Live Photo Editing and RAW Processing with Core Image

Session 505

David Hayward Pixel Perfectionist

What You Will Learn Today

A very brief introduction to Core Image

Adjusting RAW images on iOS

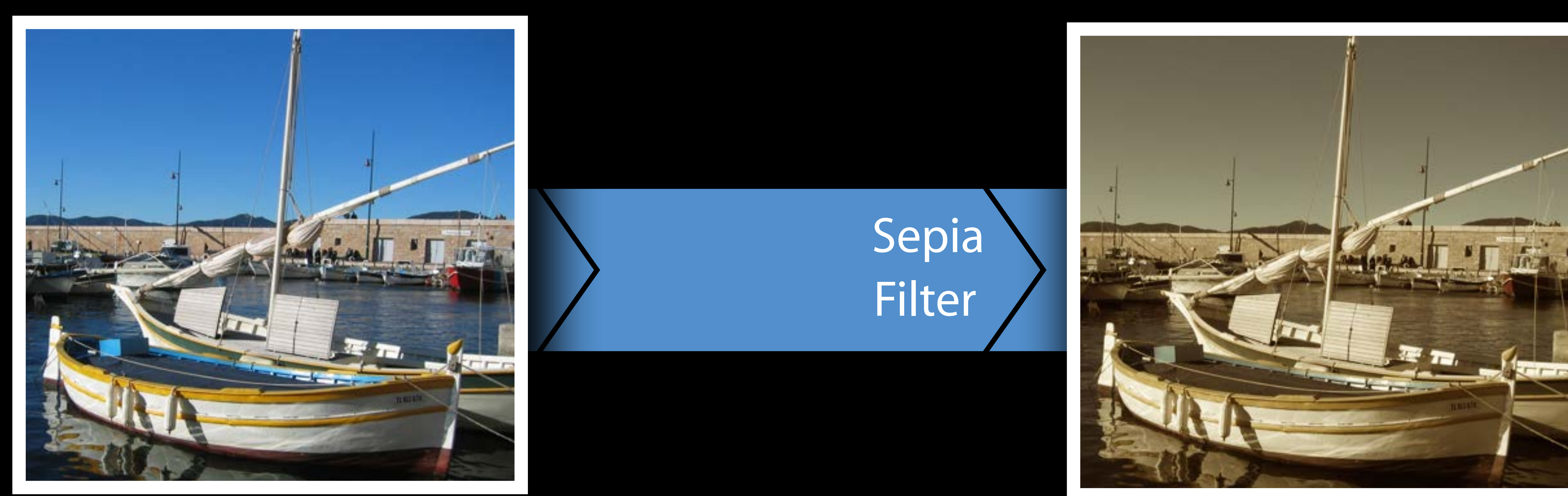
Editing Live Photos

Extending Core Image using CIImageProcessor

A Very Brief Introduction to Core Image

A Very Brief Introduction to Core Image

A simple, high-performance API to apply filters to images

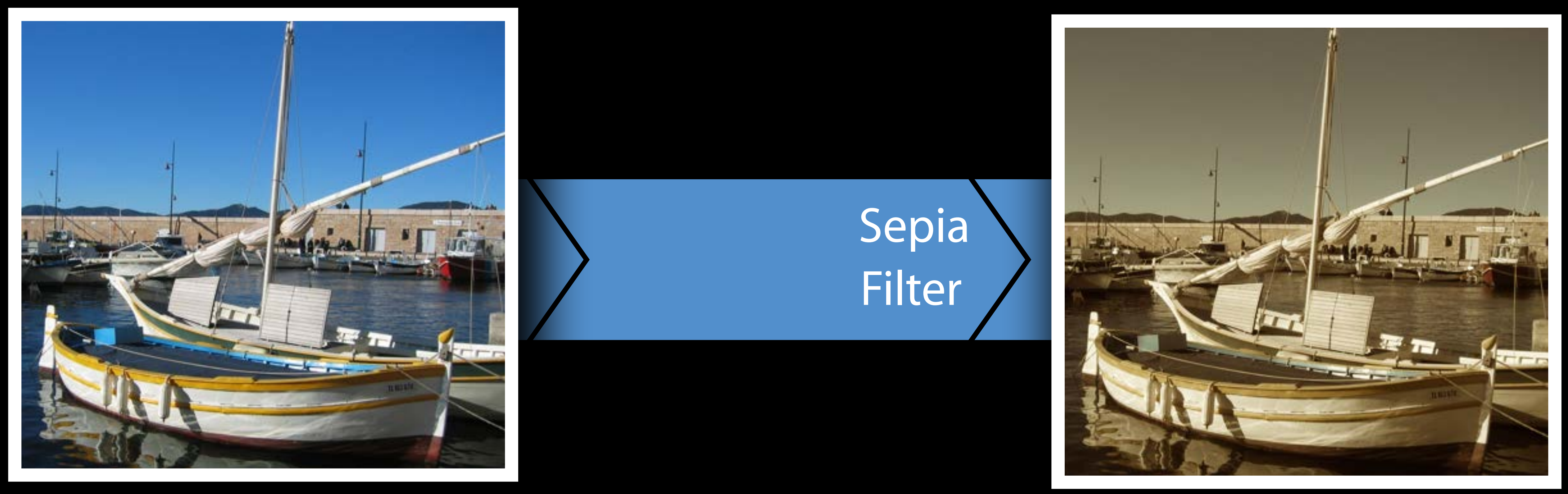


Original CIImage

Output CIImage

A Very Brief Introduction to Core Image

A simple, high-performance API to apply filters to images



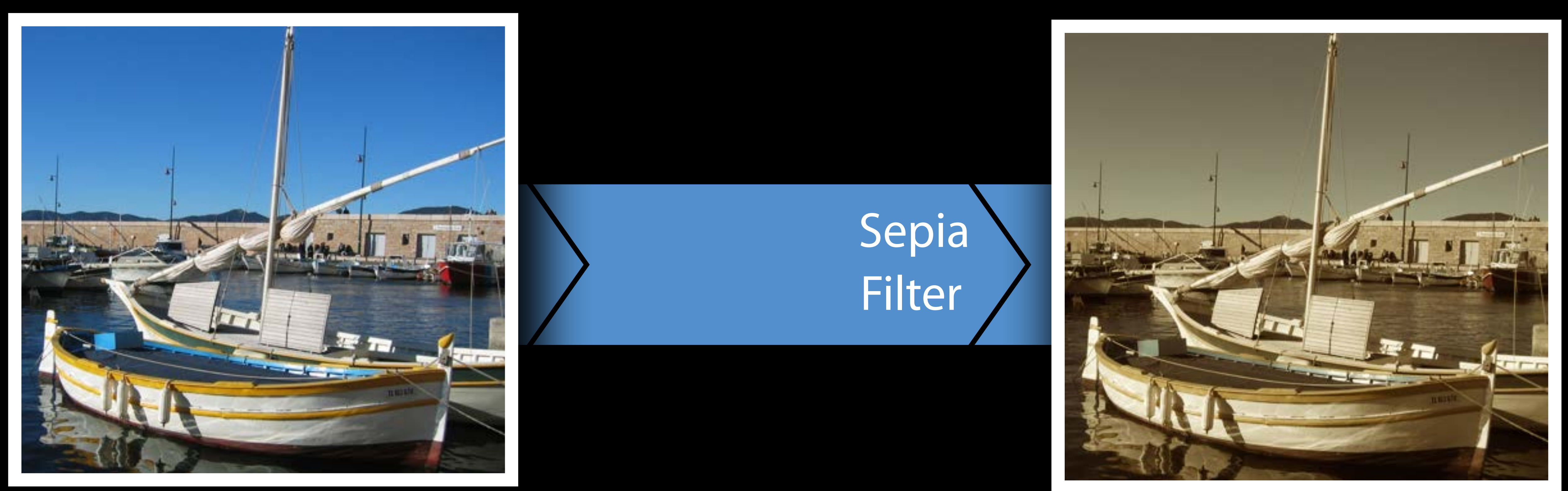
Original CIImage

Output CIImage

```
image = image.applyingFilter(
  "CISepiaTone",
   withInputParameters:
    ["inputIntensity" : 1.0])
```

A Very Brief Introduction to Core Image

A simple, high-performance API to apply filters to images

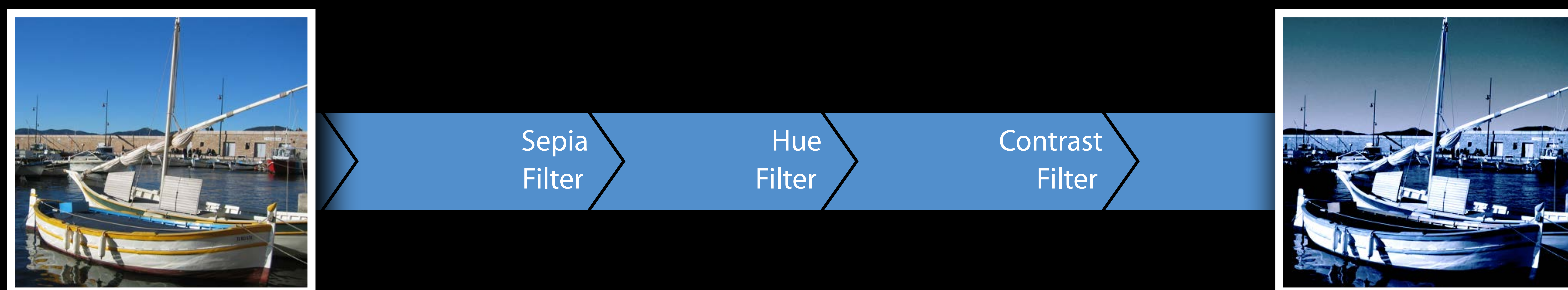


Original CIImage

Output CIImage

A Very Brief Introduction to Core Image

A simple, high-performance API to apply filters to images

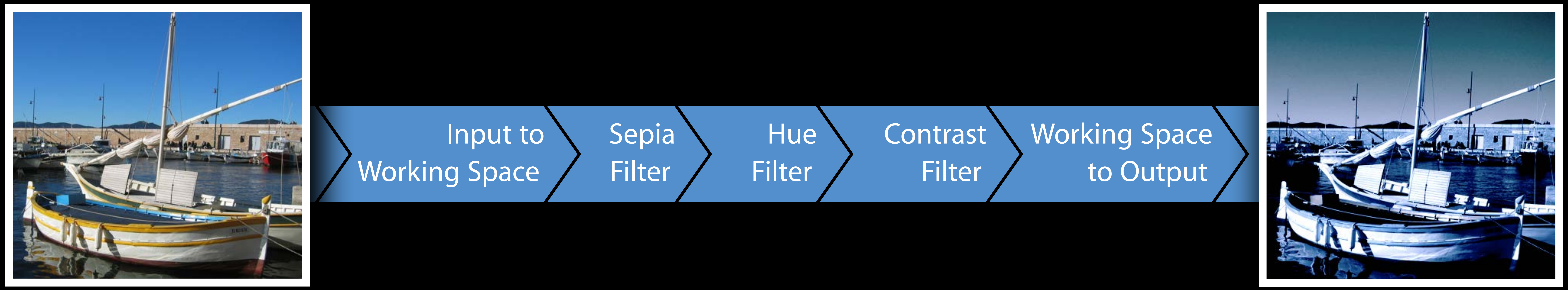


Original CIImage

Output CIImage

A Very Brief Introduction to Core Image

Automatic color management



Original CIImage

Output CIImage

A Very Brief Introduction to Core Image

Automatic color management

Original CIImage

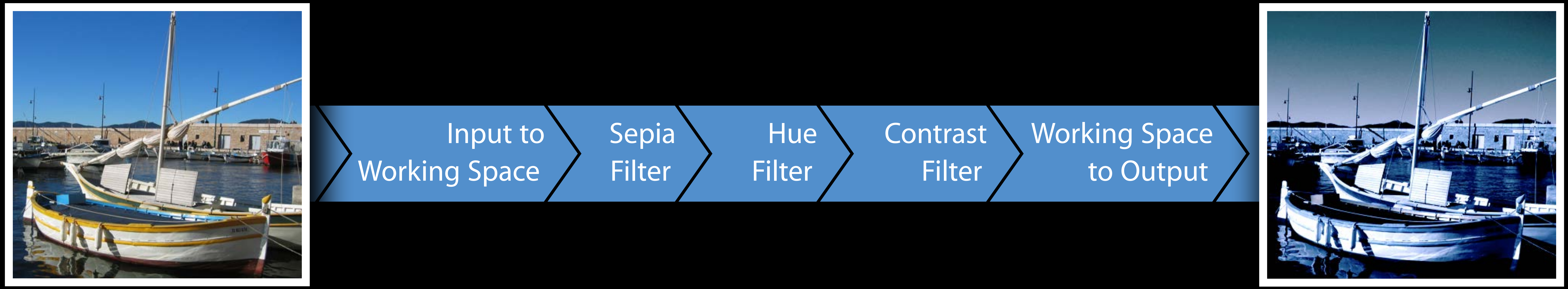
Wide color images and displays are common.

Most open-source image processing libraries do not support color management.

Output CIImage

A Very Brief Introduction to Core Image

Automatic color management

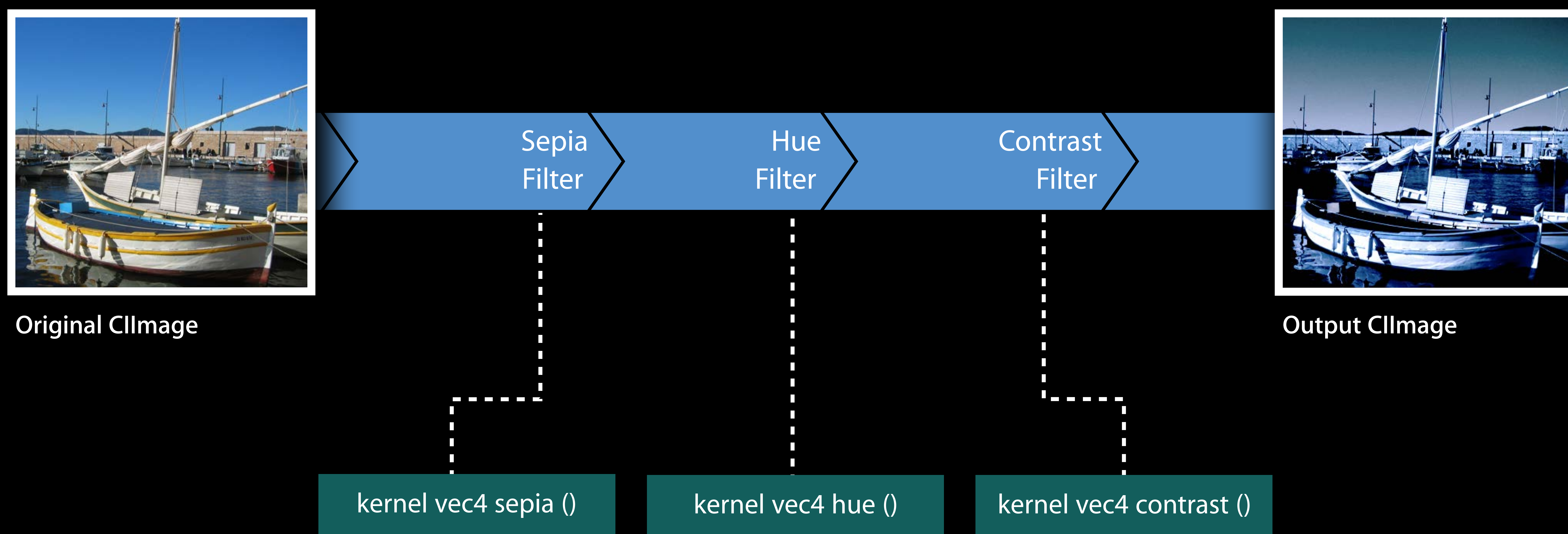


Original CIImage

Output CIImage

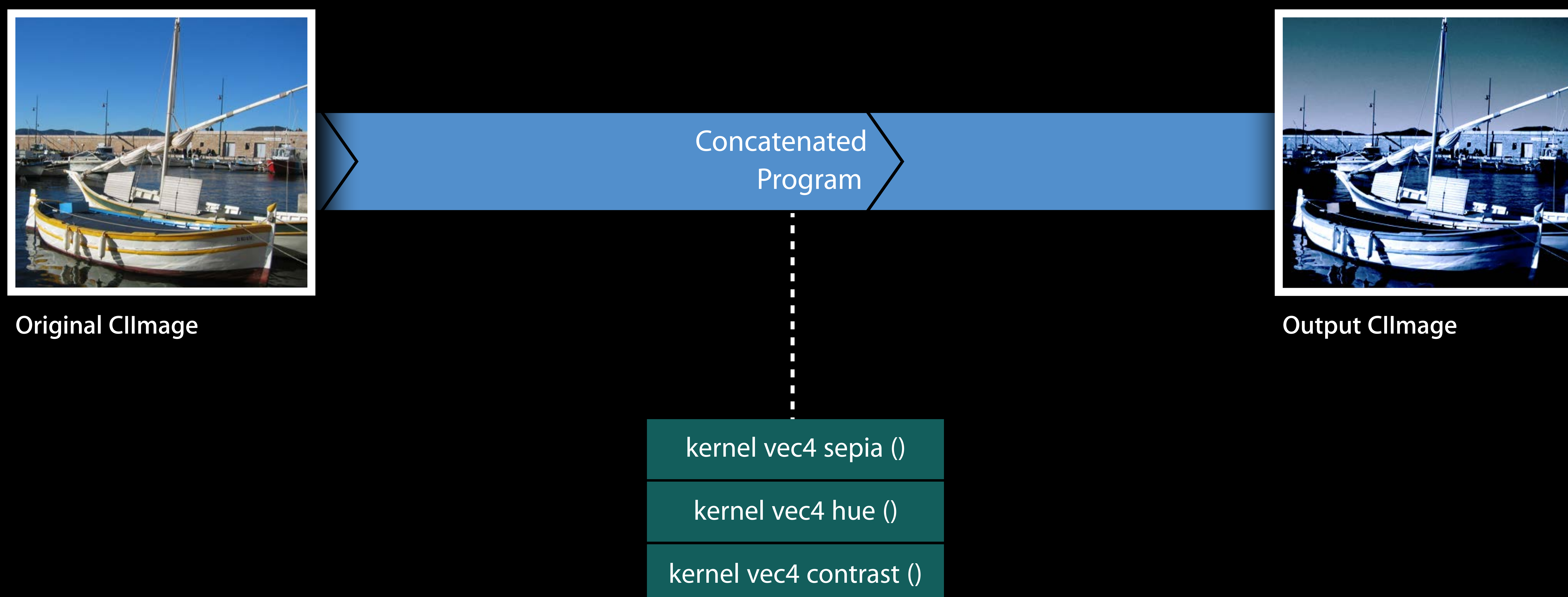
A Very Brief Introduction to Core Image

Each CIFilter has one or more CIKernel functions



A Very Brief Introduction to Core Image

Each CIFilter has one or more CIKernel functions



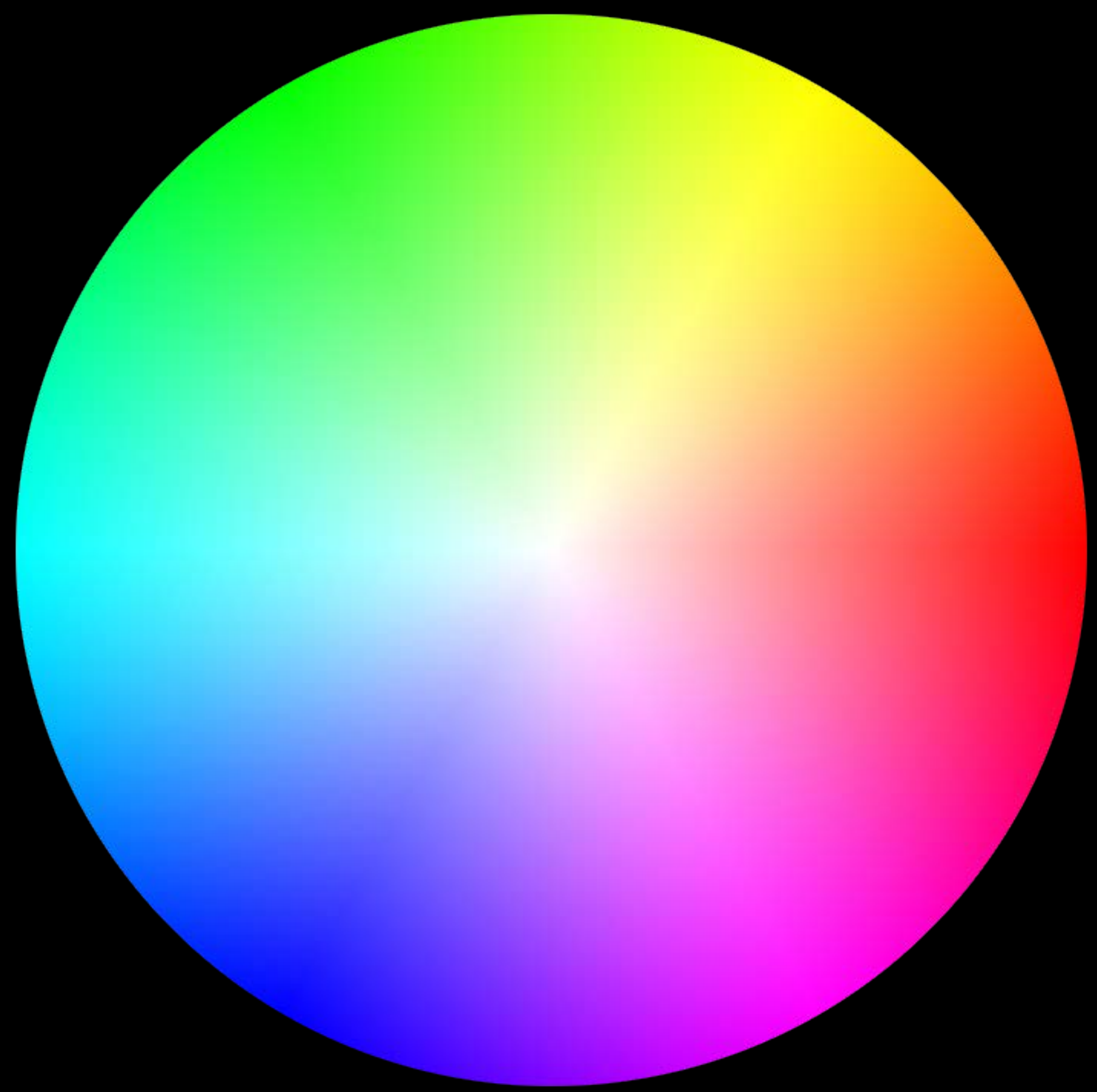
180 Built-In Filters

AccordionFoldTransition
AdditionCompositing
AffineClamp
AffineTile
AffineTransform
AreaAverage
AreaHistogram
AreaMaximum
AreaMaximumAlpha
AreaMinimum
AreaMinimumAlpha
AztecCodeGenerator
BarsSwipeTransition
BlendWithAlphaMask
BlendWithMask
Bloom
BoxBlur
BumpDistortion
BumpDistortionLinear
CheckerboardGenerator
CircleSplashDistortion
CircularScreen
CircularWrap
Clamp
CMYKHalftone
Code128BarcodeGenerator
ColorBlendMode
ColorBurnBlendMode
ColorClamp
ColorControls
ColorCrossPolynomial
ColorCube
ColorCubeWithColorSpace
ColorDodgeBlendMode
ColorInvert
ColorMap
ColorMatrix
ColorMonochrome
ColorPolynomial
ColorPosterize
ColumnAverage
ComicEffect
ConstantColorGenerator
Convolution3X3
Convolution5X5
Convolution7X7
Convolution9Horizontal
Convolution9Vertical
CopyMachineTransition
Crop
Crystallize
DarkenBlendMode
DepthOfField
DifferenceBlendMode
DiscBlur
DisintegrateWithMaskTransition
DisplacementDistortion
DissolveTransition
DivideBlendMode
DotScreen
Droste
Edges
EdgeWork
EightfoldReflectedTile
ExclusionBlendMode
ExposureAdjust
FalseColor
FlashTransition
FourfoldReflectedTile
FourfoldRotatedTile
FourfoldTranslatedTile
GammaAdjust
GaussianBlur
GaussianGradient
GlassDistortion
GlassLozenge
GlideReflectedTile
Gloom
HardLightBlendMode
HatchedScreen
HeightFieldFromMask
HexagonalPixellate
HighlightShadowAdjust
HistogramDisplayFilter
HoleDistortion
HueAdjust
HueBlendMode
HueSaturationValueGradient
Kaleidoscope
LanczosScaleTransform
LenticularHaloGenerator
LightenBlendMode
LightTunnel
LinearBurnBlendMode
LinearDodgeBlendMode
LinearGradient
LinearToSRGBToneCurve
LineOverlay
LineScreen
LuminosityBlendMode
MaskedVariableBlur
MaskToAlpha
MaximumComponent
MaximumCompositing
MedianFilter
MinimumComponent
MinimumCompositing
ModTransition
MotionBlur
MultiplyBlendMode
MultiplyCompositing
NinePartStretched
NinePartTiled
NoiseReduction
OpTile
OverlayBlendMode
PageCurlTransition
PageCurlWithShadowTransition
ParallelogramTile
PDF417BarcodeGenerator
PerspectiveCorrection
PerspectiveTile
PerspectiveTransform
PerspectiveTransformWithExtent
PhotoEffectChrome
PhotoEffectFade
PhotoEffectInstant
PhotoEffectMono
PhotoEffectNoir
PhotoEffectProcess
PhotoEffectTonal
PhotoEffectTransfer
PinchDistortion
PinLightBlendMode
Pixellate
Pointillize
QRCodeGenerator
RadialGradient
RandomGenerator
RippleTransition
RowAverage
SaturationBlendMode
ScreenBlendMode
SepiaTone
ShadedMaterial
SharpenLuminance
SixfoldReflectedTile
SixfoldRotatedTile
SmoothLinearGradient
SoftLightBlendMode
SourceAtopCompositing
SourceInCompositing
SourceOutCompositing
SourceOverCompositing
SpotColor
SpotLight
SRGBToneCurveToLinear
StarShineGenerator
StraightenFilter
StretchCrop
StripesGenerator
SubtractBlendMode
SunbeamsGenerator
SwipeTransition
TemperatureAndTint
Thermal
ToneCurve
TorusLensDistortion
TriangleKaleidoscope
TriangleTile
TwelvefoldReflectedTile
TwirlDistortion
UnsharpMask
Vibrance
Vignette
VignetteEffect
VortexDistortion
WhitePointAdjust
XRay
ZoomBlur

