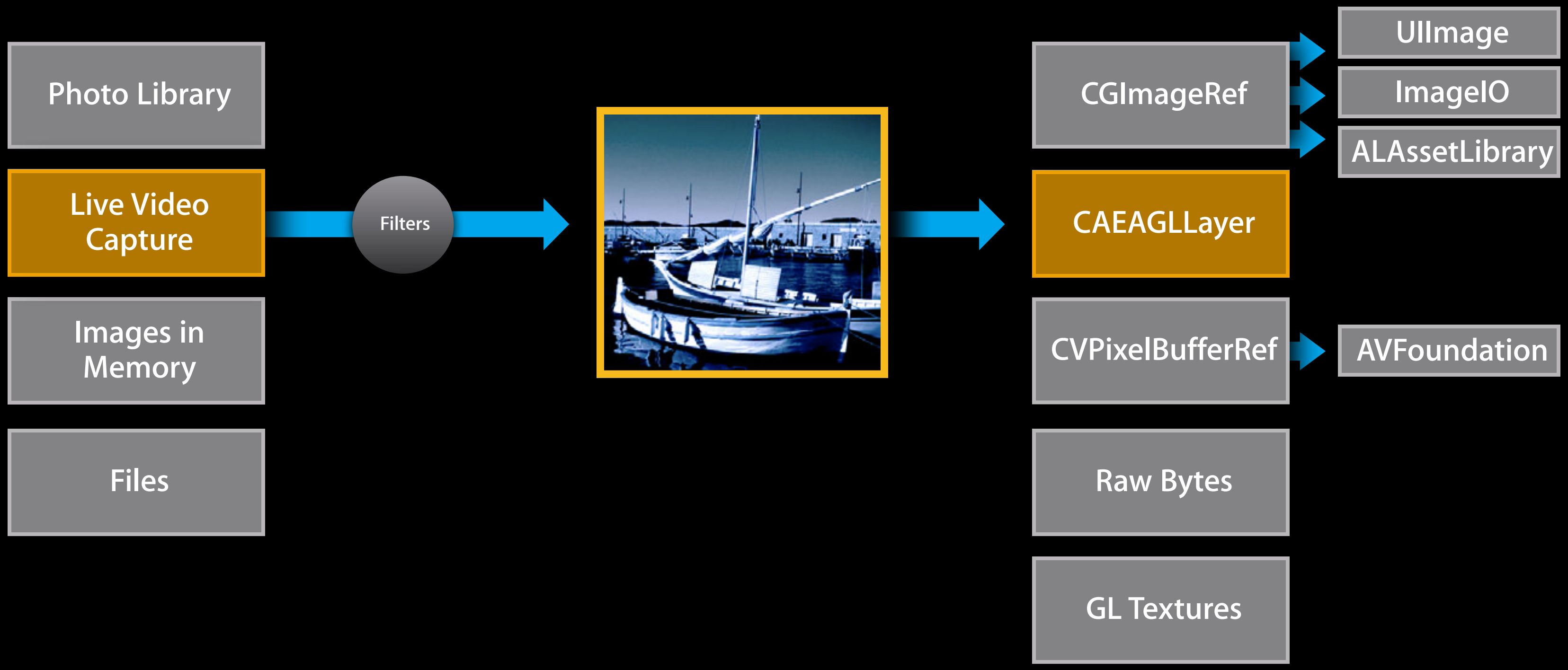


Photo Library

Live Video Capture

Images in Memory

Files

Filters

CGImageRef

UIImage

ImageIO

ALAssetLibrary

CAEAGLLayer

CVPixelBufferRef

AVFoundation

Raw Bytes

GL Textures

Demo

Can We Go Faster?

- Color management
- Leverage YUV image support
- Reduce the render size

Does Your App Need Color Management?