New Built-In CIFilters

NEW

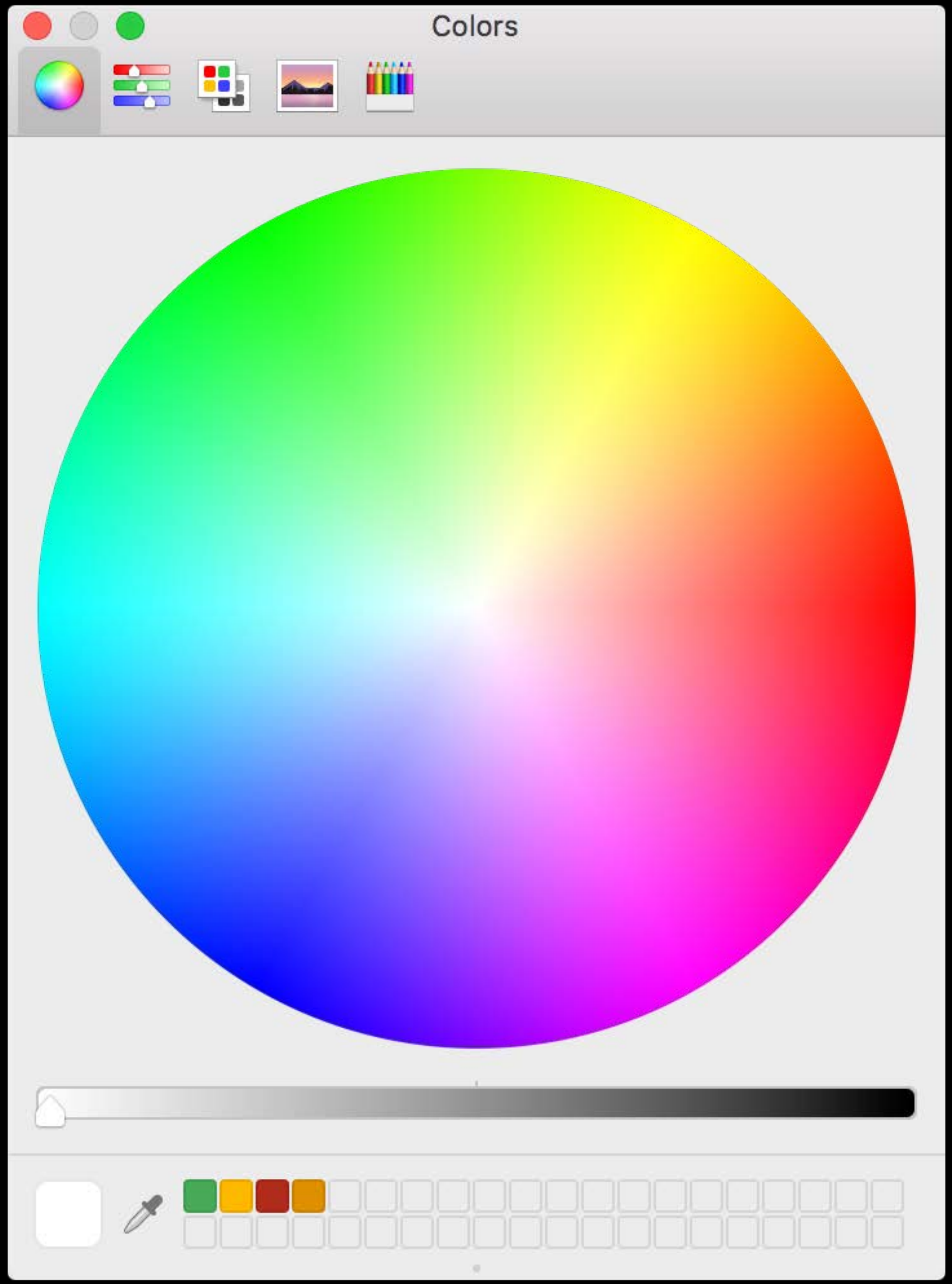
CIHueSaturationValueGradient



New Built-In CIFilters

NEW

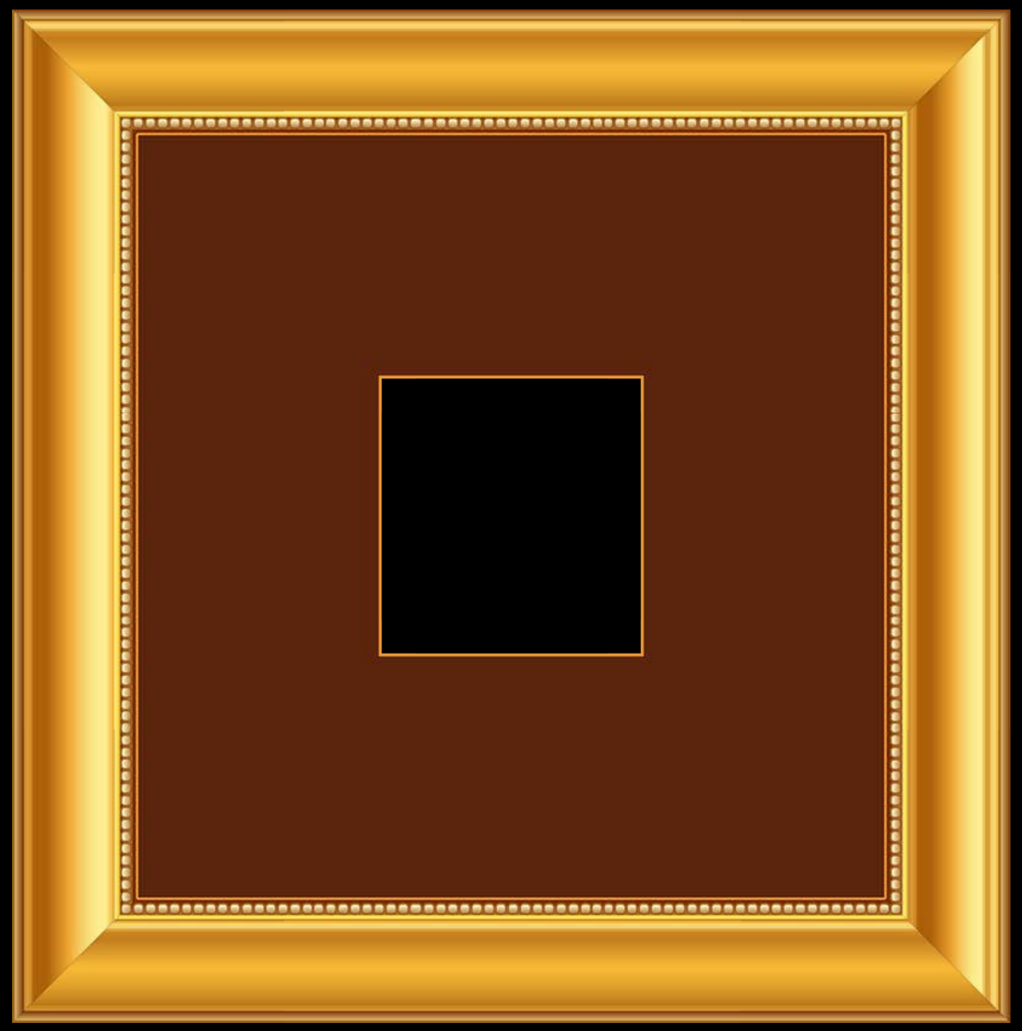
CIHueSaturationValueGradient



New Built-In CIFilters

CINinePartStretched and CINinePartTiled

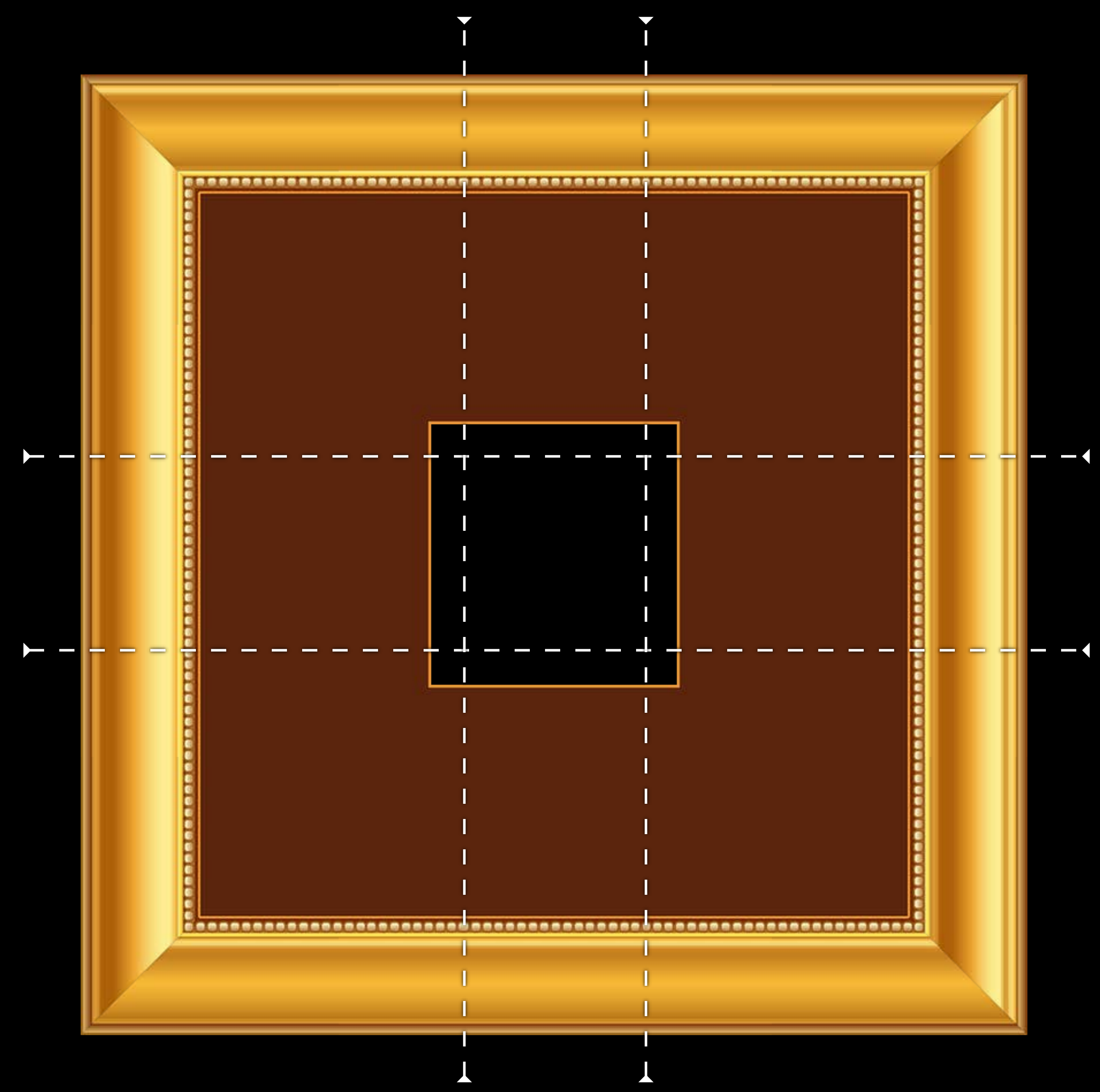
NEW



New Built-In CIFilters

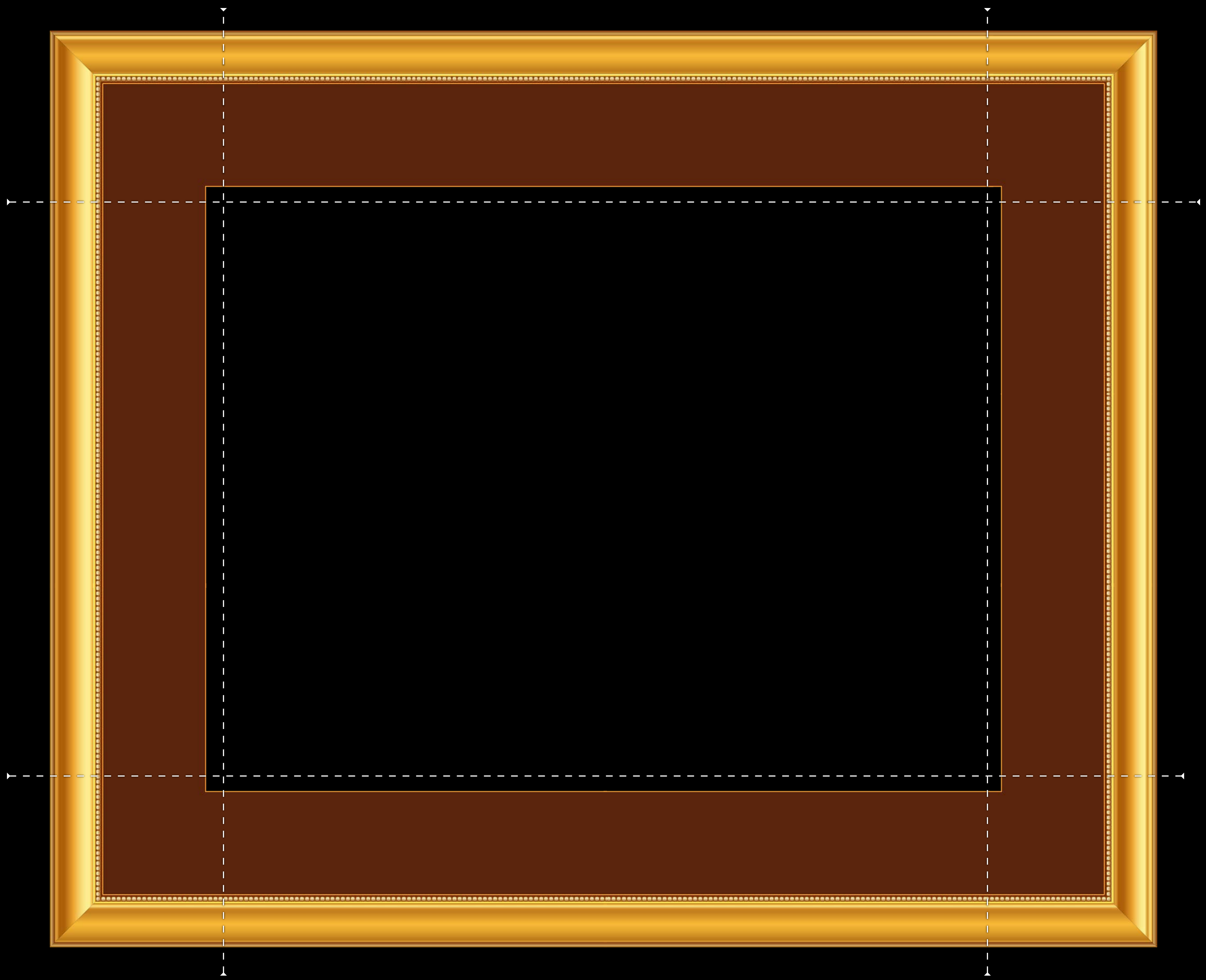
NEW

CINinePartStretched and CINinePartTiled



New Built-In CIFilters

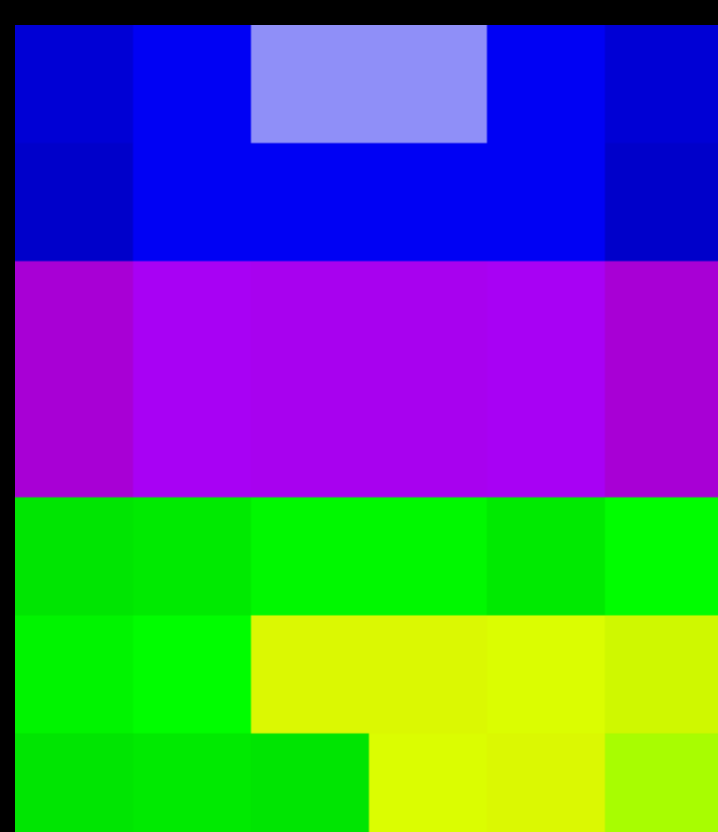
CINinePartStretched and CINinePartTiled



New Built-In CIFilters

CIEdgePreserveUpsampleFilter

NEW

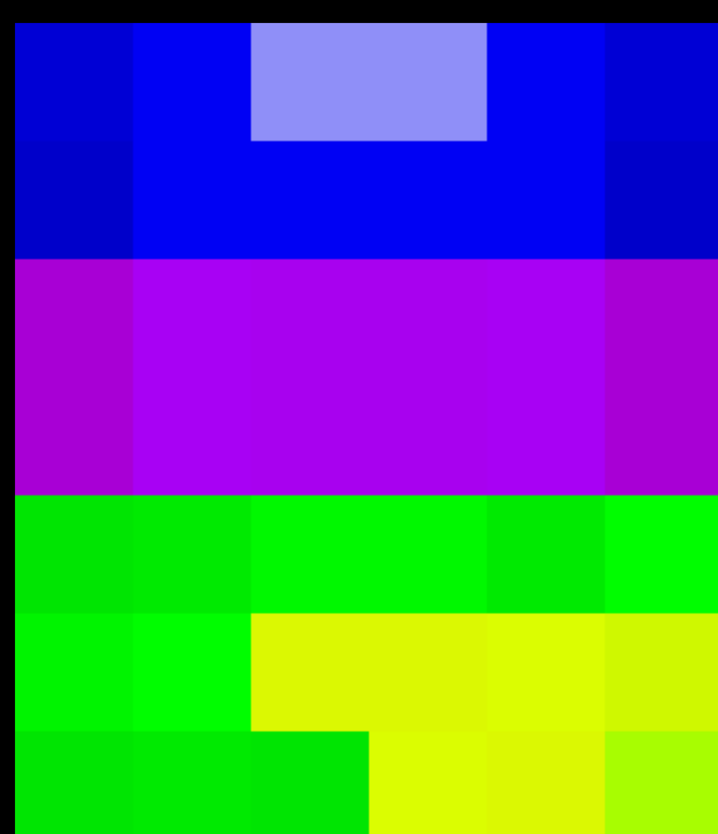


6x7 Pixel Input

New Built-In CIFilters

CIEdgePreserveUpsampleFilter

NEW



+

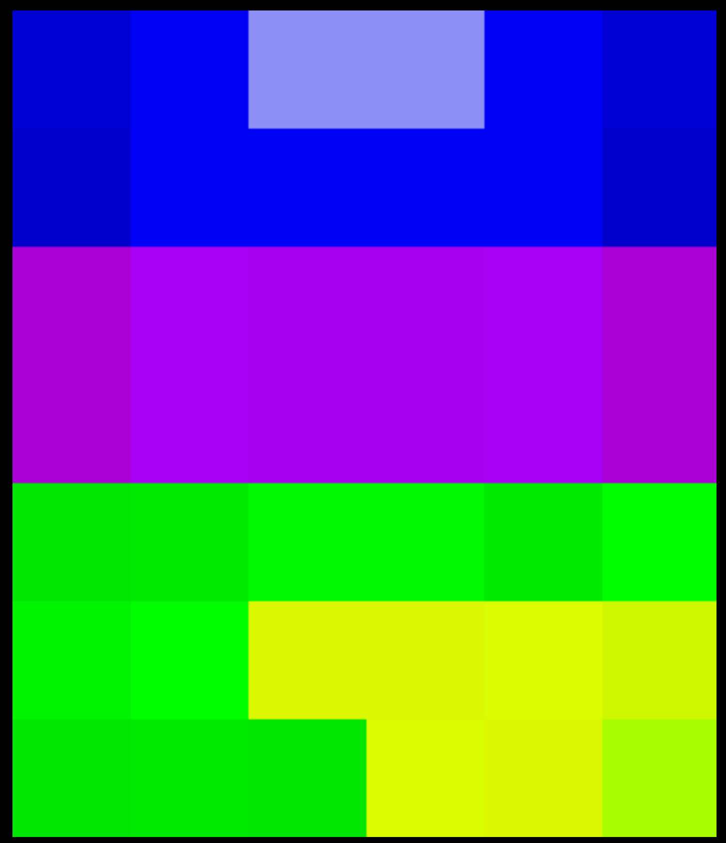
6x7 Pixel Input

1024x768 Pixel Guide

New Built-In CIFilters

NEW

CIEdgePreserveUpsampleFilter



6x7 Pixel Input

+

1024x768 Pixel Guide

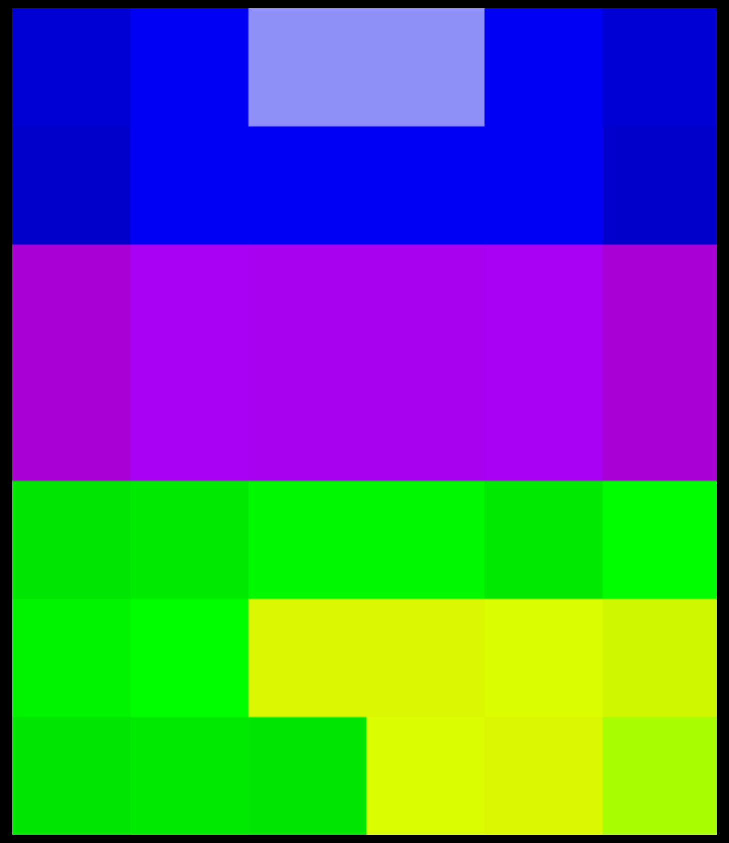
=

1024x768 Pixel Result

New Built-In CIFilters

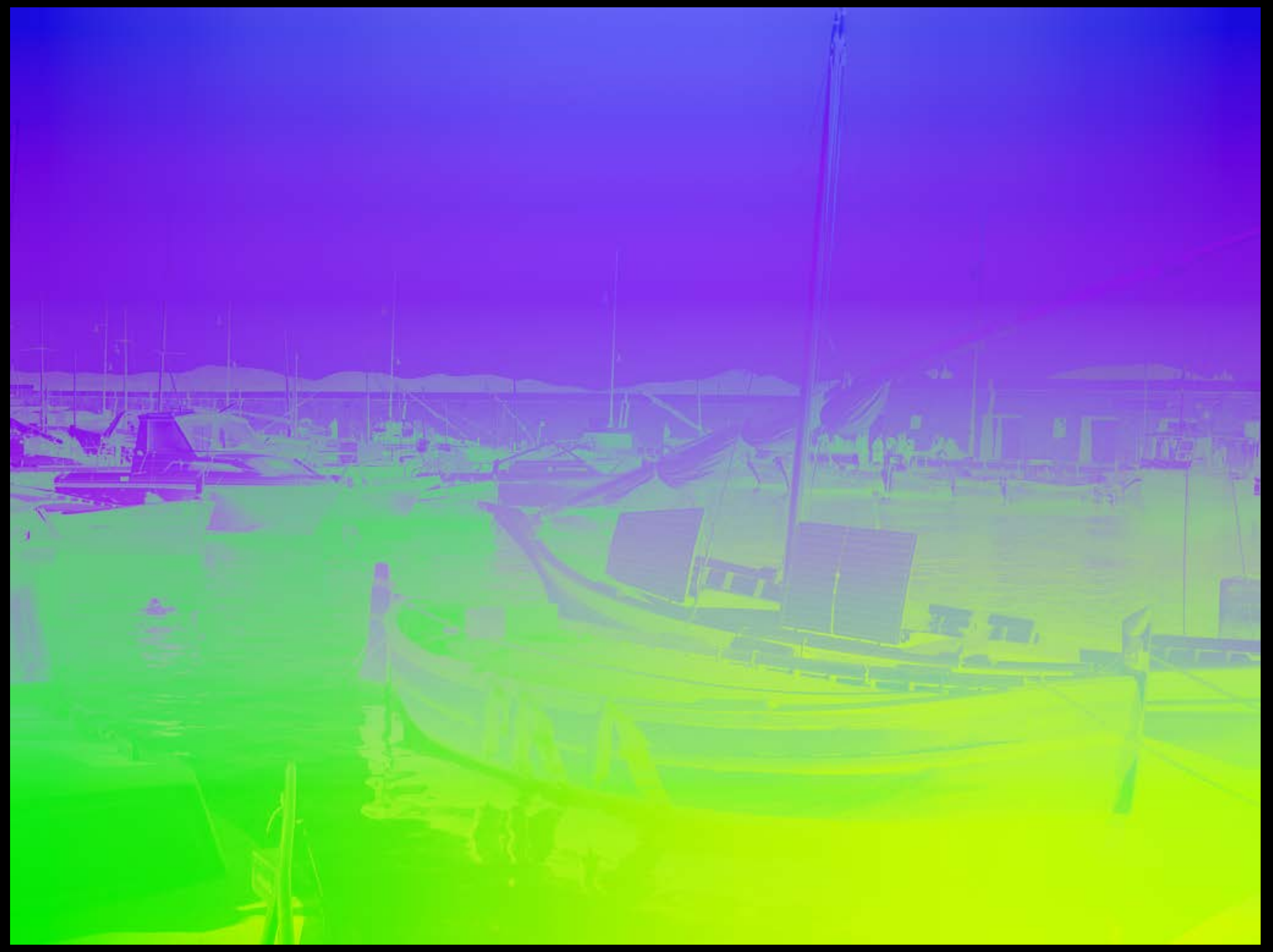
NEW

CIEdgePreserveUpsampleFilter



+

=



6x7 Pixel Input

1024x768 Pixel Guide

1024x768 Pixel Result

New Performance Controls

NEW

Metal on by default

New Performance Controls

NEW

Metal on by default

Now `UIImage(ciImage:)` is much faster

New Performance Controls

NEW

Metal on by default

Now `UIImage(ciImage:)` is much faster

Core Image supports input and output of half-float CGImageRefs

New Performance Controls

NEW

Metal on by default

Now `UIImage(ciImage:)` is much faster

Core Image supports input and output of half-float CGImageRefs

Pixel Format	Bytes Per Pixel	Bit Depth	Range	Quantization
RGBA8	4	8	0…1	linear

New Performance Controls

NEW

Metal on by default

Now `UIImage(ciImage:)` is much faster

Core Image supports input and output of half-float CGImageRefs

Pixel Format	Bytes Per Pixel	Bit Depth	Range	Quantization
RGBA8	4	8	0…1	linear
RGBAf	16	24	$-10^{38} \dots 10^{38}$	logarithmic

New Performance Controls

NEW

Metal on by default