- By default, Core Image applies all filters in light-linear color space
  - This provides the most accurate and consistent results
- Conversions from and to sRGB add to the filter complexity
  - `rgb = mix(rgb*0.0774, pow(rgb*0.9479 + 0.05213, 2.4), step(0.04045,rgb))`
  - `rgb = mix(rgb*12.92, pow(rgb,0.4167) * 1.055 - 0.055, step(0.00313,rgb))`
- Consider disabling if:
  - You need the absolute highest performance
  - Users won’t notice the quality differences after exaggerated manipulations

YUV Image Support

- Camera pixel buffers are natively YUV
- Most image processing algorithms expect RGBA data
- Conversion between the two isn't free, and requires memory
- Core Image on iOS 6 supports reading from YUV CVPixelBuffers and applying the appropriate color transform

Reduce the Render Size

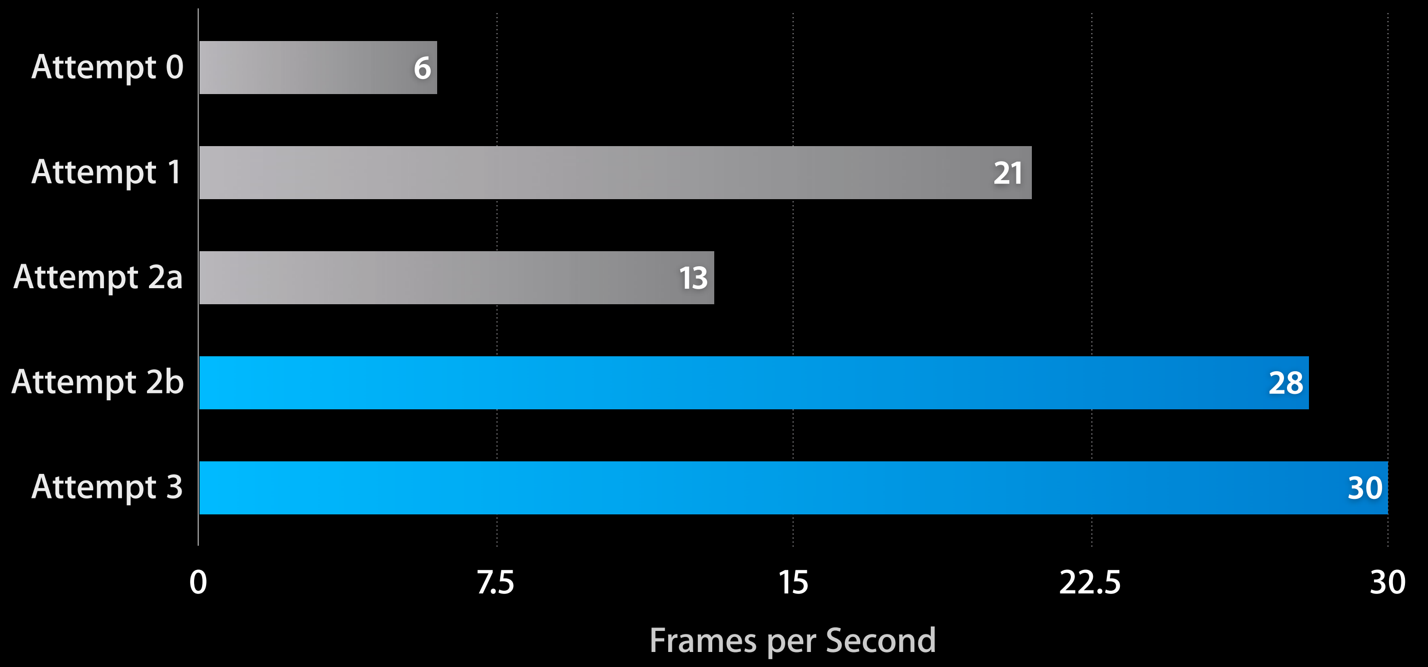
- Render time is proportional to the output pixel count
- HiDPI has four times the pixel count
- Optimal frame rates may require rendering at reduced sizes
  - Have Core Image render into a smaller view, texture, or framebuffer
  - Allow Core Animation to upscale to display size