Now `UIImage(ciImage:)` is much faster

Core Image supports input and output of half-float CGImageRefs

Pixel Format	Bytes Per Pixel	Bit Depth	Range	Quantization
RGBA8	4	8	0…1	linear
RGBAf	16	24	$-10^{38} \ldots 10^{38}$	logarithmic
RGBAh	8	10	–65519 … 65519	logarithmic

New Performance Controls

NEW

Metal on by default

Now `UIImage(ciImage:)` is much faster

Core Image supports input and output of half-float CGImageRefs

Pixel Format	Bytes Per Pixel	Bit Depth	Range	Quantization
RGBA8	4	8	0…1	linear
RGBAf	16	24	$-10^{38} \ldots 10^{38}$	logarithmic
RGBAh	8	10	–65519 … 65519	logarithmic
CVPixelFormat 30RGBLEPackedWideGamut	4	10	–0.37 … 1.62	gamma'd

Adjusting RAW Images with Core Image

Adjusting RAW Images with Core Image

What is a RAW file

Using the CIRAWFilter API

Supporting wide-gamut output

Managing memory

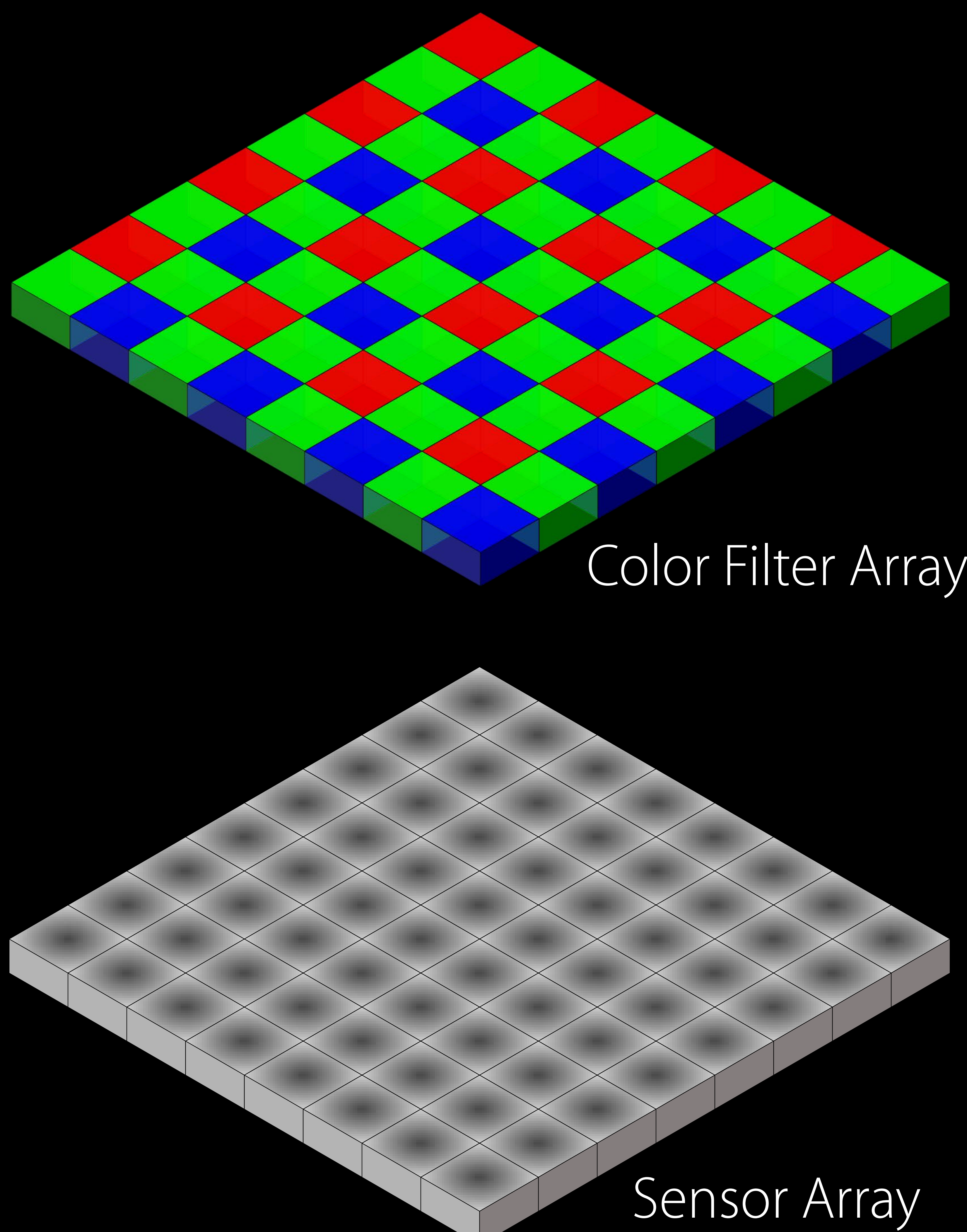
Adjusting RAW Images

What is a RAW file

Adjusting RAW Images

What is a RAW file

Most cameras use a color filter array
and a sensor array

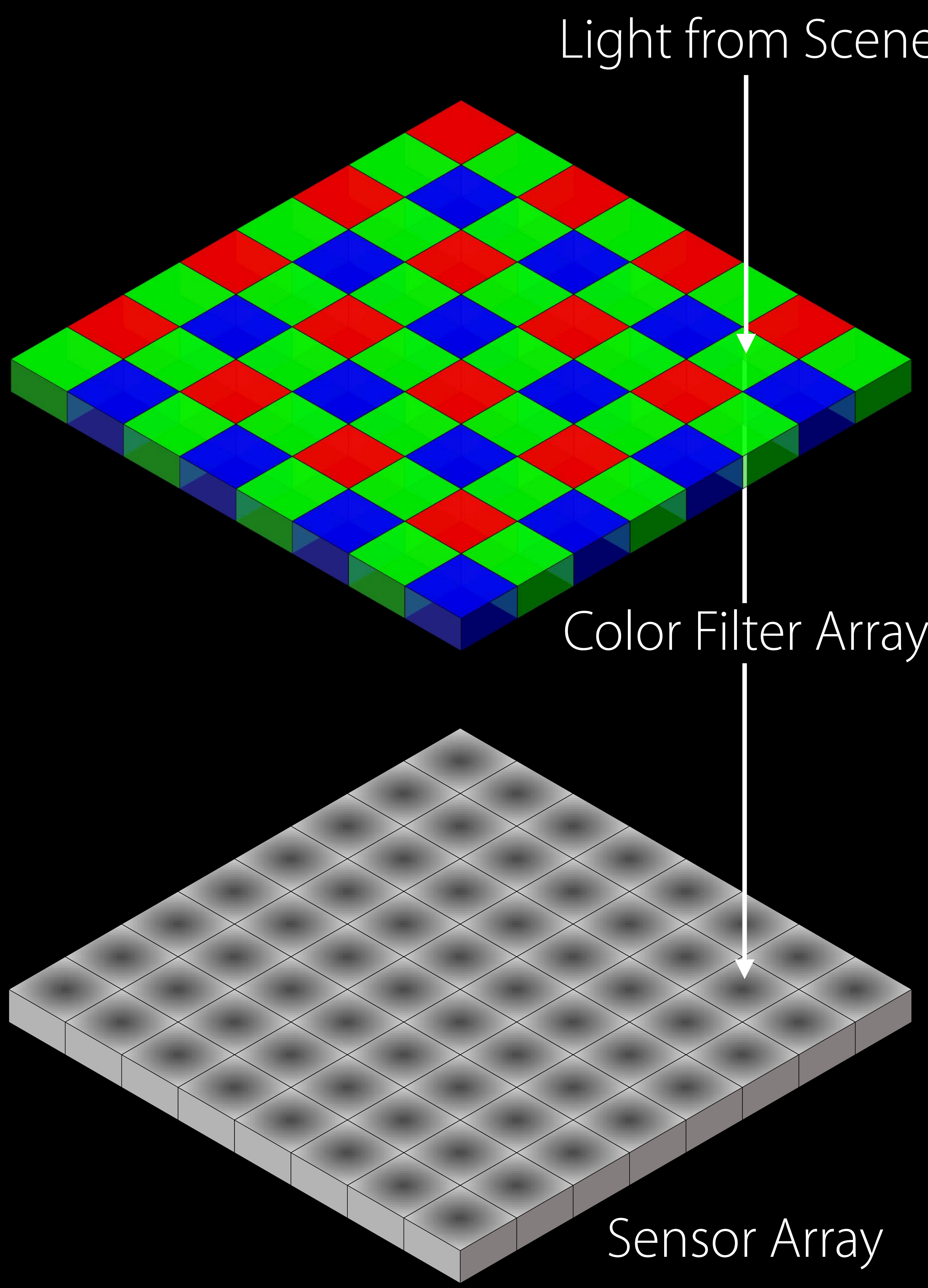


Adjusting RAW Images

What is a RAW file

Most cameras use a color filter array and a sensor array

Photons from the scene pass through the filter and are counted by the sensor

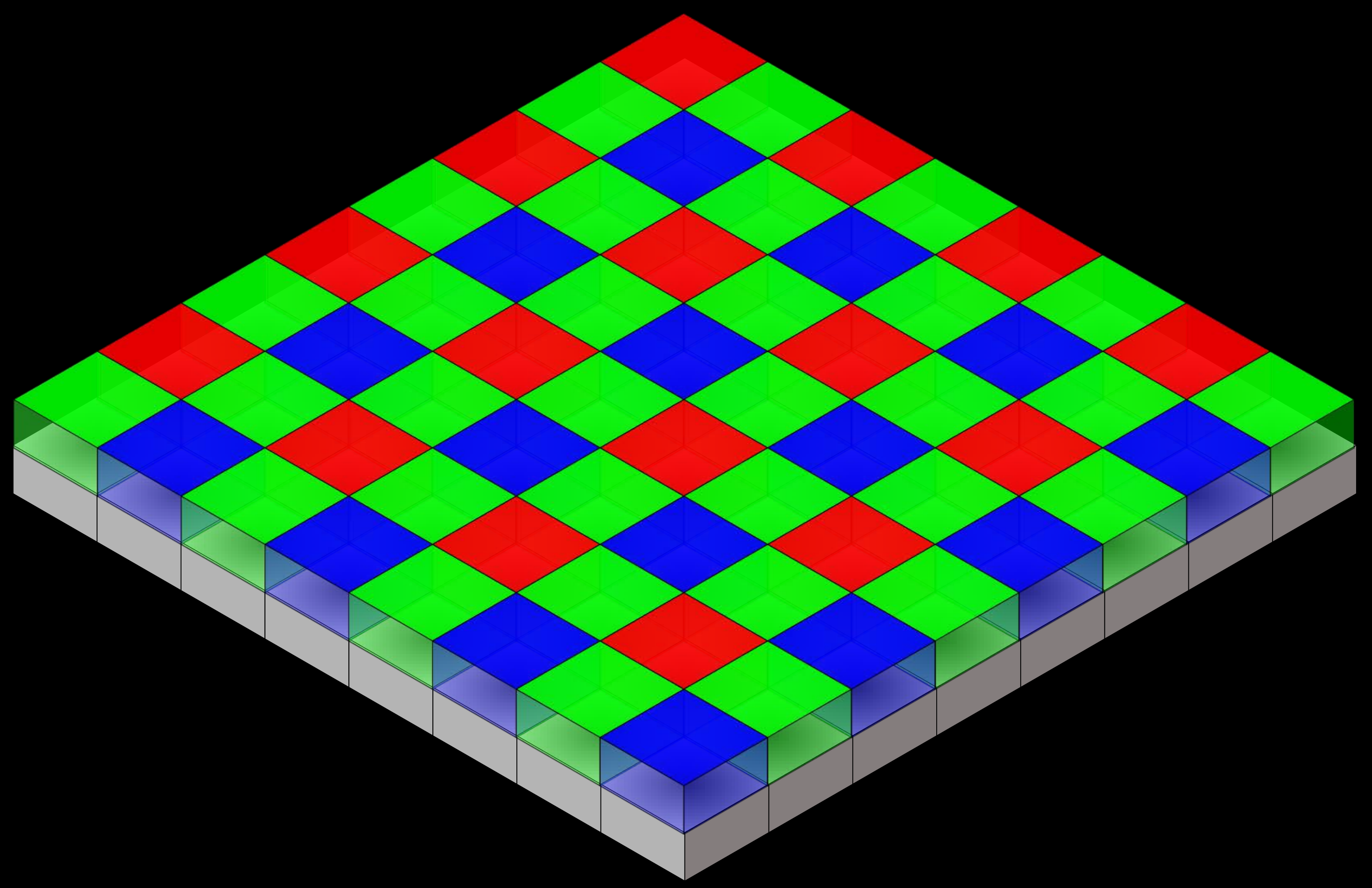


Adjusting RAW Images

What is a RAW file

Most cameras use a color filter array and a sensor array

Photons from the scene pass through the filter and are counted by the sensor

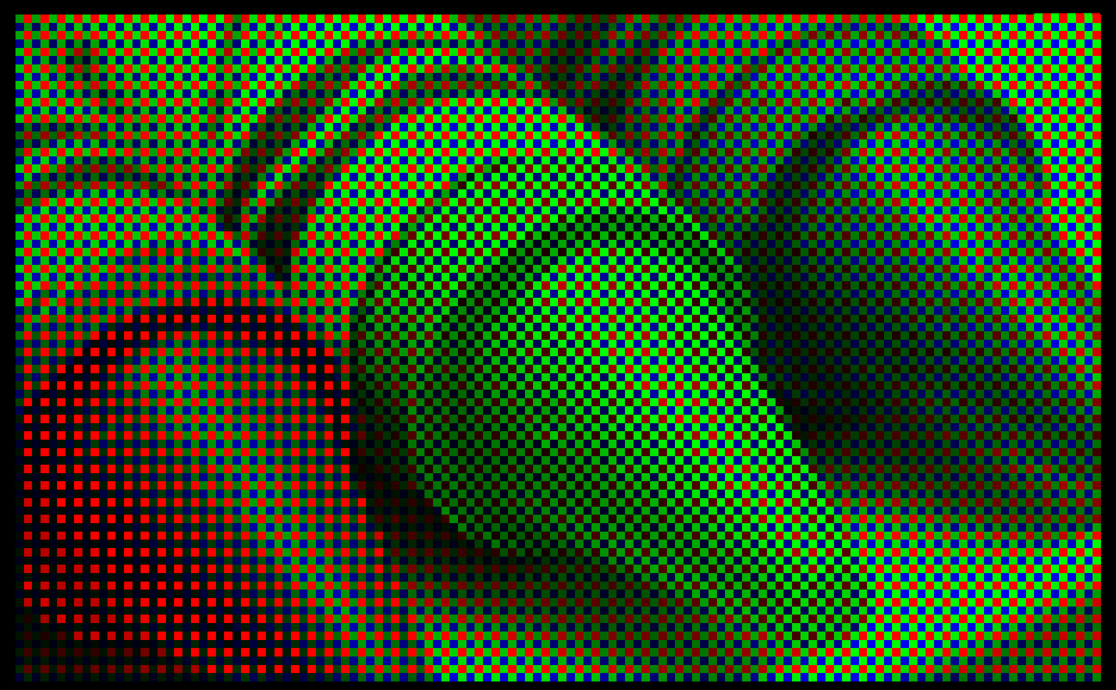


Adjusting RAW Images

What is a RAW file

Most cameras use a color filter array and a sensor array

Photons from the scene pass through the filter and are counted by the sensor



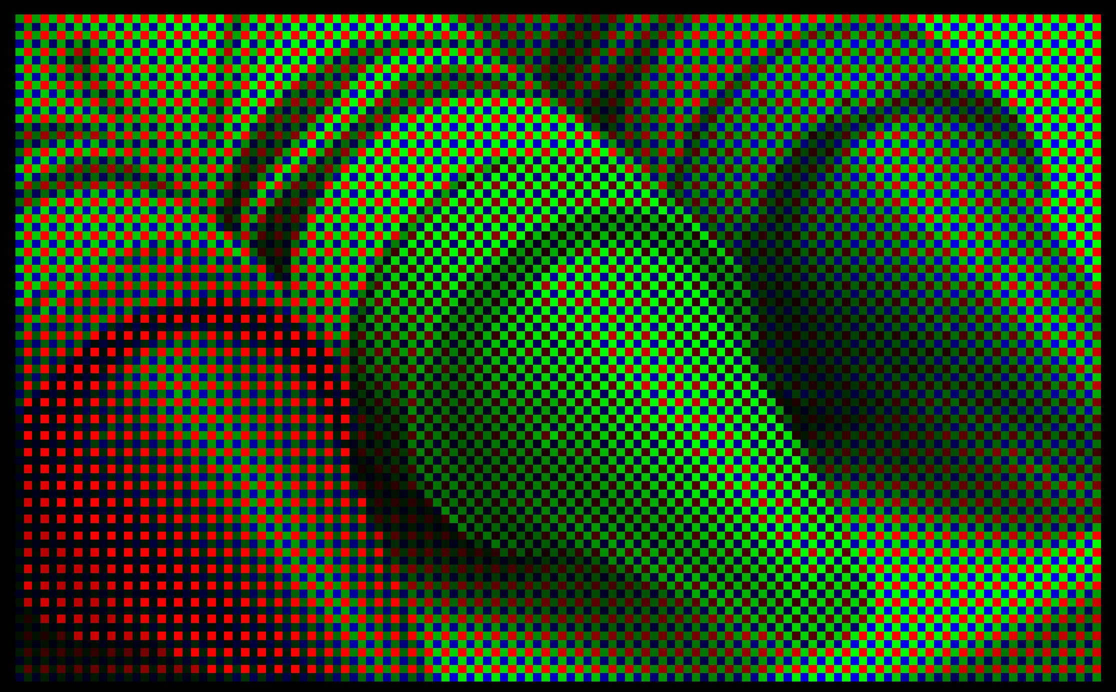
Adjusting RAW Images

What is a RAW file

Most cameras use a color filter array and a sensor array

Photons from the scene pass through the filter and are counted by the sensor

Advanced image processing is required to develop the RAW sensor data into an image suitable for output



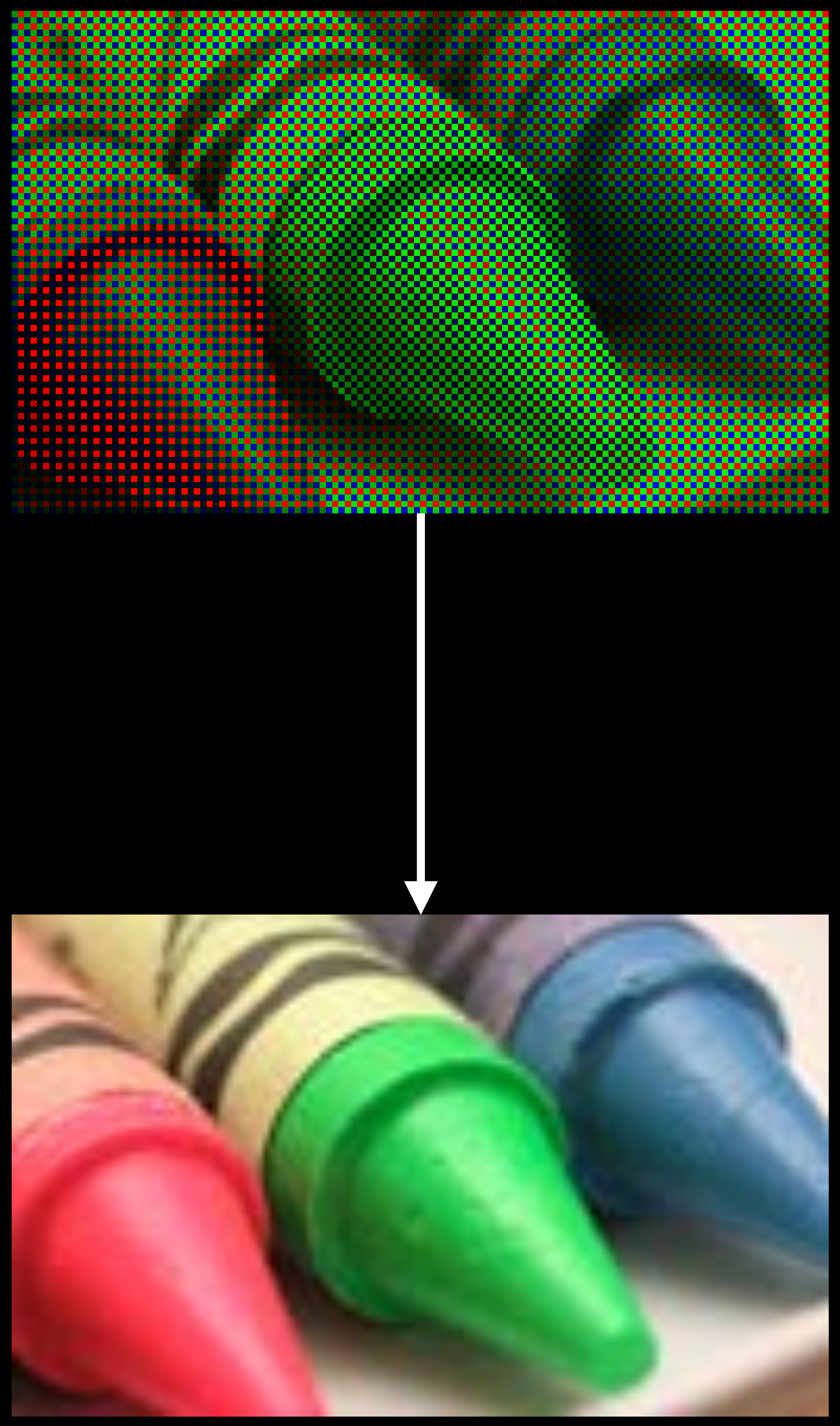
Adjusting RAW Images

What is a RAW file

Most cameras use a color filter array and a sensor array

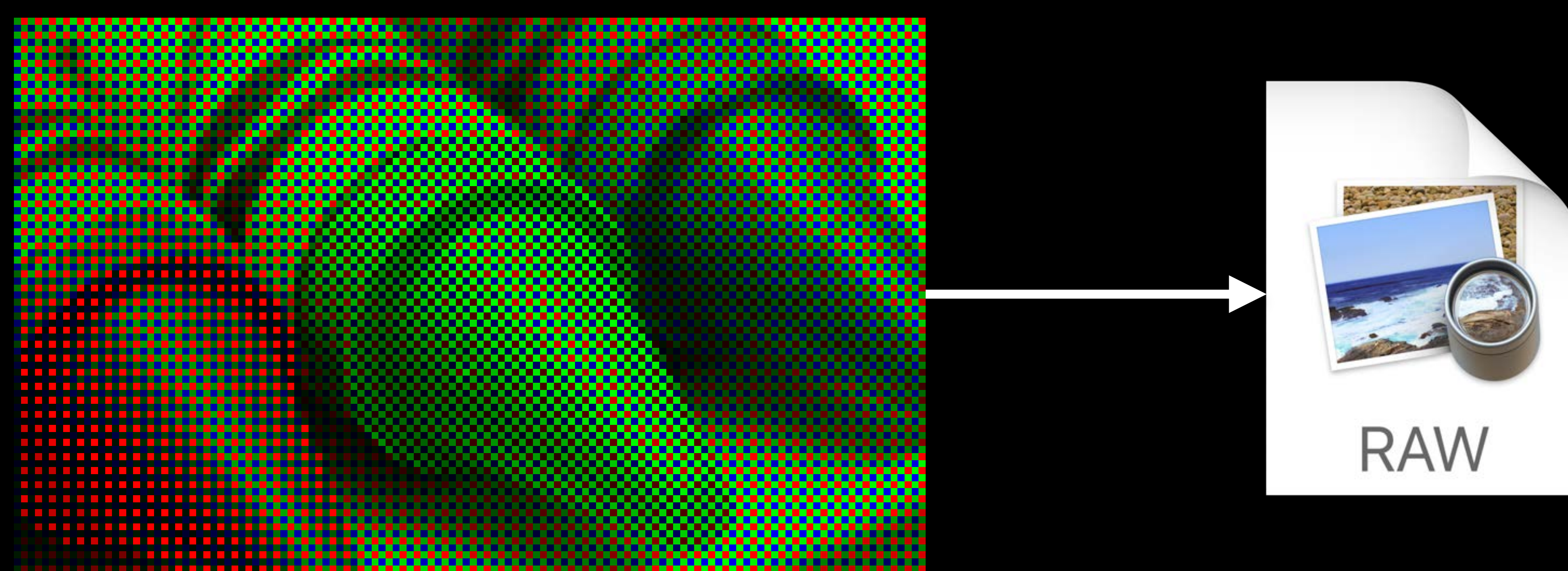
Photons from the scene pass through the filter and are counted by the sensor

Advanced image processing is required to develop the RAW sensor data into an image suitable for output