HiDPI API Best Practices

- The preferred API is `[CIContext drawImage:inRect:fromRect:]`
  - fromRect coordinates are always pixel-based
  - inRect coordinates depend on the context type
    - Points if the CIContext is CGContext-based
    - Pixels if the CIContext is GL-based
- `[CIContext drawImage:atPoint:fromRect:]` is deprecated in OS X and iOS

Demo

Performance Summary



Core Image Techniques

Leveraging OpenGL ES

Core Image and OpenGL ES

- You can create a CIContext using an existing EAGLContext
  - We create a new EAGLContext with the same share group
- GL resources are shared between your EAGLContext and Core Images
- We can leverage this sharing for some more advanced techniques

Creating CIImages from Textures

- New API to create a CIImage from a OpenGL texture ID
  - `[CIImage imageWithTexture:size:flipped:colorSpace:]`
- The CIImage is only usable if the texture ID exists in the sharegroup
- Keeps data on the GPU, avoiding costly uploads/downloads
- Make sure texture data is valid when rendering the CIImage

Rendering to Textures

- Previously, we could only render to renderbuffers
- Framebuffers with renderbuffer attachments render to screen
- We now have an easy way to render to a texture
  - Bind a texture to the framebuffer
  - `[CIContext drawImage:inRect:fromRect:]`
- Only rendering to 8 bit RGBA textures is currently supported

Asynchronous Drawing

- On iOS 5, `[CIContext drawImage:inRect:fromRect:]` is synchronous
- On iOS 6, `[CIContext drawImage:inRect:fromRect:]` is asynchronous
- Apps linked on iOS 5 will continue to be synchronous
- Be aware of OpenGL ES flush/bind best practices;
  when rendering to a texture:
    - We'll issue a `glFlush()` after our render
    - You need to rebind on your context

Example App

- Modifies the standard XCode OpenGL ES template app
- Efficiently creates GL textures from CVPixelBuffers with CVOpenGLESTextureCache
- Creates CIImages from these textures
- Renders to a texture via `[CIContext drawImage:inRect:fromRect:]`
- Uses OpenGL ES to render that texture onto cubes with custom shaders

Demo

Game Technologies Manager

Jacques Gasselin de Richebourg
Employee

Core Image Techniques for Games

Common use cases

1. Apply an effect to the full screen
2. Apply an effect to individual textures

Core Image Techniques for Games

Common use cases

1. Apply an effect to the full screen

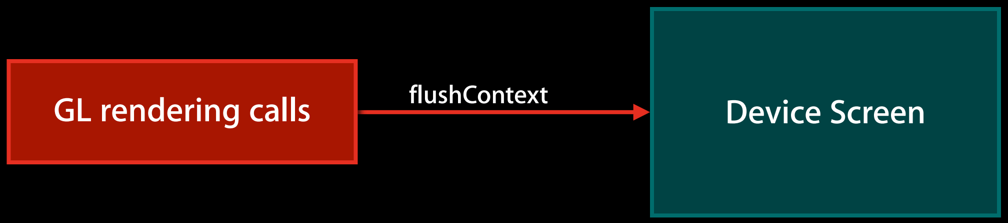
2. Apply an effect to individual textures