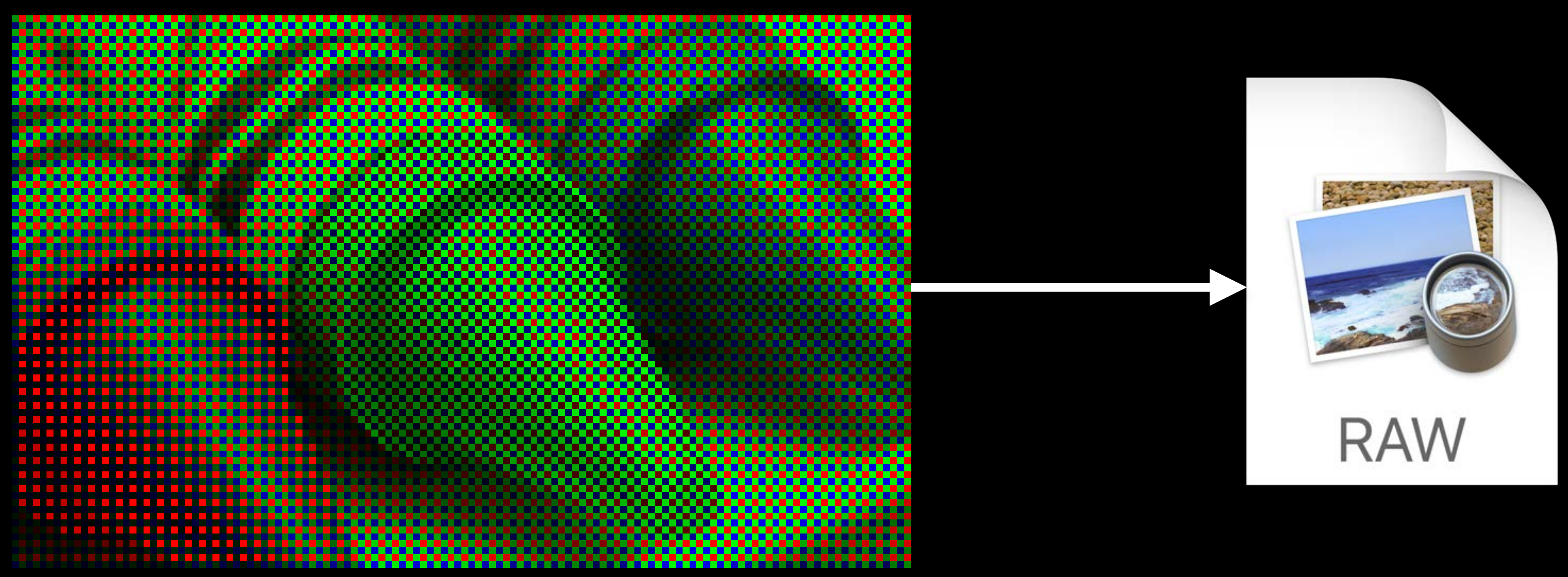
Adjusting RAW Images

What is a RAW file



Adjusting RAW Images

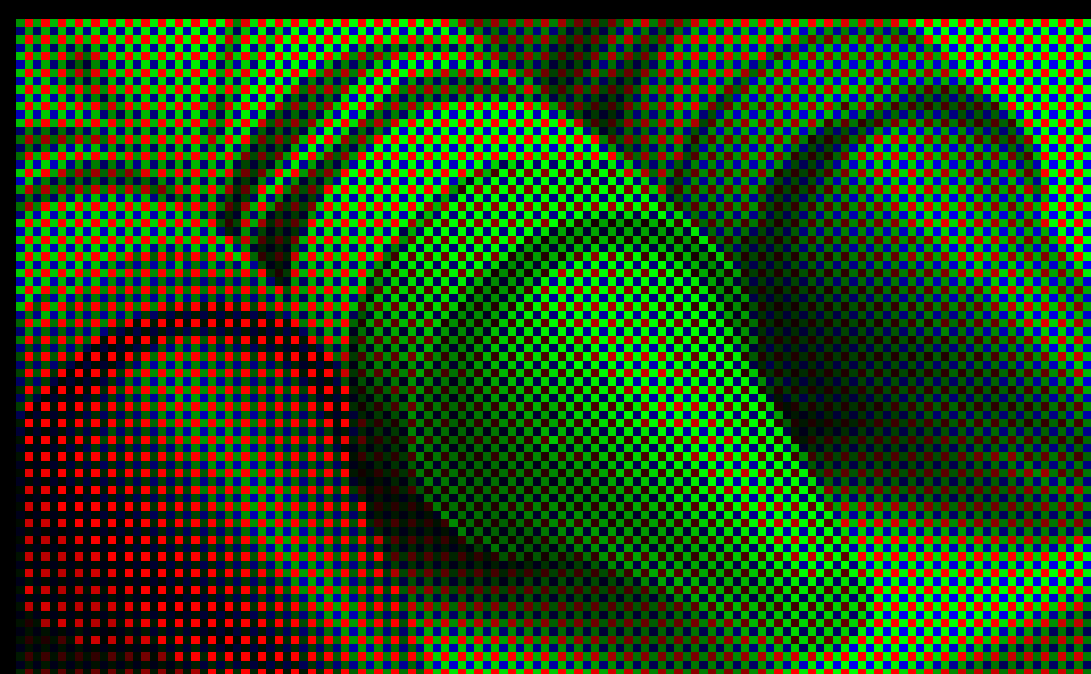
What is a RAW file

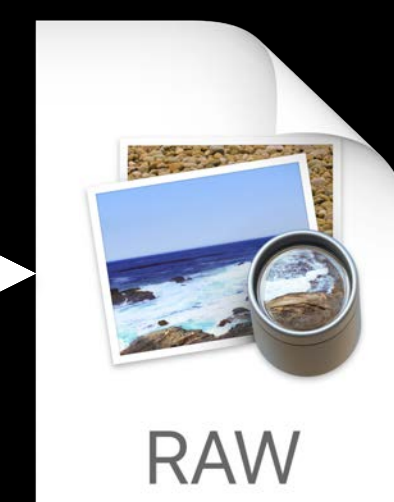


RAW files store unprocessed scene data

Adjusting RAW Images

What is a RAW file



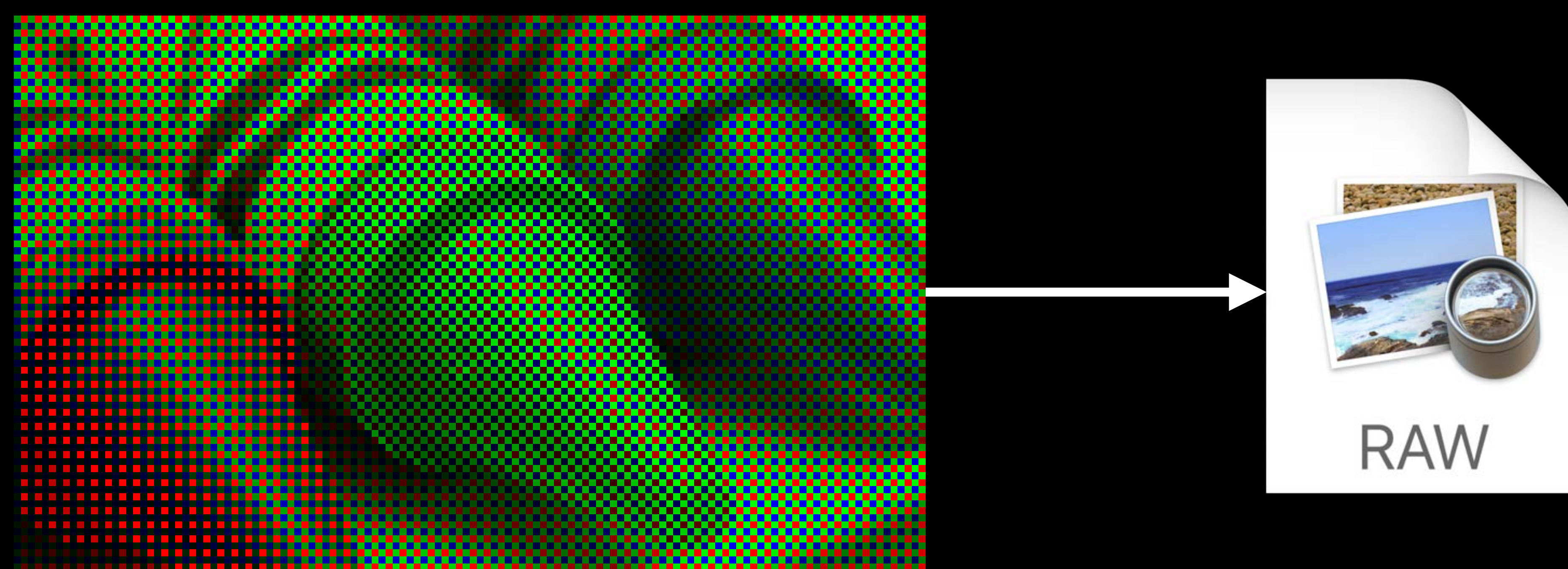


RAW files store unprocessed scene data

JPG files store processed output images

Adjusting RAW Images

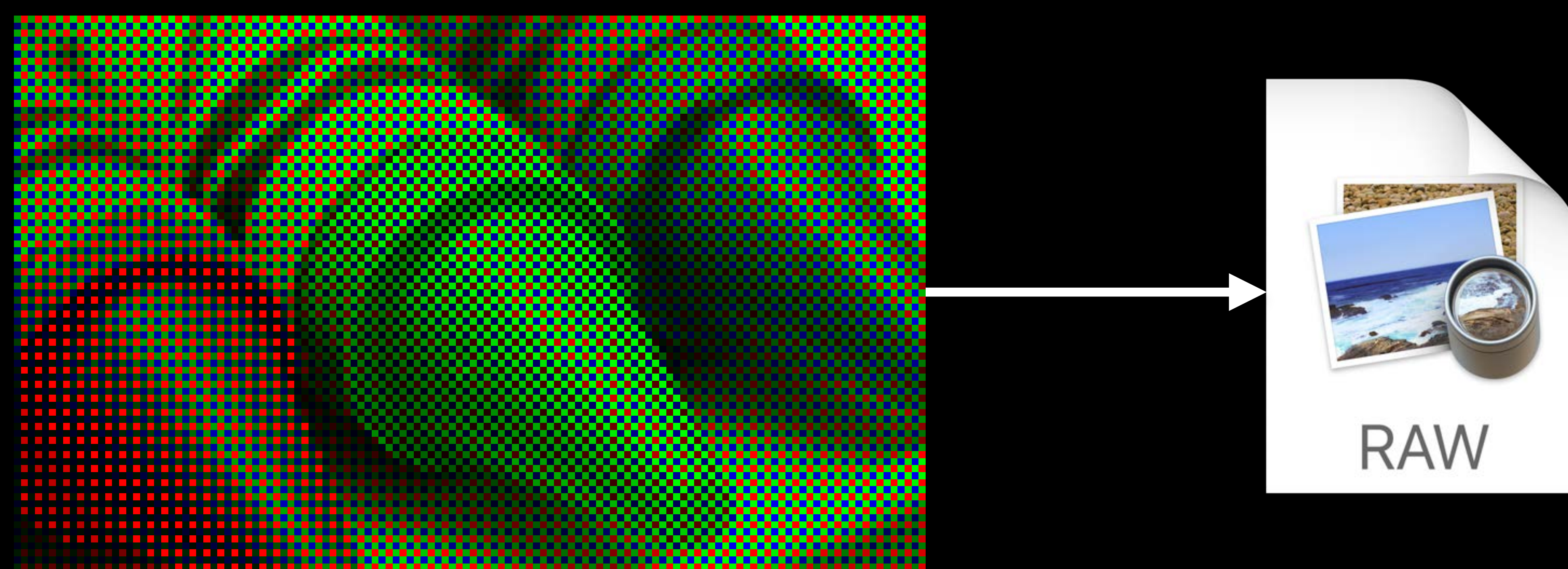
What is a RAW file



JPG files store processed output images

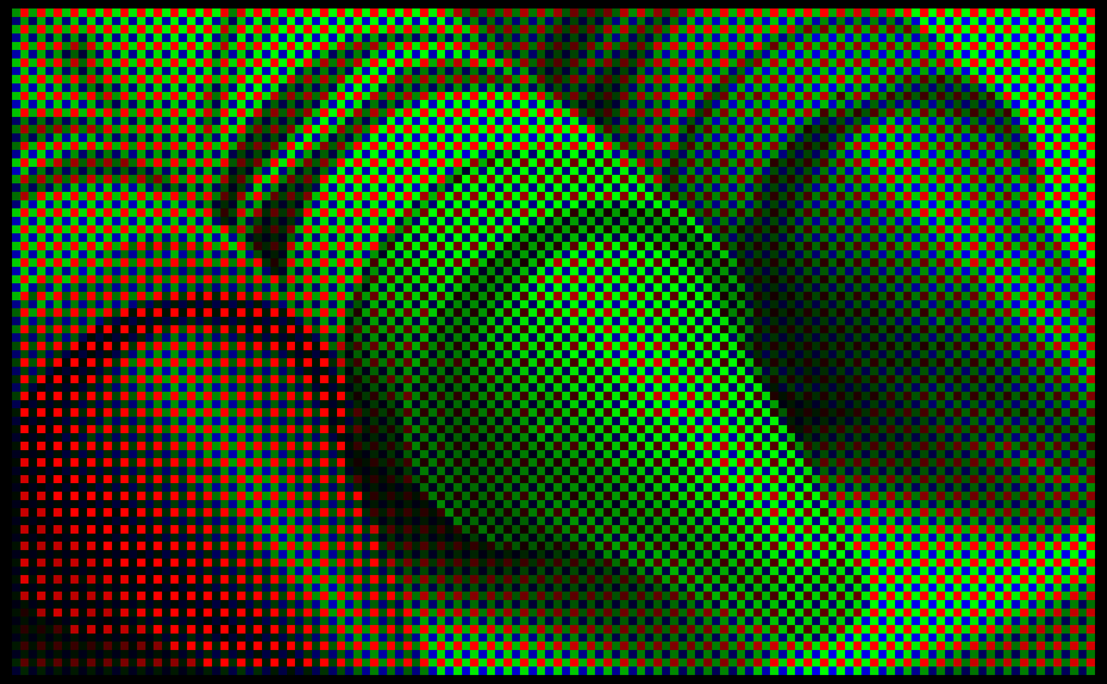
Adjusting RAW Images

What is a RAW file



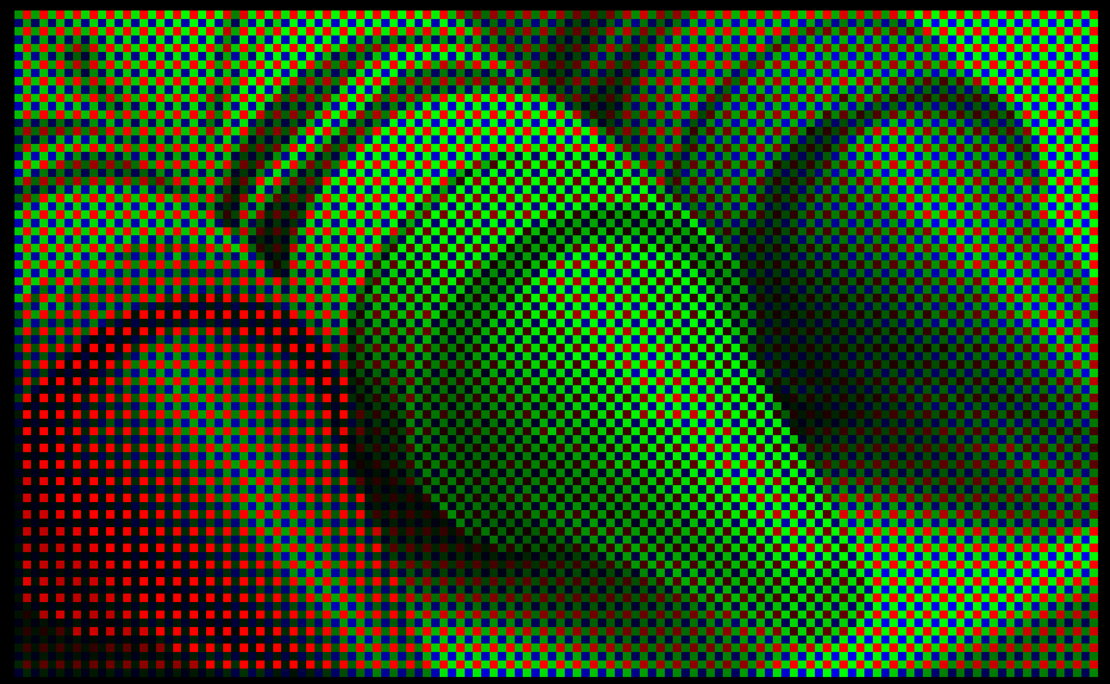
Adjusting RAW Images

Stages of RAW image processing



Adjusting RAW Images

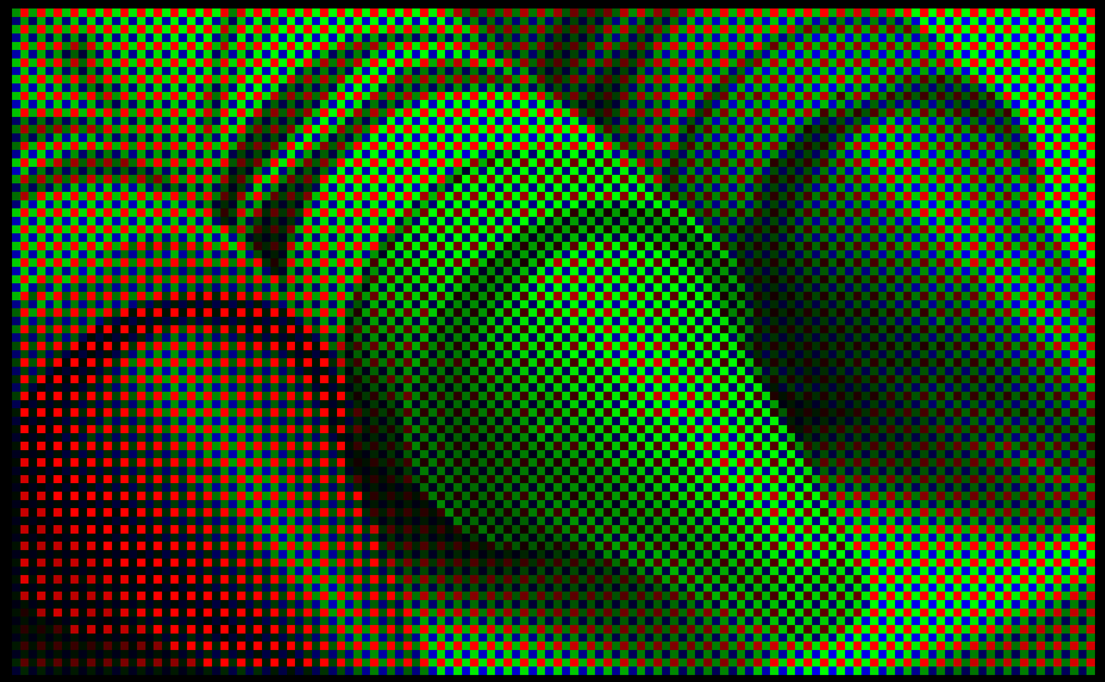
Stages of RAW image processing



Extract critical metadata

Adjusting RAW Images

Stages of RAW image processing

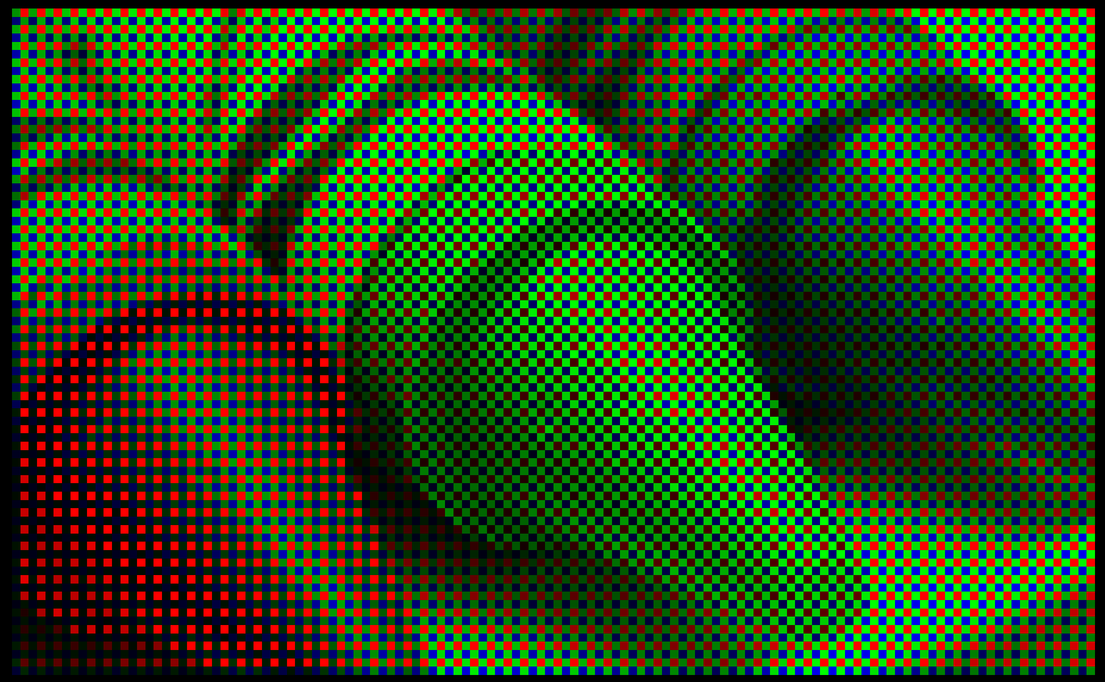


Extract critical metadata

Decode RAW sensor image

Adjusting RAW Images

Stages of RAW image processing



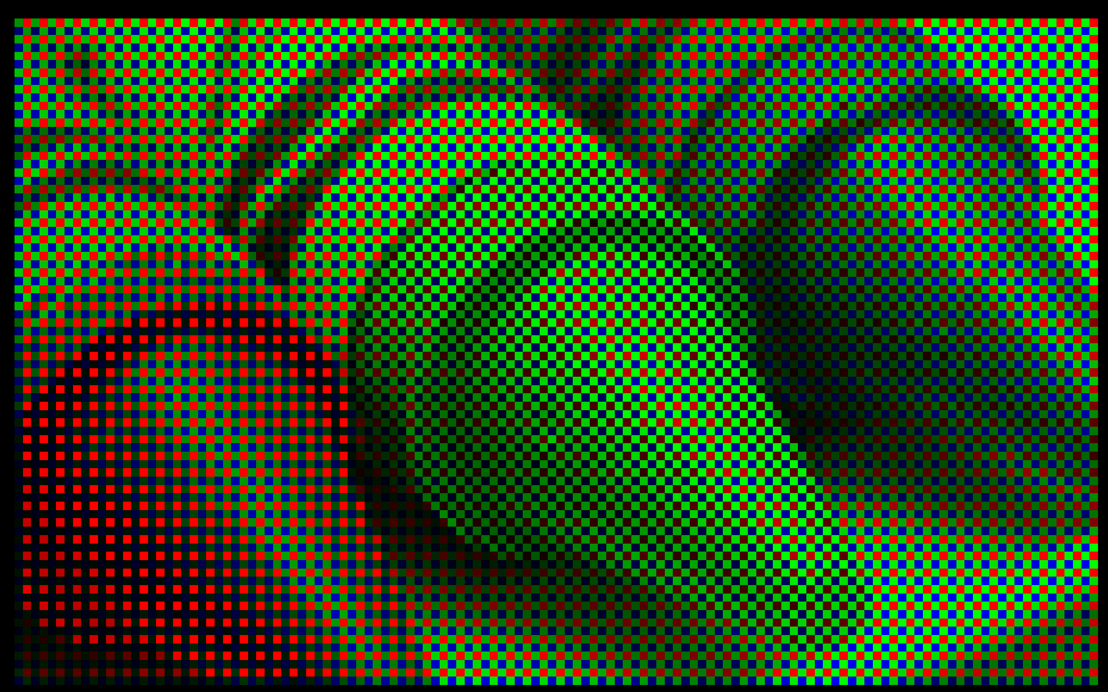
Extract critical metadata

Decode RAW sensor image

De-mosaic reconstruction

Adjusting RAW Images

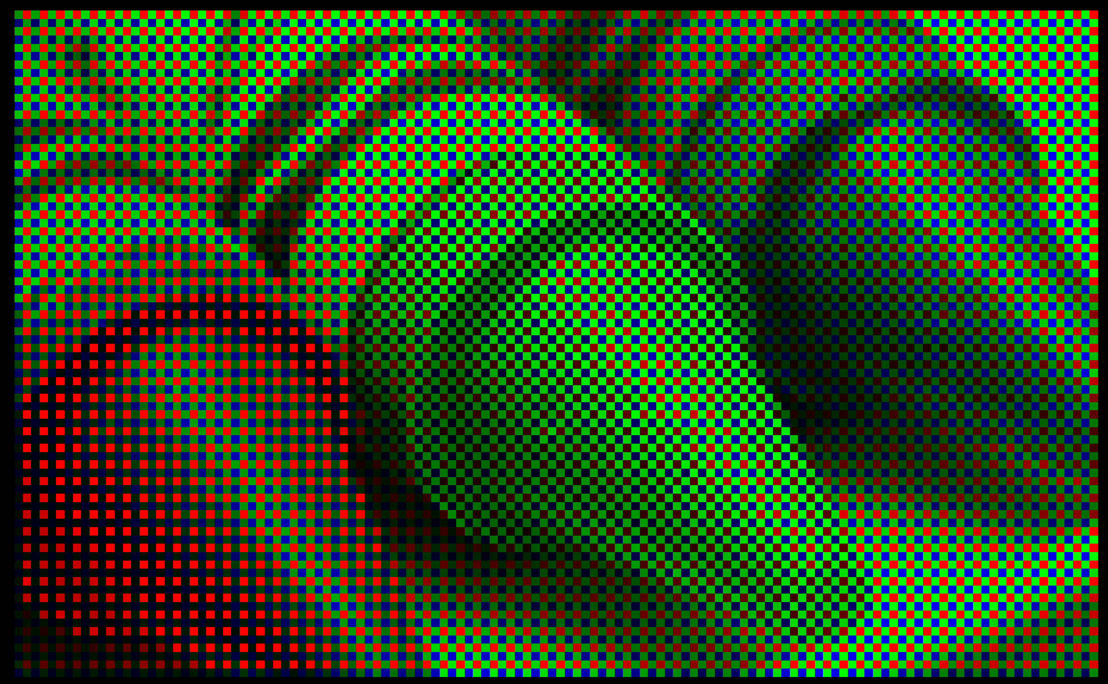
Stages of RAW image processing



- Extract critical metadata
- Decode RAW sensor image
- De-mosaic reconstruction
- Apply lens correction

Adjusting RAW Images

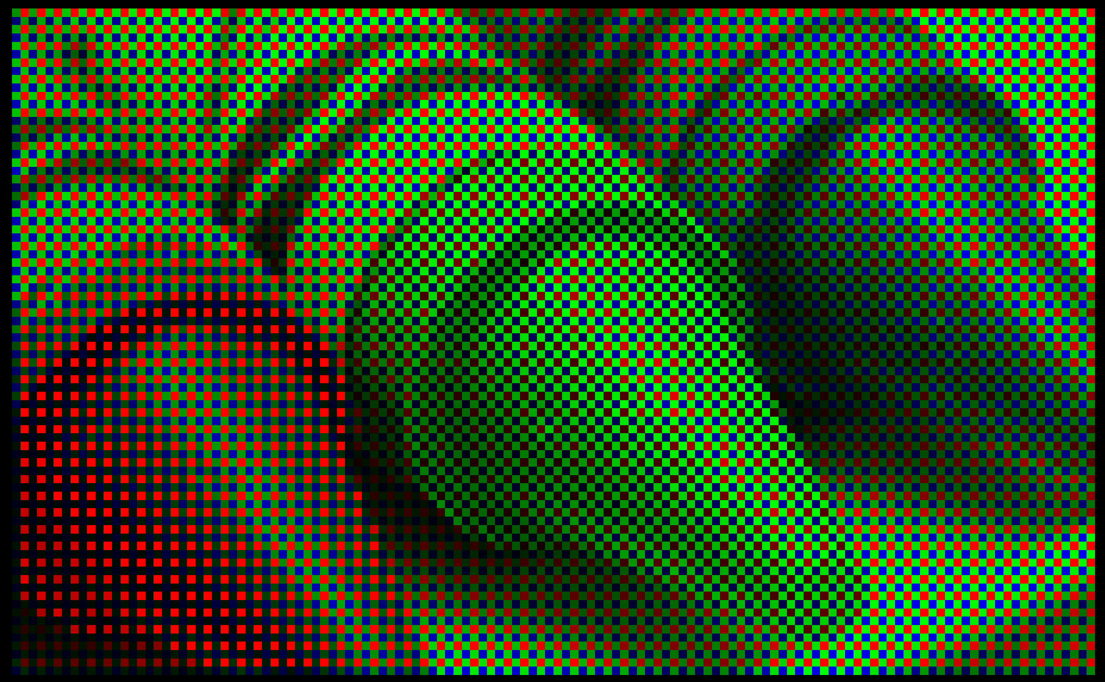
Stages of RAW image processing



- Extract critical metadata
- Decode RAW sensor image
- De-mosaic reconstruction
- Apply lens correction
- Reduce noise

Adjusting RAW Images

Stages of RAW image processing



Extract critical metadata

Decode RAW sensor image

De-mosaic reconstruction

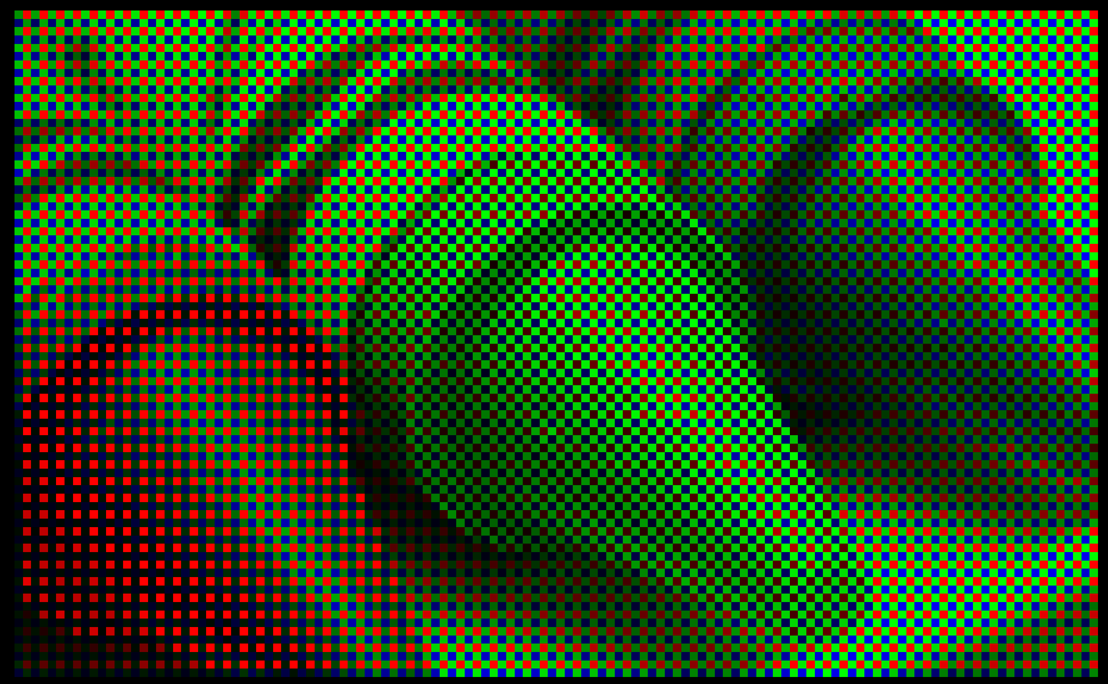
Apply lens correction

Reduce noise

Color-match scene-referred sensor values to output-referred color space

Adjusting RAW Images

Stages of RAW image processing



Extract critical metadata

Decode RAW sensor image

De-mosaic reconstruction

Apply lens correction

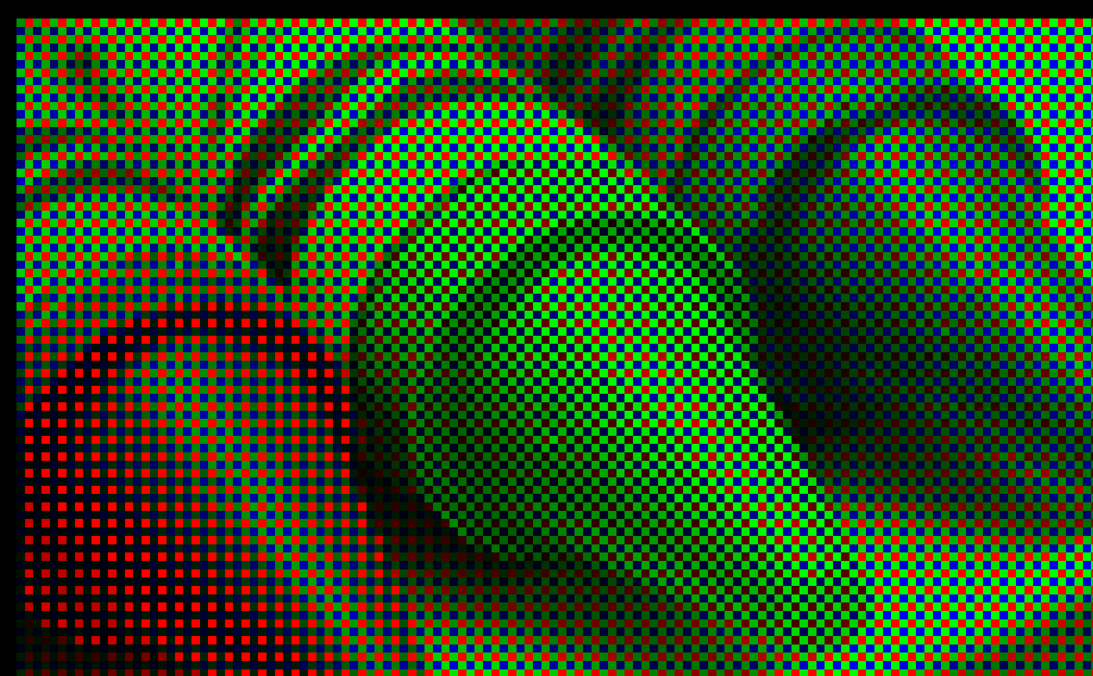
Reduce noise

Color-match scene-referred sensor values to output-referred color space

Adjust exposure and temperature/tint

Adjusting RAW Images

Stages of RAW image processing



Extract critical metadata

Decode RAW sensor image

De-mosaic reconstruction

Apply lens correction

Reduce noise

Color-match scene-referred sensor values to output-referred color space

Adjust exposure and temperature/tint

Add sharpening, contrast, and saturation

Adjusting RAW Images

Advantages of RAW

Adjusting RAW Images

Advantages of RAW

Contains linear and deep pixel data which enables great editability

Adjusting RAW Images

Advantages of RAW

Contains linear and deep pixel data which enables great editability

Image processing gets better every year

Adjusting RAW Images

Advantages of RAW

Contains linear and deep pixel data which enables great editability

Image processing gets better every year

Can be rendered to any color space

Adjusting RAW Images

Advantages of RAW

Contains linear and deep pixel data which enables great editability

Image processing gets better every year

Can be rendered to any color space

Users can use different software to interpret the image

Adjusting RAW Images

Advantages of JPEG

Adjusting RAW Images

Advantages of JPEG

Fast to load and display

Adjusting RAW Images

Advantages of JPEG

Fast to load and display

Contains colors targeting a specific color space

Adjusting RAW Images

Advantages of JPEG

Fast to load and display

Contains colors targeting a specific color space

Predictable results

Adjusting RAW Images

Advantages of JPEG

Fast to load and display

Contains colors targeting a specific color space

Predictable results

Cameras can provide a great default image for display

Adjusting RAW Images

Advantages of JPEG

Fast to load and display

Contains colors targeting a specific color space

Predictable results

Cameras can provide a great default image for display

- iOS cameras are a good example of this

Adjusting RAW Images

Platform support

Adjusting RAW Images

Platform support

Now Core Image fully supports RAW on iOS and tvOS

Adjusting RAW Images

Platform support

Now Core Image fully supports RAW on iOS and tvOS

- Supports over 400 unique camera models from 16 vendors

Adjusting RAW Images

Platform support

Now Core Image fully supports RAW on iOS and tvOS

- Supports over 400 unique camera models from 16 vendors
- Also supports DNG files captured from iOS devices

Adjusting RAW Images

Platform support

Now Core Image fully supports RAW on iOS and tvOS

- Supports over 400 unique camera models from 16 vendors
- Also supports DNG files captured from iOS devices
  - iSight cameras on iPhone 6S, iPhone 6S Plus, iPhone SE, iPad Pro (9.7-inch)

Adjusting RAW Images

Platform support

Now Core Image fully supports RAW on iOS and tvOS

- Supports over 400 unique camera models from 16 vendors
- Also supports DNG files captured from iOS devices
  - iSight cameras on iPhone 6S, iPhone 6S Plus, iPhone SE, iPad Pro (9.7-inch)

Adjusting RAW Images

Platform support

Now Core Image fully supports RAW on iOS and tvOS

- Supports over 400 unique camera models from 16 vendors
- Also supports DNG files captured from iOS devices
  - iSight cameras on iPhone 6S, iPhone 6S Plus, iPhone SE, iPad Pro (9.7-inch)
- The same high-performance RAW pipeline as on macOS

Adjusting RAW Images

Platform support

Now Core Image fully supports RAW on iOS and tvOS

- Supports over 400 unique camera models from 16 vendors
- Also supports DNG files captured from iOS devices
  - iSight cameras on iPhone 6S, iPhone 6S Plus, iPhone SE, iPad Pro (9.7-inch)
- The same high-performance RAW pipeline as on macOS
- Requires A8 or newer processor (iOS GPU Family 2)

Adjusting RAW Images

Platform support

Adjusting RAW Images

Platform support

We continuously add support for cameras and improve quality

Adjusting RAW Images

Platform support

We continuously add support for cameras and improve quality

- New cameras are added in software updates

Adjusting RAW Images

Platform support

We continuously add support for cameras and improve quality

- New cameras are added in software updates
- Pipeline improvements are versioned

Demo

Adjusting images on iOS

Adjusting RAW Images

CIRAWFilter API lets you control the stages

CIRAWFilter gives your application:

Adjusting RAW Images

CIRAWFilter API lets you control the stages

CIRAWFilter gives your application:

- CIImage with wide gamut, extended range, half-float precision

Adjusting RAW Images

CIRAWFilter API lets you control the stages

CIRAWFilter gives your application:

- CIImage with wide gamut, extended range, half-float precision
- Easy control over RAW processing parameters

Adjusting RAW Images

CIRAWFilter API lets you control the stages

CIRAWFilter gives your application:

- CIImage with wide gamut, extended range, half-float precision
- Easy control over RAW processing parameters
- Fast, interactive performance using GPU