Demo

Apply an effect to the full screen

1 Apply an Effect to the Full Screen

1 Apply an Effect to the Full Screen

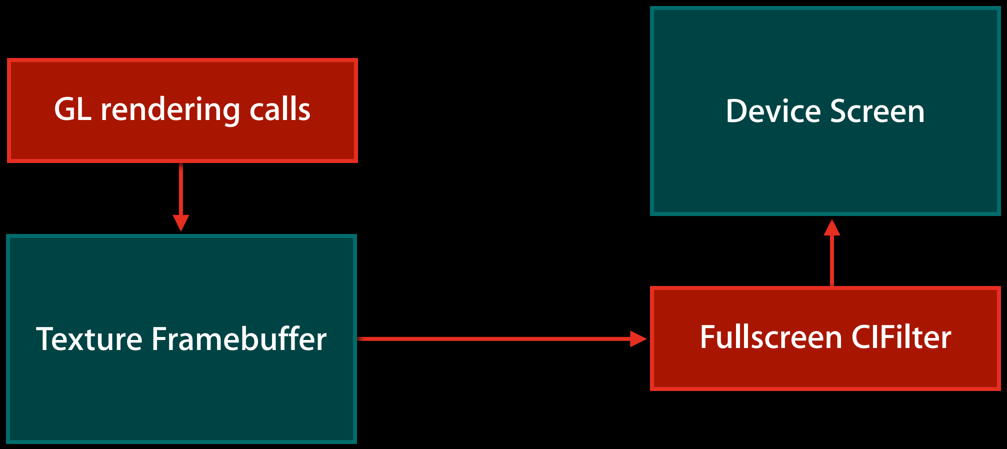


GL rendering calls

flushContext

Device Screen

1 Apply an Effect to the Full Screen



GL rendering calls

Texture Framebuffer

Fullscreen CIFilter

Device Screen

How Did We Do This

The workflow

1. Create an OpenGL texture FBO
    - Render content into this
2. Create a CIImage, using the FBO
3. Create the CIFilter
    - Set the `kCIInputImageKey` value to the CIImage
4. Create a CIContext to filter to render to the device

How Did We Do This

1. Create the texture FBO (OS X)

```
//Create an empty 32bit texture of dimensions (width, height)
GLuint texture = ...;
glBindTexture(GL_TEXTURE_RECTANGLE_ARB, texture);
glTexImage2D(GL_TEXTURE_RECTANGLE_ARB, 0, GL_RGBA, width, height, 0,
    GL_BGRA, GL_UNSIGNED_INT_8_8_8_8_REV, NULL);

//Create a texture framebuffer bound to the color buffer
GLuint framebuffer = ...;
glBindFramebuffer(GL_FRAMEBUFFER, framebuffer);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0,
     GL_TEXTURE_RECTANGLE_ARB, texture, 0);
```

How Did We Do This

1. Create the texture FBO (iOS)

```
//Create an empty 32bit texture of dimensions (width, height)
GLuint texture = ...;
glBindTexture(GL_TEXTURE_2D, texture);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0,
    GL_BGRA, GL_UNSIGNED_INT_8_8_8_8_REV, NULL);

//Create a texture framebuffer bound to the color buffer
GLuint framebuffer = ...;
glBindFramebuffer(GL_FRAMEBUFFER, framebuffer);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0,
     GL_TEXTURE_2D, texture, 0);
```

How Did We Do This

2. Create the CIImage

```
GLuint texture = ...

CIImage *input = [CIImage imageWithTexture:texture
                                      size:CGSizeMake(width, height)
                                   flipped:NO
                                colorSpace:nil];
```

How Did We Do This

3. Create the CIFilter

```
GLuint texture = ...

CIImage *input = ...

CIFilter *filter = [CIFilter filterWithName:@"CIBloom"];
[filter setDefaults];
[filter setValue:input forKey:kCIInputImageKey];
```

How Did We Do This

4. Create the CIContext (OS X)

```
GLuint texture = ...

CIImage *input = ...

CIFilter *filter = ...

CGLContextObj cglContext = [nsglctx CGLContextObj];
CGLPixelFormatObj cglPixelFormat = CGLGetPixelFormat(cglContext);

NSDictionary *opts = @{ kCIContextWorkingColorSpace : [NSNull null] };

CIContext *ciContext = [CIContext contextWithCGLContext:cglContext
                                            pixelFormat:cglPixelFormat
                                             colorSpace:nil
                                                options:opts];
```

How Did We Do This

4. Create the CIContext (iOS)

```
GLuint texture = ...

CIImage *input = ...

CIFilter *filter = ...

EAGLContext *glContext = ...


NSDictionary *opts = @{ kCIContextWorkingColorSpace : [NSNull null] };

CIContext *ciContext = [CIContext contextWithEAGLContext:glContext
                                                 options:opts];
```