Adjusting RAW Images

Using the CIRAWFilter API

RAW Image File

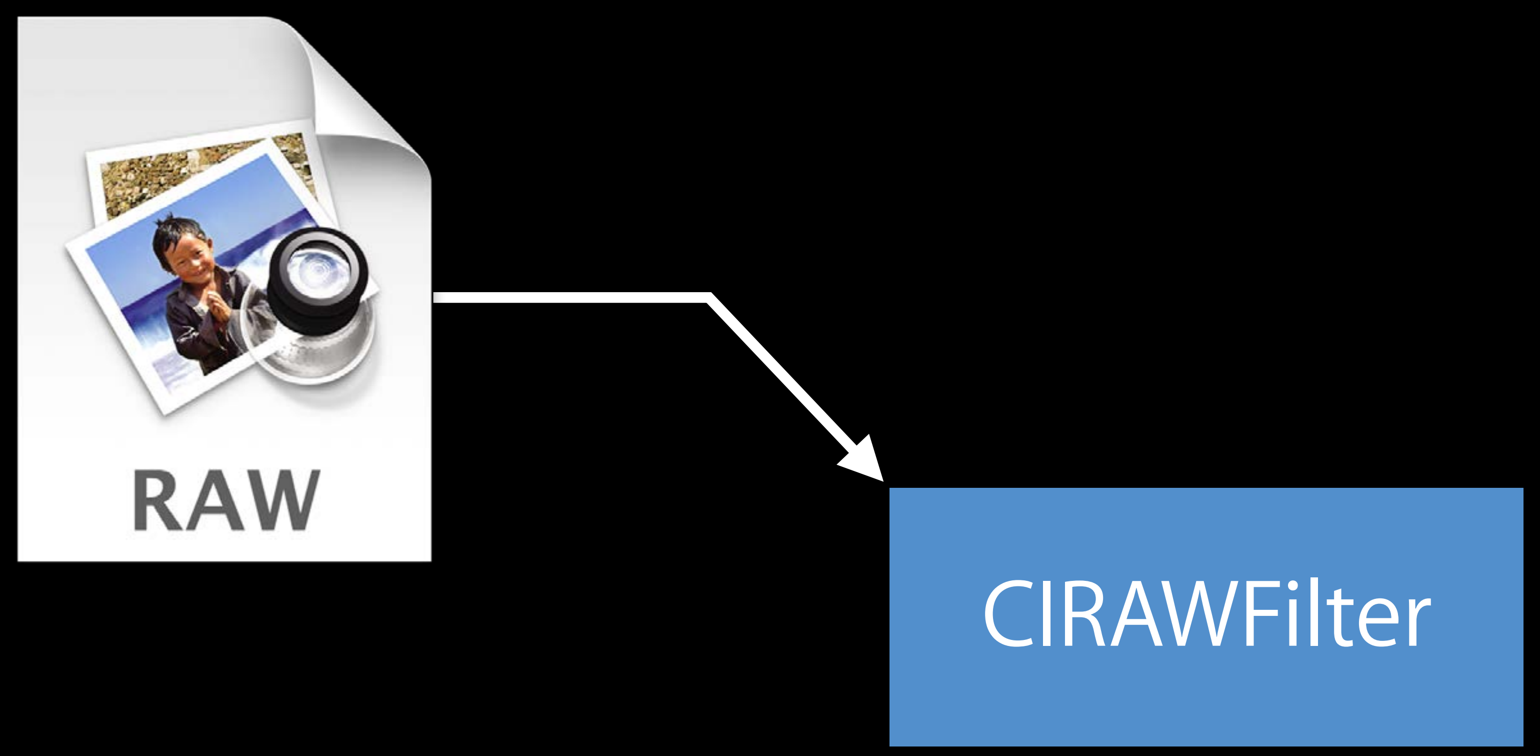
- File URL
- File data
- CVPixelBuffer

Adjusting RAW Images

Using the CIRAWFilter API

RAW Image File

- File URL
- File data
- CVPixelBuffer



Adjusting RAW Images

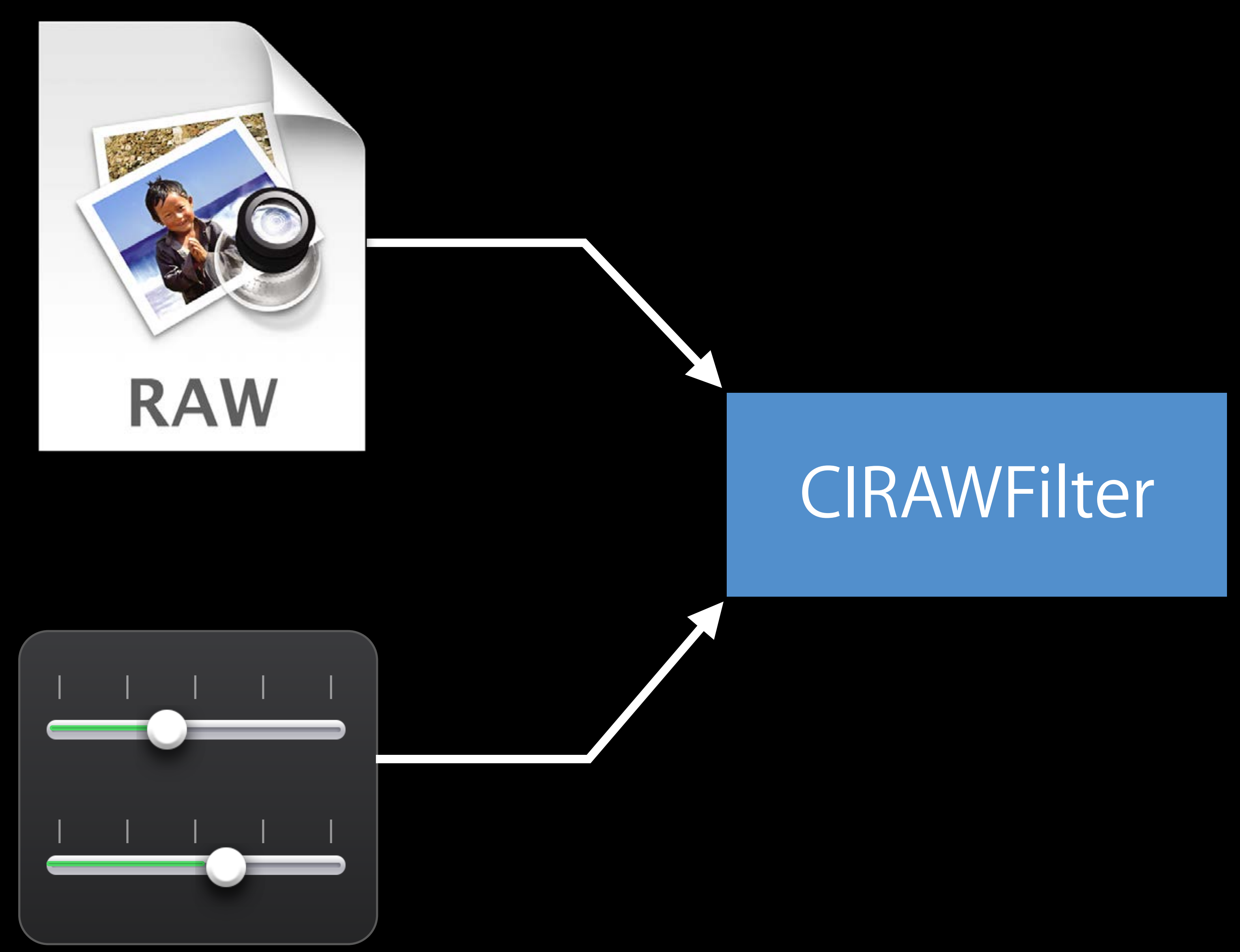
Using the CIRAWFilter API

RAW Image File

- File URL
- File data
- CVPixelBuffer

User Adjustments

- Exposure
- Temperature, tint
- Noise reduction



Adjusting RAW Images

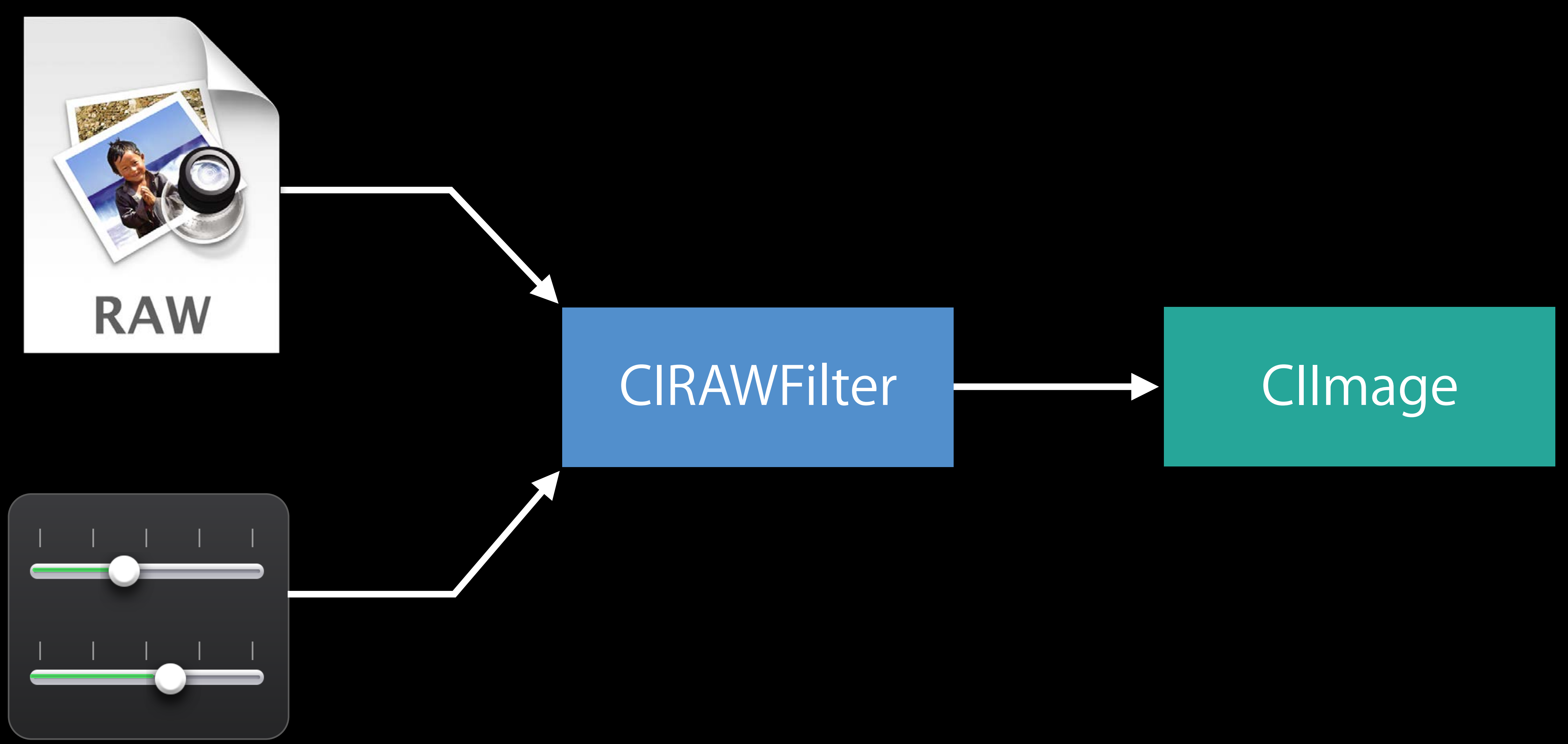
Using the CIRAWFilter API

RAW Image File

- File URL
- File data
- CVPixelBuffer

User Adjustments

- Exposure
- Temperature, tint
- Noise reduction



```swift
// Using the CIRAWFilter API

func getAdjustedRAW(url: URL) -> CIImage?
{
    // Load the image
    let f = CIFilter(imageURL: url, options:nil)

    // Get the NR amount
    if let nr = f.value(forKey: kCIInputLuminanceNoiseReductionAmountKey) {

        // Change the NR amount
        f.setValue(nr.doubleValue + 0.1,
                   forKey: kCIInputLuminanceNoiseReductionAmountKey)
    }

    // Get the adjusted image
    return f.outputImage
}
```

```swift
// Using the CIRAWFilter API

func getAdjustedRAW(url: URL) -> CIImage?
{
    // Load the image
    let f = CIFilter(imageURL: url, options:nil)

    // Get the NR amount
    if let nr = f.value(forKey: kCIInputLuminanceNoiseReductionAmountKey) {

        // Change the NR amount
        f.setValue(nr.doubleValue + 0.1,
                   forKey: kCIInputLuminanceNoiseReductionAmountKey)
    }

    // Get the adjusted image
    return f.outputImage
}
```

```swift
// Using the CIRAWFilter API

func getAdjustedRAW(url: URL) -> CIImage?
{
    // Load the image
    let f = CIFilter(imageURL: url, options:nil)

    // Get the NR amount
    if let nr = f.value(forKey: kCIInputLuminanceNoiseReductionAmountKey) {

        // Change the NR amount
        f.setValue(nr.doubleValue + 0.1,
                   forKey: kCIInputLuminanceNoiseReductionAmountKey)
    }

    // Get the adjusted image
    return f.outputImage
}
```

```swift
// Using the CIRAWFilter API

func getAdjustedRAW(url: URL) -> CIImage?
{
    // Load the image
    let f = CIFilter(imageURL: url, options:nil)

    // Get the NR amount
    if let nr = f.value(forKey: kCIInputLuminanceNoiseReductionAmountKey) {

        // Change the NR amount
        f.setValue(nr.doubleValue + 0.1,
                   forKey: kCIInputLuminanceNoiseReductionAmountKey)
    }

    // Get the adjusted image
    return f.outputImage
}
```

```swift
// Using the CIRAWFilter API

func getAdjustedRAW(url: URL) -> CIImage?
{
    // Load the image
    let f = CIFilter(imageURL: url, options:nil)

    // Get the NR amount
    if let nr = f.value(forKey: kCIInputLuminanceNoiseReductionAmountKey) {

        // Change the NR amount
        f.setValue(nr.doubleValue + 0.1,
                   forKey: kCIInputLuminanceNoiseReductionAmountKey)
    }

    // Get the adjusted image
    return f.outputImage
}
```

```swift
// Using the CIRAWFilter API

func getAdjustedRAW(url: URL) -> CIImage?
{
    // Load the image
    let f = CIFilter(imageURL: url, options:nil)

    // Get the NR amount
    if let nr = f.value(forKey: kCIInputLuminanceNoiseReductionAmountKey) {

        // Change the NR amount
        f.setValue(nr.doubleValue + 0.1,
                   forKey: kCIInputLuminanceNoiseReductionAmountKey)
    }

    // Get the adjusted image
    return f.outputImage
}
```

Adjusting RAW Images

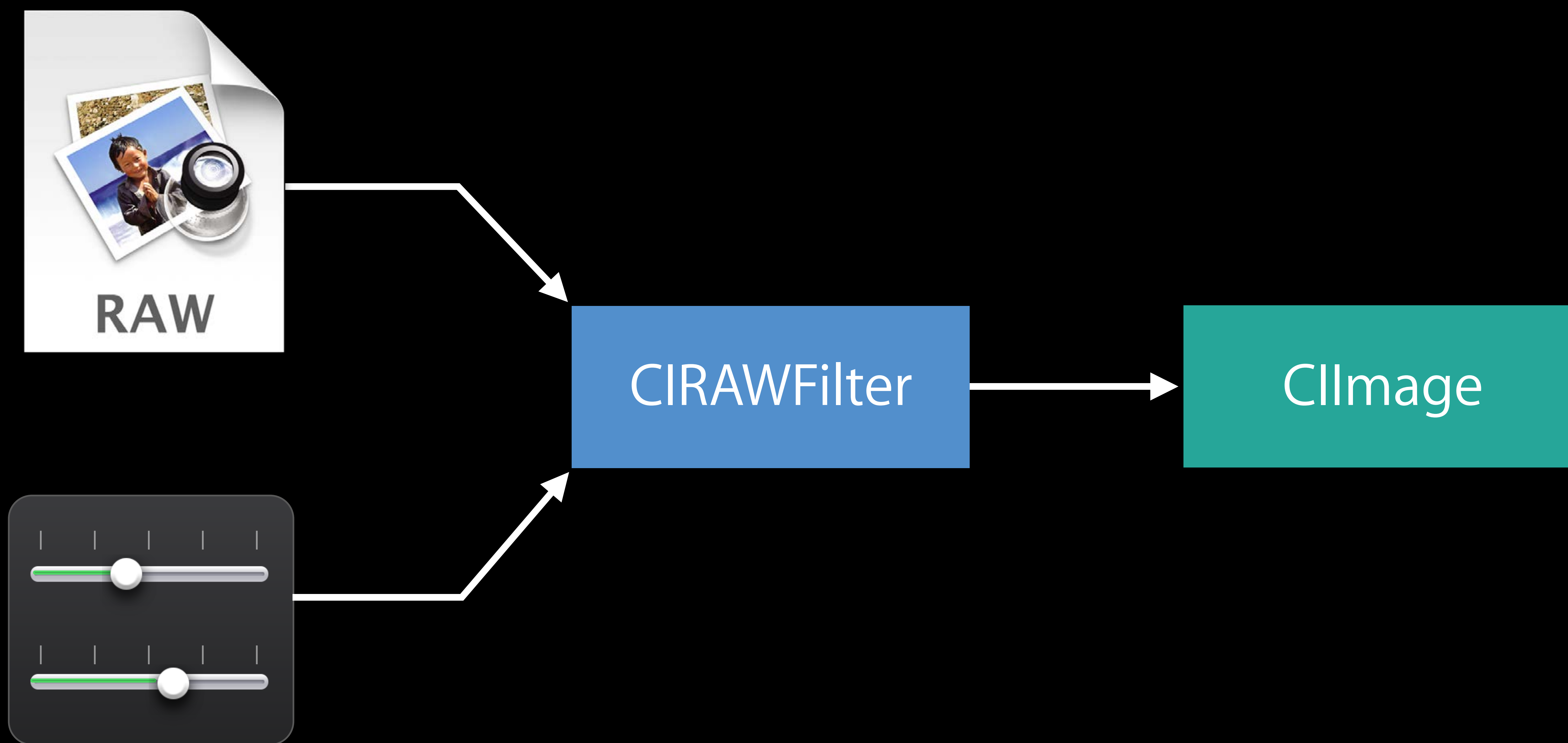
Using the CIRAWFilter API

RAW Image File

- File URL
- File data
- CVPixelBuffer

User Adjustments

- Exposure
- Temperature, tint
- Noise reduction



Adjusting RAW Images

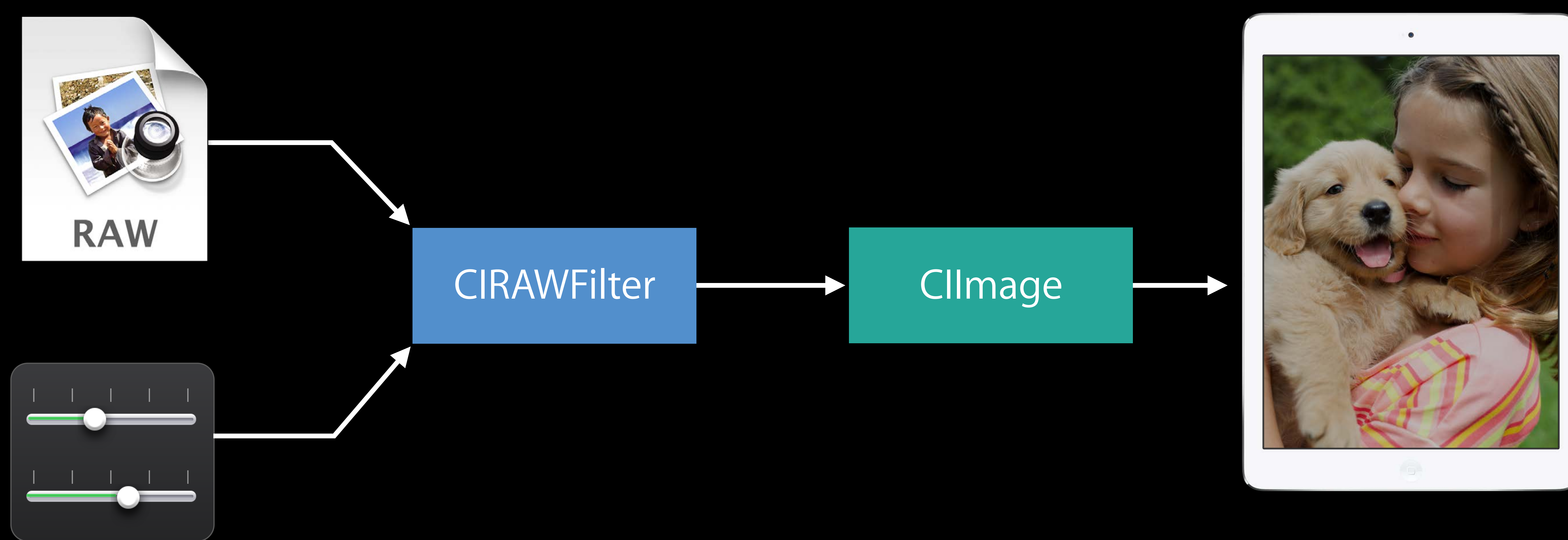
Using the CIRAWFilter API

RAW Image File

- File URL
- File data
- CVPixelBuffer

User Adjustments

- Exposure
- Temperature, tint
- Noise reduction



Adjusting RAW Images

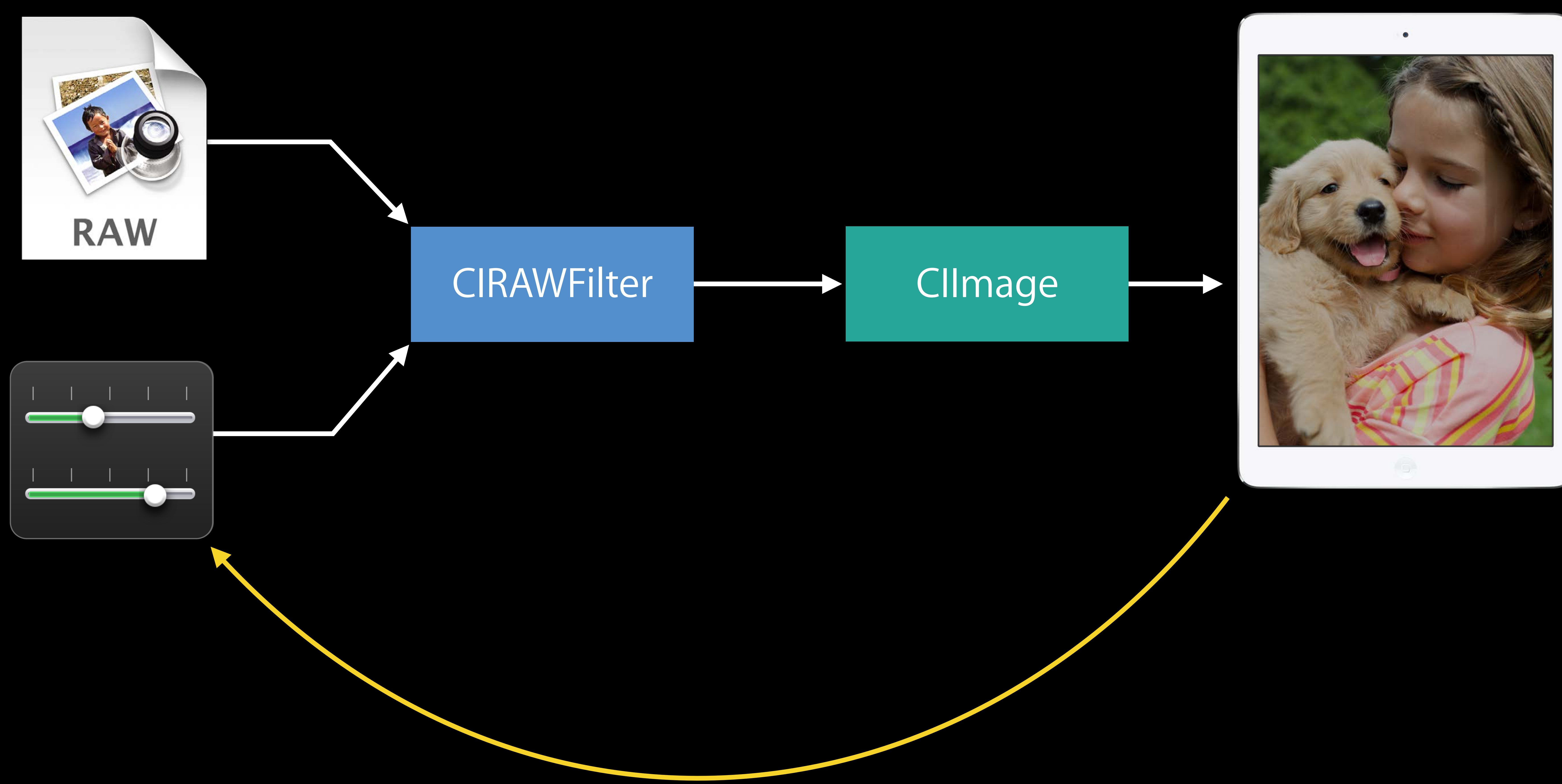
Using the CIRAWFilter API

RAW Image File

- File URL
- File data
- CVPixelBuffer

User Adjustments

- Exposure
- Temperature, tint
- Noise reduction



Adjusting RAW Images

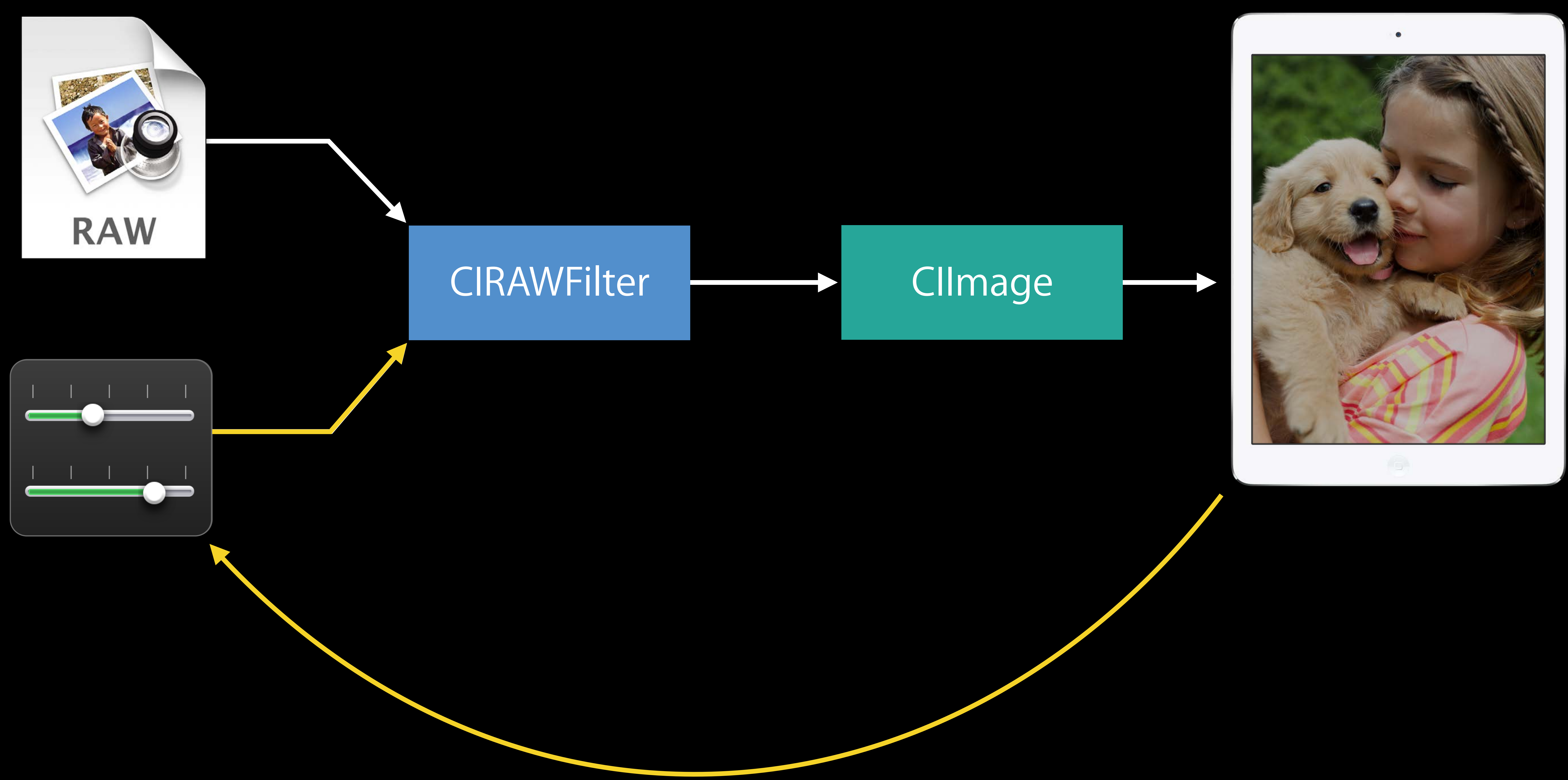
Using the CIRAWFilter API

RAW Image File

- File URL
- File data
- CVPixelBuffer

User Adjustments

- Exposure
- Temperature, tint
- Noise reduction



Adjusting RAW Images

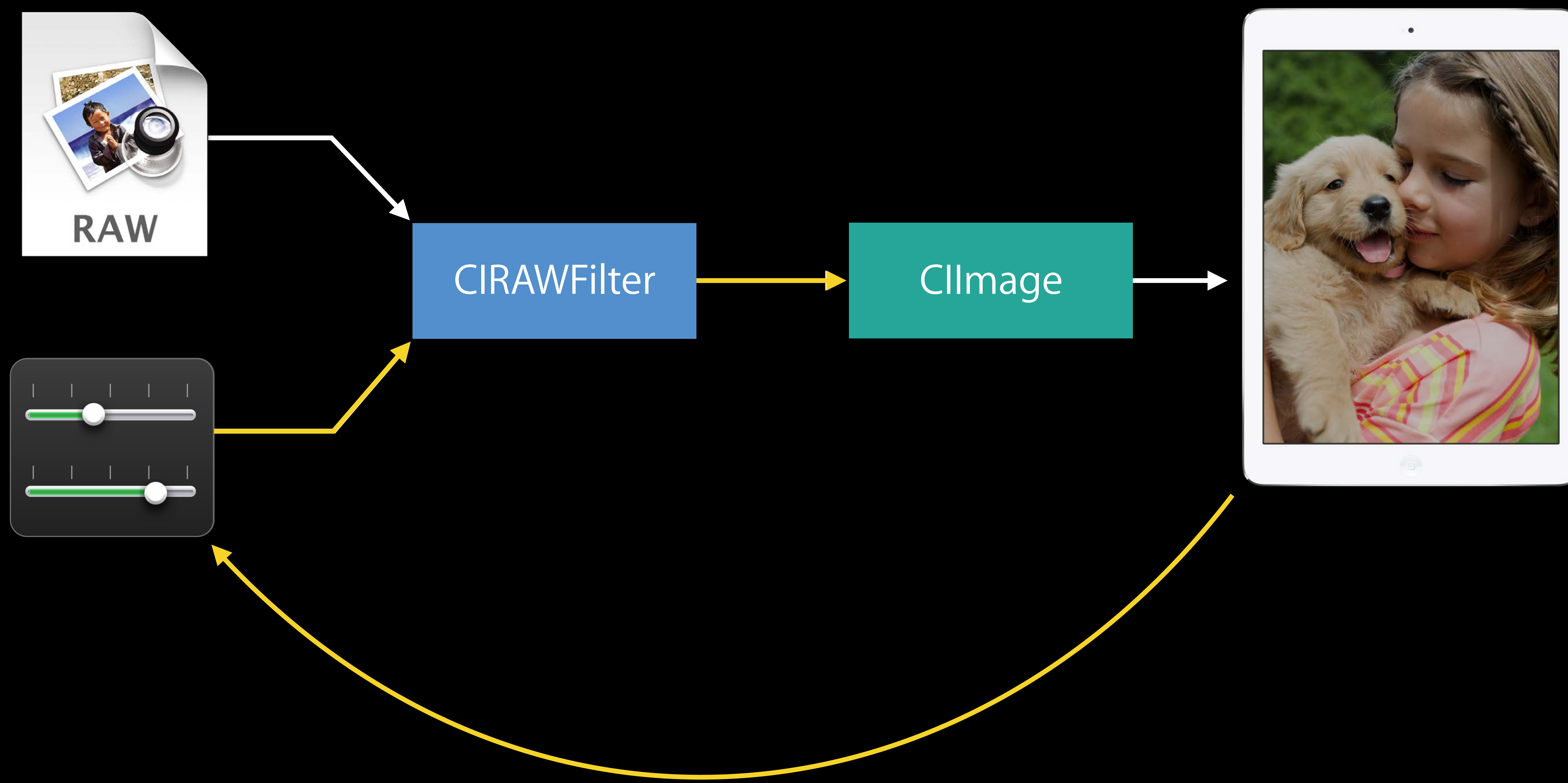
Using the CIRAWFilter API

RAW Image File

- File URL
- File data
- CVPixelBuffer

User Adjustments

- Exposure
- Temperature, tint
- Noise reduction



Adjusting RAW Images

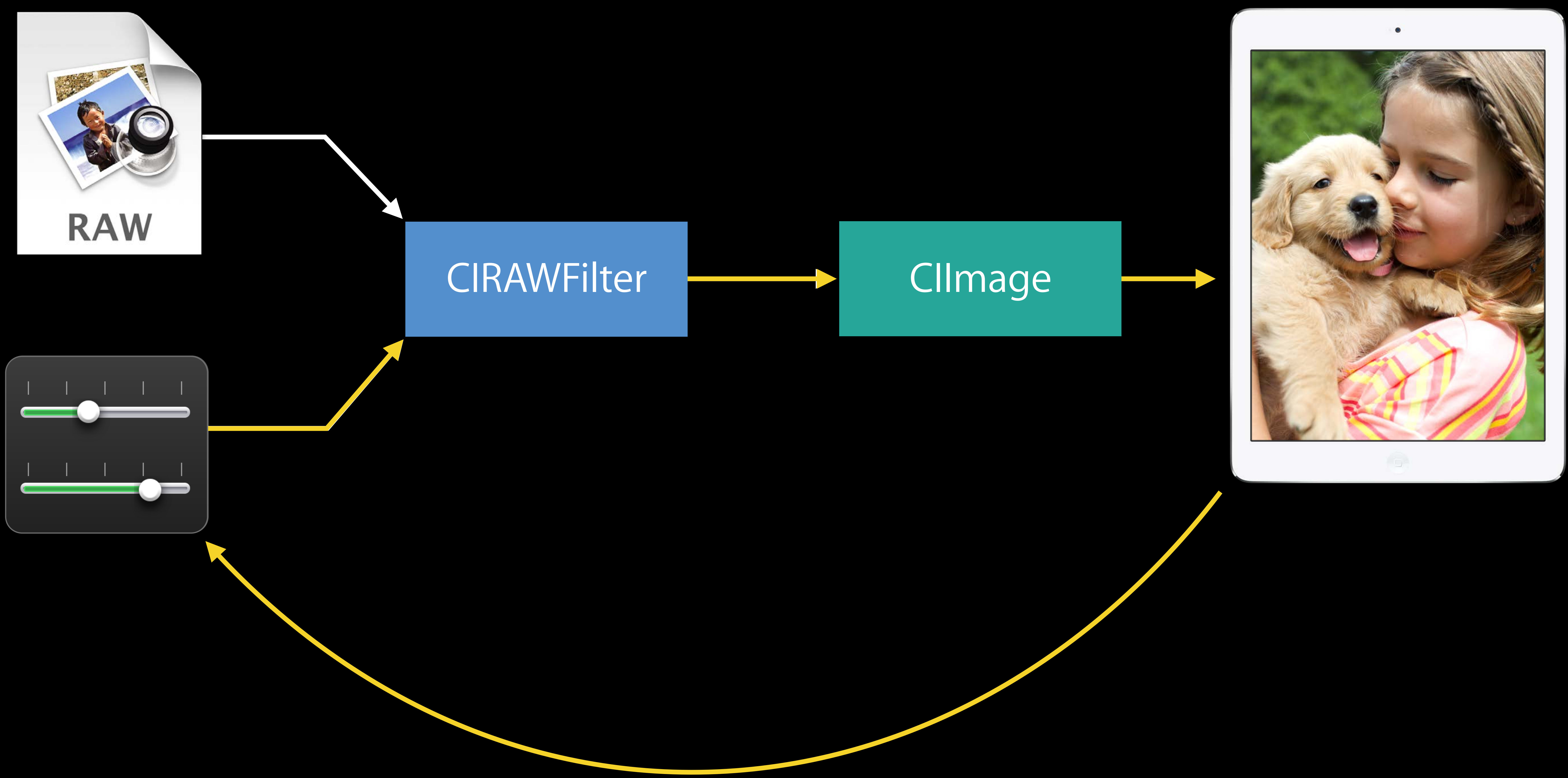
Using the CIRAWFilter API

RAW Image File

- File URL
- File data
- CVPixelBuffer

User Adjustments

- Exposure
- Temperature, tint
- Noise reduction



Adjusting RAW Images

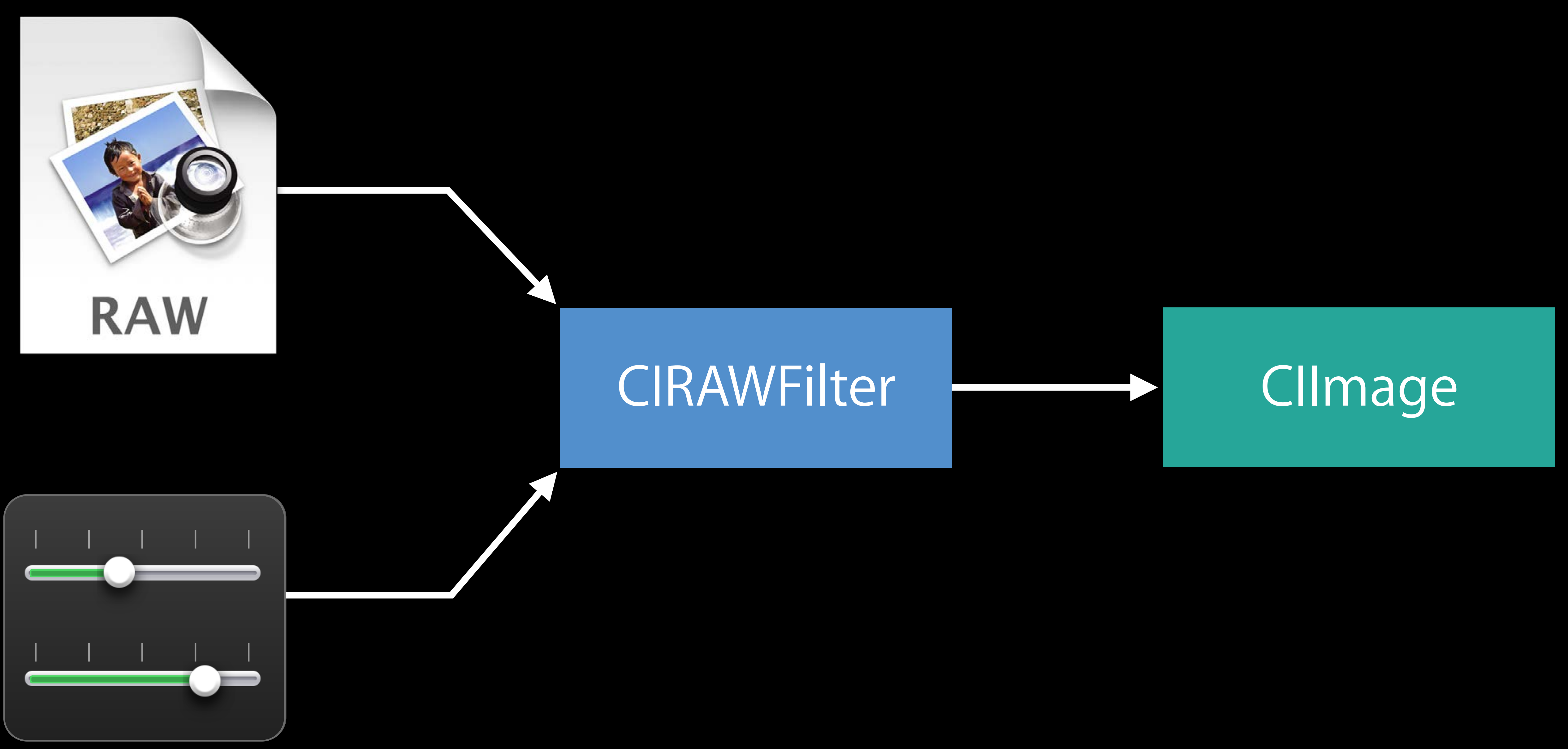
Using the CIRAWFilter API

RAW Image File

- File URL
- File data
- CVPixelBuffer

User Adjustments

- Exposure
- Temperature, tint
- Noise reduction



Adjusting RAW Images

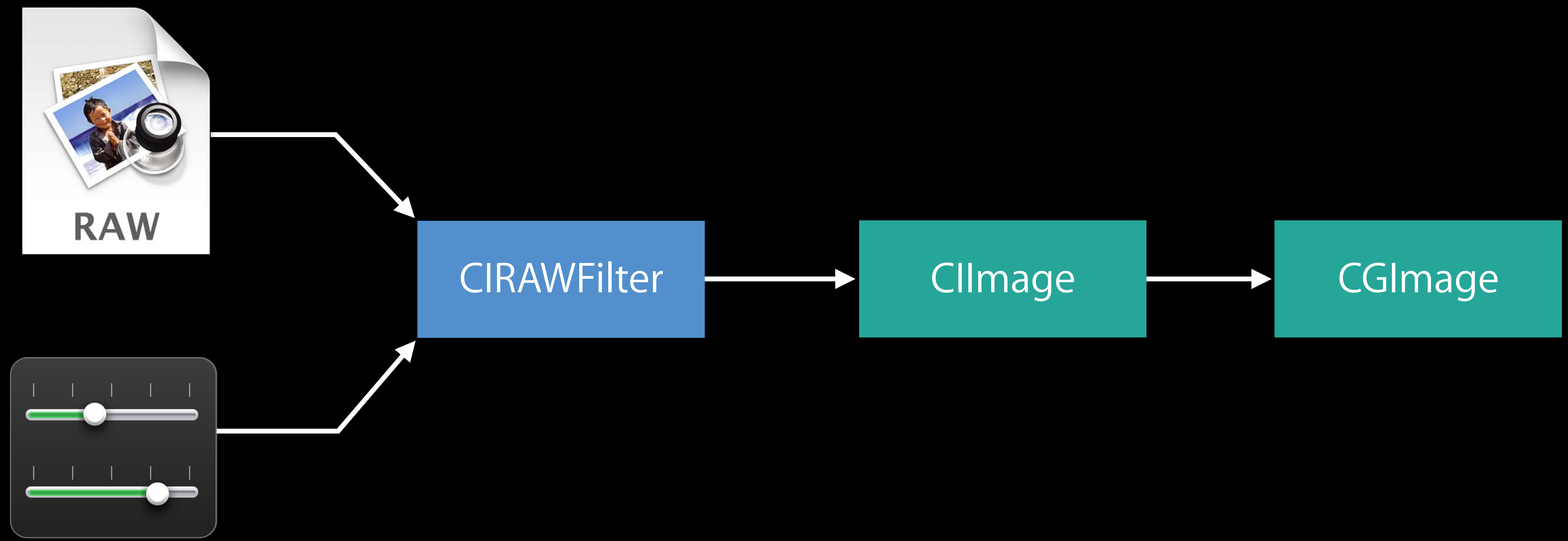
Using the CIRAWFilter API

RAW Image File

- File URL
- File data
- CVPixelBuffer

User Adjustments

- Exposure
- Temperature, tint
- Noise reduction



Adjusting RAW Images

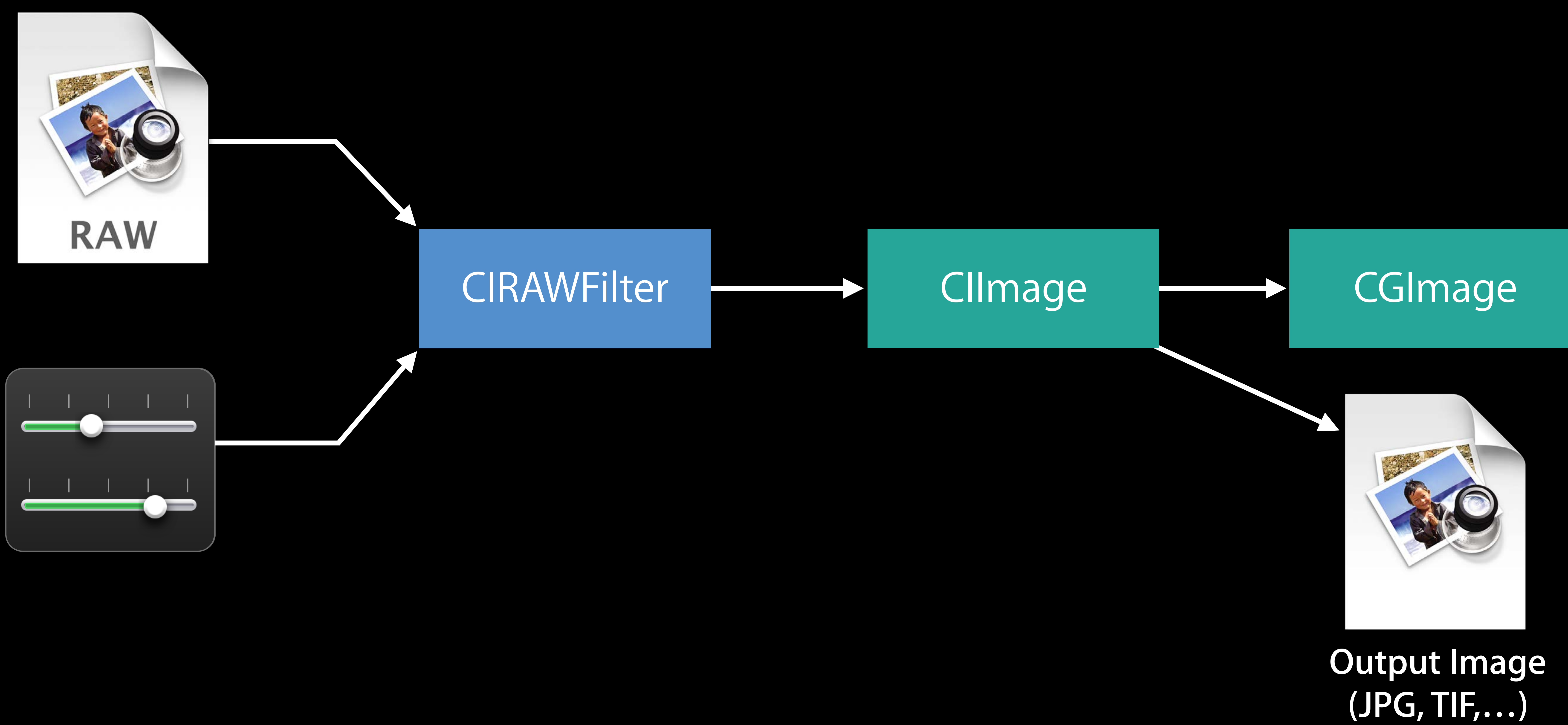
Using the CIRAWFilter API

RAW Image File

- File URL
- File data
- CVPixelBuffer

User Adjustments

- Exposure
- Temperature, tint
- Noise reduction



```swift
// Saving a RAW to a JPEG or TIFF

class myClass {

lazy var contextForSaving: CIContext = CIContext(options:
    [kCIContextCacheIntermediates : false,
     kCIContextPriorityRequestLow : true]) // Now this works on macOS too!
```

```swift
// Saving a RAW to a JPEG or TIFF

class myClass {

lazy var contextForSaving: CIContext = CIContext(options:
    [kCIContextCacheIntermediates : false,
     kCIContextPriorityRequestLow : true]) // Now this works on macOS too!
```

```swift
// Saving a RAW to a JPEG or TIFF

class myClass {

lazy var contextForSaving: CIContext = CIContext(options:
    [kCIContextCacheIntermediates : false,
     kCIContextPriorityRequestLow : true]) // Now this works on macOS too!
```

```swift
// Saving a RAW to a JPEG or TIFF

class myClass {

lazy var contextForSaving: CIContext = CIContext(options:
    [kCIContextCacheIntermediates : false,
     kCIContextPriorityRequestLow : true]) // Now this works on macOS too!
```

```swift
// Saving a RAW to a JPEG or TIFF

class myClass {

lazy var contextForSaving: CIContext = CIContext(options:
    [kCIContextCacheIntermediates : false,
     kCIContextPriorityRequestLow : true]) // Now this works on macOS too!
```

```swift
// Saving a RAW to a JPEG or TIFF

func save(from rawImage: CIImage,
          to jpegDestination: URL) throws
{
    let cs = CGColorSpace(name: CGColorSpace.displayP3)!

    try contextForSaving.writeJPEGRepresentation(
            of: rawImage,
            to: jpegDestination,
            colorSpace: cs,
            options: [kCGImageDestinationLossyCompressionQuality: 1.0])
}
```

```swift
// Saving a RAW to a JPEG or TIFF

func save(from rawImage: CIImage,
          to jpegDestination: URL) throws
{
    let cs = CGColorSpace(name: CGColorSpace.displayP3)!

    try contextForSaving.writeJPEGRepresentation(
            of: rawImage,
            to: jpegDestination,
            colorSpace: cs,
            options: [kCGImageDestinationLossyCompressionQuality: 1.0])
}
```

```swift
// Saving a RAW to a JPEG or TIFF

func save(from rawImage: CIImage,
          to jpegDestination: URL) throws
{
    let cs = CGColorSpace(name: CGColorSpace.displayP3)!

    try contextForSaving.writeJPEGRepresentation(
            of: rawImage,
            to: jpegDestination,
            colorSpace: cs,
            options: [kCGImageDestinationLossyCompressionQuality: 1.0])
}
```

```swift
// Saving a RAW to a JPEG or TIFF

func save(from rawImage: CIImage,
          to jpegDestination: URL) throws
{
    let cs = CGColorSpace(name: CGColorSpace.displayP3)!

    try contextForSaving.writeJPEGRepresentation(
            of: rawImage,
            to: jpegDestination,
            colorSpace: cs,
            options: [kCGImageDestinationLossyCompressionQuality: 1.0])
}
```

```swift
// Saving a RAW to a JPEG or TIFF

func save(from rawImage: CIImage,
          to jpegDestination: URL) throws
{
    let cs = CGColorSpace(name: CGColorSpace.displayP3)!

    try contextForSaving.writeJPEGRepresentation(
            of: rawImage,
            to: jpegDestination,
            colorSpace: cs,
            options: [kCGImageDestinationLossyCompressionQuality: 1.0])
}
```

```swift
// Share a RAW to a JPEG or TIFF
// Useful if the receiver doesn't support color management
func share(from rawImage: CIImage,
           to jpegDestination: URL) throws
{
    let cs = CGColorSpace(name: CGColorSpace.displayP3)!

    try contextForSaving.writeJPEGRepresentation(
            of: rawImage,
            to: jpegDestination,
            colorSpace: cs,
            options: [kCGImageDestinationLossyCompressionQuality: 1.0,
                      kCGImageDestinationOptimizeColorForSharing: true])
}
```

```swift
// Share a RAW to a JPEG or TIFF
// Useful if the receiver doesn't support color management
func share(from rawImage: CIImage,
           to jpegDestination: URL) throws
{
    let cs = CGColorSpace(name: CGColorSpace.displayP3)!

    try contextForSaving.writeJPEGRepresentation(
            of: rawImage,
            to: jpegDestination,
            colorSpace: cs,
            options: [kCGImageDestinationLossyCompressionQuality: 1.0,
                      kCGImageDestinationOptimizeColorForSharing: true])
}
```

```swift
// Saving a RAW to a CGImageRef

func createCGImage(from rawImage: CIImage) -> CGImage?
{
   return contextForSaving.createCGImage(
        rawImage,
        from: rawImage.extent,
        format: kCIFormatRGBA8,
        colorSpace: CGColorSpace(name: CGColorSpace.displayP3),
        deferred: true) // process the RAW when returned CGImage is drawn
}
```

```swift
// Saving a RAW to a CGImageRef

func createCGImage(from rawImage: CIImage) -> CGImage?
{
    return contextForSaving.createCGImage(
          rawImage,
          from: rawImage.extent,
          format: kCIFormatRGBA8,
          colorSpace: CGColorSpace(name: CGColorSpace.displayP3),
          deferred: true) // process the RAW when returned CGImage is drawn
}
```

```swift
// Saving a RAW to a CGImageRef

func createCGImage(from rawImage: CIImage) -> CGImage?
{
   return contextForSaving.createCGImage(
        rawImage,
        from: rawImage.extent,
        format: kCIFormatRGBAh,
        colorSpace: CGColorSpace(name: CGColorSpace.extendedLinearSRGB),
        deferred: true) // process the RAW when returned CGImage is drawn
}
```

```swift
// Saving a RAW to a CGImageRef

func createCGImage(from rawImage: CIImage) -> CGImage?
{
    return contextForSaving.createCGImage(
         rawImage,
         from: rawImage.extent,
         format: kCIFormatRGBAh,
         colorSpace: CGColorSpace(name: CGColorSpace.extendedLinearSRGB),
         deferred: true) // process the RAW when returned CGImage is drawn
}
```

```swift
// Saving a RAW to a CGImageRef

func createCGImage(from rawImage: CIImage) -> CGImage?
{
  return contextForSaving.createCGImage(
        rawImage,
        from: rawImage.extent,
        format: kCIFormatRGBAh,
        colorSpace: CGColorSpace(name: CGColorSpace.extendedLinearSRGB),
        deferred: false) // process the RAW once before this returns
}
```

```swift
// Saving a RAW to a CGImageRef

func createCGImage(from rawImage: CIImage) -> CGImage?
{
   return contextForSaving.createCGImage(
        rawImage,
        from: rawImage.extent,
        format: kCIFormatRGBAh,
        colorSpace: CGColorSpace(name: CGColorSpace.extendedLinearSRGB),
        deferred: false) // process the RAW once before this returns
}
```