Core Image Techniques for Games

Common use cases

1. Apply an effect to the full screen
2. Apply an effect to individual textures

Core Image Techniques for Games

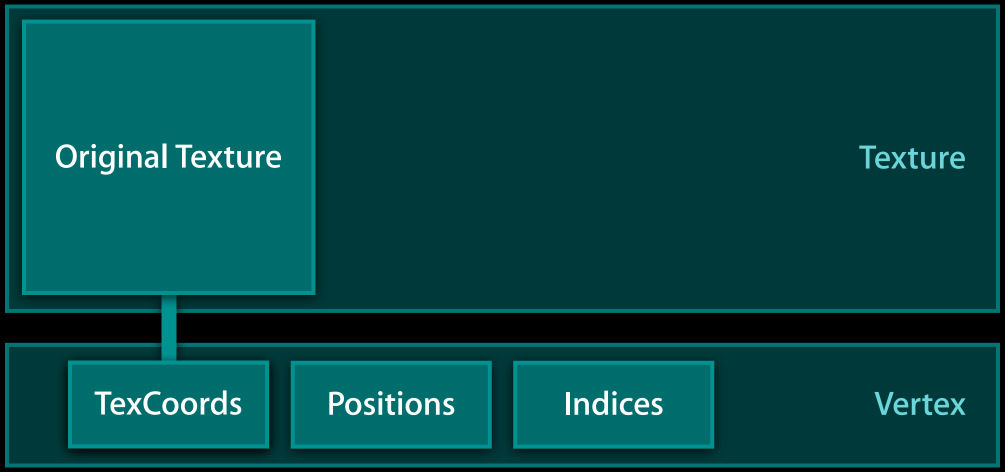
Common use cases

1. Apply an effect to the full screen
2. Apply an effect to individual textures

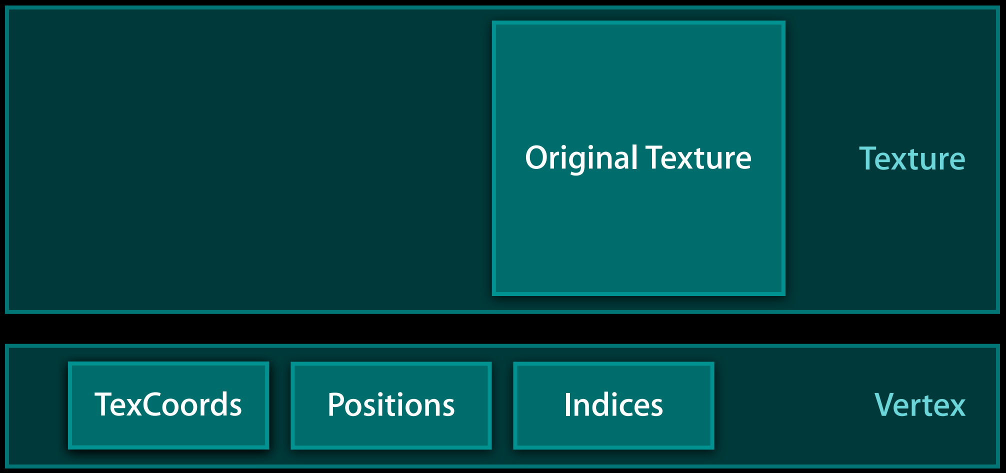
Demo

Apply an effect to individual textures

2 Apply an Effect to Individual Textures



2 Apply an Effect to Individual Textures



Original Texture

Texture

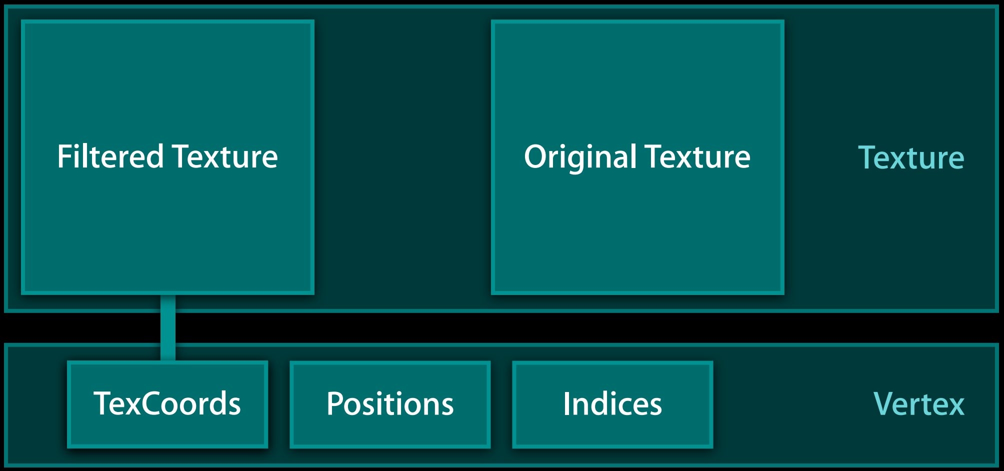
TexCoords

Positions

Indices

Vertex

2 Apply an Effect to Individual Textures



Filtered Texture

Original Texture

Texture

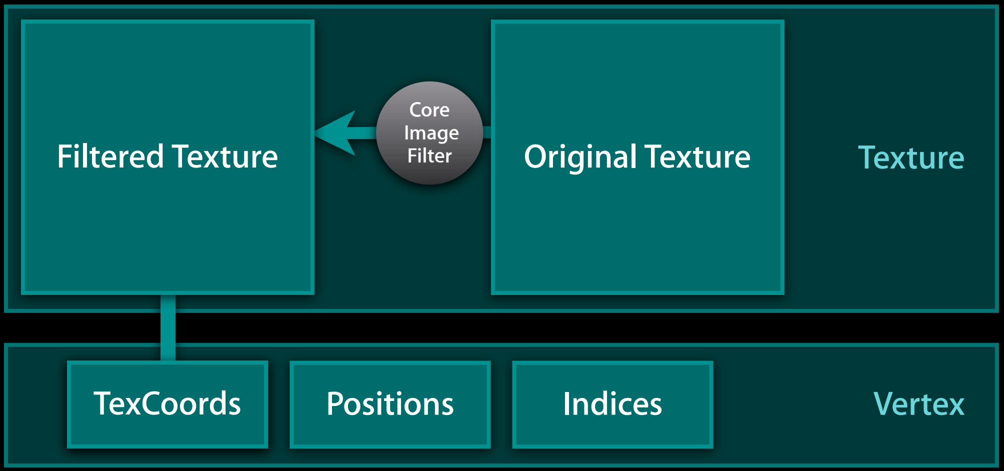
TexCoords

Positions

Indices

Vertex

2 Apply an Effect to Individual Textures



Filtered Texture

Core Image Filter

Original Texture

Texture

TexCoords

Positions

Indices

Vertex

How Did We Do This?

The workflow

1. Create a CIImage, using the texture
2. Create the CIFilter
    - Set the kCIInputImageKey value to the CIImage
3. Create an OpenGL texture FBO
    - This is the new texture
4. Create a CIContext to filter to
    - Targets the new texture

Filter to Texture

Differences from the fullscreen case

```
//Create a texture to render into
GLuint outputTexture = ...

//Make outputTexture the target of a texture framebuffer
GLuint framebuffer = ...

//bind the texture framebuffer as the output instead of the screen
glBindFramebuffer(GL_FRAMEBUFFER, framebuffer);

//This now draws the filter output to the texture instead of to the screen
CGRect rect = CGRectMake(0, 0, width, height);
[ciContext drawImage:[filter valueForKey:kCIOutputImageKey]
              inRect:rect
            fromRect:rect];
```

Core Image and Games

Great features for games out-of-the-box

- 93 combinable filters
    - Billions of unique combinations
- Render to and from OpenGL textures
- Selectable quality vs. performance settings

More Information

Allan Schaffer
Graphics and Imaging Evangelist
aschaffer@apple.com

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

Related Sessions

Getting Started with Core Image	Pacific Heights Wednesday 10:15AM
Advances in OpenGL and OpenGL ES	Pacific Heights Wednesday 2:00PM

Labs

Core Image Lab	Graphics, Media, & Games Lab A Wednesday 2:00PM

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.