Adjusting RAW Images

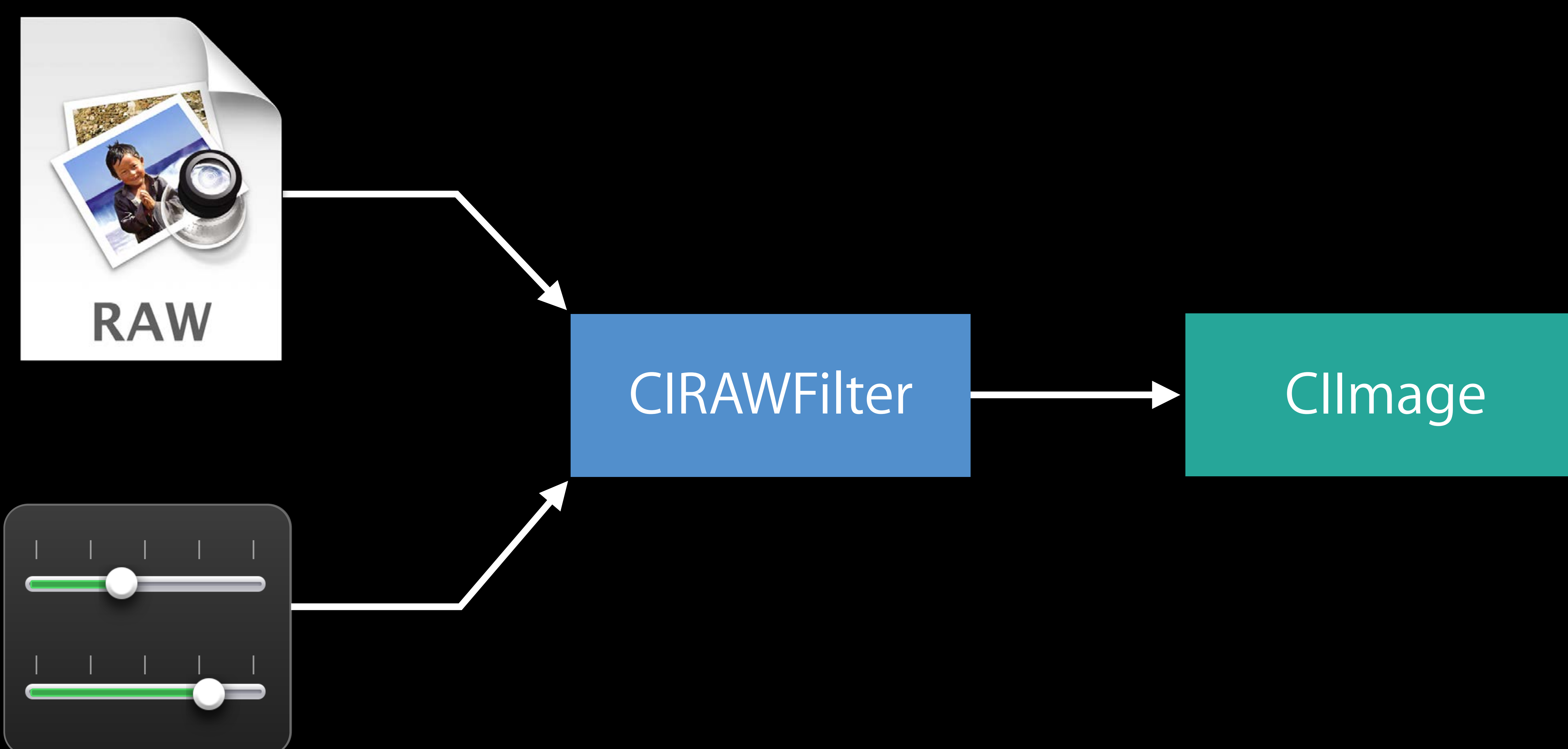
Using the CIRAWFilter API

RAW Image File

- File URL
- File data
- CVPixelBuffer

User Adjustments

- Exposure
- Temperature, tint
- Noise reduction



Adjusting RAW Images

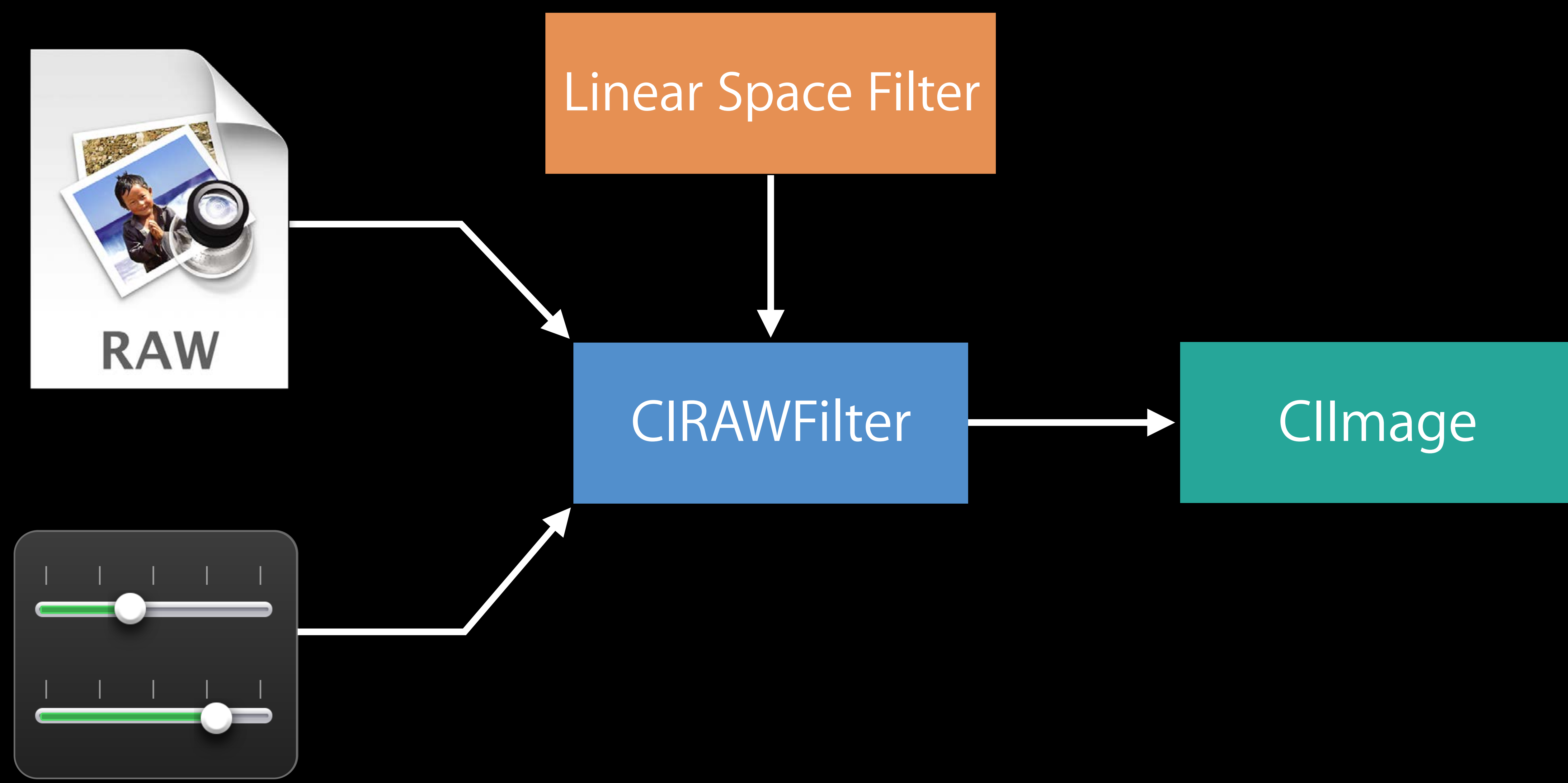
Using the CIRAWFilter API

RAW Image File

- File URL
- File data
- CVPixelBuffer

User Adjustments

- Exposure
- Temperature, tint
- Noise reduction



Adjusting RAW Images

Supporting wide gamut

Adjusting RAW Images

Supporting wide gamut

CIKernel Language uses float precision

Adjusting RAW Images

Supporting wide gamut

CIKernel Language uses float precision

- When needed, intermediate buffers use the CIContext's current working format

Adjusting RAW Images

Supporting wide gamut

CIKernel Language uses float precision

- When needed, intermediate buffers use the CIContext's current working format
- On macOS, the default working format is `kCIFormatRGBAh`

Adjusting RAW Images

Supporting wide gamut

CIKernel Language uses float precision

- When needed, intermediate buffers use the CIContext's current working format
- On macOS, the default working format is `kCIFormatRGBAh`
- On iOS/tvOS, the default working format is `kCIFormatBGRA8`

Adjusting RAW Images

Supporting wide gamut

CIKernel Language uses float precision

- When needed, intermediate buffers use the CIContext's current working format
- On macOS, the default working format is `kCIFormatRGBAh`
- On iOS/tvOS, the default working format is `kCIFormatBGRA8`
- RAW pipeline CIKernels always use `kCIFormatRGBAh` working format

Adjusting RAW Images

Supporting wide gamut

CIKernel Language uses float precision

- When needed, intermediate buffers use the CIContext's current working format
- On macOS, the default working format is `kCIFormatRGBAh`
- On iOS/tvOS, the default working format is `kCIFormatBGRA8`
- RAW pipeline CIKernels always use `kCIFormatRGBAh` working format

Create your CIContext with a `kCIContextWorkingFormat` option set to `kCIFormatRGBAh` ensure wide gamut won't be clipped.

Adjusting RAW Images

Supporting wide gamut

CIKernel Language uses float precision

- When needed, intermediate buffers use the CIContext's current working format
- On macOS, the default working format is `kCIFormatRGBAh`
- On iOS/tvOS, the default working format is `kCIFormatBGRA8`
- RAW pipeline CIKernels always use `kCIFormatRGBAh` working format

Create your CIContext with a `kCIContextWorkingFormat` option set to `kCIFormatRGBAh` ensure wide gamut won't be clipped.

Core Image supports wide gamut output color spaces

Adjusting RAW Images

Supporting wide gamut

CIKernel Language uses float precision

- When needed, intermediate buffers use the CIContext's current working format
- On macOS, the default working format is `kCIFormatRGBAh`
- On iOS/tvOS, the default working format is `kCIFormatBGRA8`
- RAW pipeline CIKernels always use `kCIFormatRGBAh` working format

Create your CIContext with a `kCIContextWorkingFormat` option set to `kCIFormatRGBAh` ensure wide gamut won't be clipped.

Core Image supports wide gamut output color spaces

- Such as Extended Linear sRGB, Adobe RGB, or Display P3

Saving RAW Images

Warning: “Objects are larger than they appear”

Saving RAW Images

Warning:“Objects are larger than they appear”

RAW files can be very large and require several intermediate buffers to render

Saving RAW Images

Warning: "Objects are larger than they appear"

RAW files can be very large and require several intermediate buffers to render

To reduce memory high water-mark use these new APIs:

Saving RAW Images

Warning: "Objects are larger than they appear"

RAW files can be very large and require several intermediate buffers to render

To reduce memory high water-mark use these new APIs:

```
CIContext(options: [kCIContextCacheIntermediates: false])
```

Saving RAW Images

Warning: "Objects are larger than they appear"

RAW files can be very large and require several intermediate buffers to render

To reduce memory high water-mark use these new APIs:

```
CIContext(options: [kCIContextCacheIntermediates: false])

context.writeJPEGRepresentationOfImage()
```

Saving RAW Images

Warning: "Objects are larger than they appear"

RAW files can be very large and require several intermediate buffers to render

To reduce memory high water-mark use these new APIs:

```
CIContext(options: [kCIContextCacheIntermediates: false])
context.writeJPEGRepresentationOfImage()
context.createCGImage(... deferred: true)
```

Saving RAW Images

Warning: "Objects are larger than they appear"

Application Type	Supports RAWs

Saving RAW Images

Warning: "Objects are larger than they appear"

Application Type	Supports RAWs
Apps on ≥2GB RAM Devices	Up to 120 Megapixels

Saving RAW Images

Warning: "Objects are larger than they appear"

Application Type	Supports RAWs
Apps on ≥2GB RAM Devices	Up to 120 Megapixels
Apps on 1GB RAM Devices	Up to 60 Megapixels

Saving RAW Images

Warning: "Objects are larger than they appear"

Application Type	Supports RAWs
Apps on ≥2GB RAM Devices	Up to 120 Megapixels
Apps on 1GB RAM Devices	Up to 60 Megapixels
Photo Editing Extensions	Up to 60 Megapixels

Editing Live Photos

Etienne Guerard Live Photo Editor-in-Chief

Editing Live Photos

Agenda

Editing Live Photos

Agenda

Introduction

Editing Live Photos

Agenda

Introduction

What Can be Edited?

Editing Live Photos

Agenda

Introduction

What Can be Edited?

Obtaining a Live Photo for Editing

Editing Live Photos

Agenda

Introduction

What Can be Edited?

Obtaining a Live Photo for Editing

Setting Up a Live Photo Editing Context

Editing Live Photos

Agenda

Introduction

What Can be Edited?

Obtaining a Live Photo for Editing

Setting Up a Live Photo Editing Context

Applying Core Image Filters

Editing Live Photos

Agenda

Introduction

What Can be Edited?

Obtaining a Live Photo for Editing

Setting Up a Live Photo Editing Context

Applying Core Image Filters

Previewing an Edited Live Photo

Editing Live Photos

Agenda

Introduction

What Can be Edited?

Obtaining a Live Photo for Editing

Setting Up a Live Photo Editing Context

Applying Core Image Filters

Previewing an Edited Live Photo

Saving to the PhotoLibrary

Editing Live Photos

Agenda

Introduction

What Can be Edited?

Obtaining a Live Photo for Editing

Setting Up a Live Photo Editing Context

Applying Core Image Filters

Previewing an Edited Live Photo

Saving to the PhotoLibrary

Demo

Live Photo

Introduction

Live Photo

Introduction

Live Photos include audio, photo, and video media

Live Photo
Introduction

Live Photos include audio, photo, and video media

Live Photos can be captured on recent devices

Live Photo

Introduction

NEW

Live Photos include audio, photo, and video media

Live Photos can be captured on recent devices

New this year:

Live Photo

Introduction

NEW

Live Photos include audio, photo, and video media

Live Photos can be captured on recent devices

New this year:

- Users can fully edit Live Photos in Photos

Live Photo

Introduction

NEW

Live Photos include audio, photo, and video media

Live Photos can be captured on recent devices

New this year:

- Users can fully edit Live Photos in Photos
- New API to capture Live Photos

Live Photo
Introduction

NEW

Live Photos include audio, photo, and video media

Live Photos can be captured on recent devices

New this year:

- Users can fully edit Live Photos in Photos
- New API to capture Live Photos
- New API to edit Live Photos!

Live Photo

What can be edited?

Live Photo

What can be edited?

Photo

Live Photo

What can be edited?

Photo

Video frames

Live Photo

What can be edited?

Photo

Video frames

Audio volume

Live Photo

What can be edited?

Photo

Video frames

Audio volume

Dimensions

Obtaining a Live Photo for Editing

Photo editing extension

```
<!-- Info.plist -->
<key>NSExtension</key>
<dict>
    <key>NSExtensionAttributes</key>
    <dict>
      <key>PHSupportedMediaTypes</key>
      <array>
        <string>LivePhoto</string>
      </array>
    </dict>
</dict>
```

Obtaining a Live Photo for Editing

Photo editing extension

```
<!-- Info.plist -->
<key>NSExtension</key>
<dict>
    <key>NSExtensionAttributes</key>
    <dict>
      <key>PHSupportedMediaTypes</key>
      <array>
        <string>LivePhoto</string>
      </array>
    </dict>
</dict>
```

Obtaining a Live Photo for Editing

Photo editing extension

```
// Called automatically by Photos when your extension starts
func startContentEditing(input: PHContentEditingInput, placeholderImage: UIImage) {
    // See if we have a Live Photo
    if input.mediaType == .image && input.mediaSubtypes.contains(.photoLive) {
        // Edit Live Photo
        // ...
    }
    else {
        // Not a Live Photo
    }
}
```

Obtaining a Live Photo for Editing

Photo editing extension

```swift
// Called automatically by Photos when your extension starts
func startContentEditing(input: PHContentEditingInput, placeholderImage: UIImage) {
    // See if we have a Live Photo
    if input.mediaType == .image && input.mediaSubtypes.contains(.photoLive) {
        // Edit Live Photo
        // ...
    }
    else {
        // Not a Live Photo
    }
}
```

Obtaining a Live Photo for Editing

Photo editing extension

```swift
// Called automatically by Photos when your extension starts
func startContentEditing(input: PHContentEditingInput, placeholderImage: UIImage) {
    // See if we have a Live Photo
    if input.mediaType == .image && input.mediaSubtypes.contains(.photoLive) {
        // Edit Live Photo
        // ...
    }
    else {
        // Not a Live Photo
    }
}
```

Obtaining a Live Photo for Editing

PhotoKit App

```swift
// Request a content editing input for a PHAsset
asset.requestContentEditingInput(options) {
    (input: PHContentEditingInput?, info: [NSObject: AnyObject]) in
    guard let input = input else { print("Error: \(info)"); return }
    // See if we have a live photo
    if input.mediaType == .image && input.mediaSubtypes.contains(.photoLive) {
        // Edit Live Photo
        // ...
    }
    else {
        // Not a Live Photo
    }
}
```

Obtaining a Live Photo for Editing

PhotoKit App

```
// Request a content editing input for a PHAsset
asset.requestContentEditingInput(options) {
    (input: PHContentEditingInput?, info: [NSObject: AnyObject]) in
    guard let input = input else { print("Error: \(info)"); return }
    // See if we have a live photo
    if input.mediaType == .image && input.mediaSubtypes.contains(.photoLive) {
        // Edit Live Photo
        // ...
    }
    else {
        // Not a Live Photo
    }
}
```

Obtaining a Live Photo for Editing

PhotoKit App

```
// Request a content editing input for a PHAsset
asset.requestContentEditingInput(options) {
    (input: PHContentEditingInput?, info: [NSObject: AnyObject]) in
    guard let input = input else { print("Error: \(info)"); return }
    // See if we have a live photo
    if input.mediaType == .image && input.mediaSubtypes.contains(.photoLive) {
        // Edit Live Photo
        // ...
    }
    else {
        // Not a Live Photo
    }
}
```

Setting Up a Live Photo Editing Context

PHLivePhotoEditingContext

Setting Up a Live Photo Editing Context

PHLivePhotoEditingContext

Info about the Live Photo

Setting Up a Live Photo Editing Context

PHLivePhotoEditingContext

Info about the Live Photo

Frame processor block

Setting Up a Live Photo Editing Context

PHLivePhotoEditingContext

- Info about the Live Photo
- Frame processor block
- Audio volume

Setting Up a Live Photo Editing Context

PHLivePhotoEditingContext

Info about the Live Photo

Frame processor block

Audio volume

Prepare Live Photo for playback

Setting Up a Live Photo Editing Context

PHLivePhotoEditingContext

Info about the Live Photo

Frame processor block

Audio volume

Prepare Live Photo for playback

Process Live Photo for saving

Setting Up a Live Photo Editing Context

PHLivePhotoEditingContext

Info about the Live Photo

Frame processor block

Audio volume

Prepare Live Photo for playback

Process Live Photo for saving

```
// Setup Live Photo editing context
self.context = PHLivePhotoEditingContext(livePhotoEditingInput: input)
```

Working with the Frame Processor

PHLivePhotoFrame

Working with the Frame Processor

PHLivePhotoFrame

Input image

Working with the Frame Processor

PHLivePhotoFrame

Input image

Frame type

Working with the Frame Processor

PHLivePhotoFrame

Input image

Frame type

Frame time

Working with the Frame Processor

PHLivePhotoFrame

Input image

Frame type

Frame time

Render scale

Working with the Frame Processor

PHLivePhotoFrame

Input image

Frame type

Frame time

Render scale

```
self.livePhotoEditingContext.frameProcessor = {
    (frame: PHLivePhotoFrame, error: NSErrorPointer) -> CIImage? in
    // Your adjustments go here...
    return frame.image
}
```

Working with the Frame Processor

PHLivePhotoFrame

Input image

Frame type

Frame time

Render scale

```
self.livePhotoEditingContext.frameProcessor = {
    (frame: PHLivePhotoFrame, error: NSErrorPointer) -> CIImage? in
    // Your adjustments go here...
    return frame.image
}
```

Working with the Frame Processor

PHLivePhotoFrame

Input image

Frame type

Frame time

Render scale

```
self.livePhotoEditingContext.frameProcessor = {
    (frame: PHLivePhotoFrame, error: NSErrorPointer) -> CIImage? in
    // Your adjustments go here...
    return frame.image
}
```

Working with the Frame Processor

PHLivePhotoFrame

Input image

Frame type

Frame time

Render scale

```
self.livePhotoEditingContext.frameProcessor = {
    (frame: PHLivePhotoFrame, error: NSErrorPointer) -> CIImage? in
    // Your adjustments go here...
    return frame.image
}
```


Cancel
Done

```swift
// Applying a static adjustment

self.livePhotoEditingContext.frameProcessor = {
   (frame: PHLivePhotoFrame, error: NSErrorPointer) -> CIImage? in
   var image = frame.image
   // Crop to square
   let extent = image.extent
   let size = min(extent.width, extent.height)
   let rect = CGRect(x: (extent.width - size) / 2, y: (extent.height - size) / 2,
      width: size, height: size)
   image = image.cropping(to: rect)
   return image
}
```

```swift
// Applying a static adjustment

self.livePhotoEditingContext.frameProcessor = {
   (frame: PHLivePhotoFrame, error: NSErrorPointer) -> CIImage? in
   var image = frame.image
   // Crop to square
   let extent = image.extent
   let size = min(extent.width, extent.height)
   let rect = CGRect(x: (extent.width - size) / 2, y: (extent.height - size) / 2,
      width: size, height: size)
   image = image.cropping(to: rect)
   return image
}
```

```swift
// Applying a static adjustment

self.livePhotoEditingContext.frameProcessor = {
   (frame: PHLivePhotoFrame, error: NSErrorPointer) -> CIImage? in
   var image = frame.image
   // Crop to square
   let extent = image.extent
   let size = min(extent.width, extent.height)
   let rect = CGRect(x: (extent.width - size) / 2, y: (extent.height - size) / 2,
      width: size, height: size)
   image = image.cropping(to: rect)
   return image
}
```

```swift
// Applying a static adjustment

self.livePhotoEditingContext.frameProcessor = {
   (frame: PHLivePhotoFrame, error: NSErrorPointer) -> CIImage? in
   var image = frame.image
   // Crop to square
   let extent = image.extent
   let size = min(extent.width, extent.height)
   let rect = CGRect(x: (extent.width - size) / 2, y: (extent.height - size) / 2,
      width: size, height: size)
   image = image.cropping(to: rect)
   return image
}
```


Cancel
LivePhotoEditor
Done

```swift
// Applying a time-based adjustment

let tP = CMTimeGetSeconds(self.livePhotoEditingContext.photoTime)
let duration = CMTimeGetSeconds(self.livePhotoEditingContext.duration)
self.livePhotoEditingContext.frameProcessor = {
   (frame: PHLivePhotoFrame, error: NSErrorPointer) -> CIImage? in
   var image = frame.image
   let tF = CMTimeGetSeconds(frame.time)
   // Simple linear ramp function from (0, tP, duration) to (-1, 0, +1)
   let dt = (tF < tP) ? CGFloat((tF - tP) / tP) : CGFloat((tF - tP) / (duration - tP))
  // Animate crop rect
   image = image.cropping(to: rect.offsetBy(dx: dt * rect.minX, dy: dt * rect.minY))
   return image
}
```

```swift
// Applying a time-based adjustment

let tP = CMTimeGetSeconds(self.livePhotoEditingContext.photoTime)
let duration = CMTimeGetSeconds(self.livePhotoEditingContext.duration)
self.livePhotoEditingContext.frameProcessor = {
   (frame: PHLivePhotoFrame, error: NSErrorPointer) -> CIImage? in
   var image = frame.image
   let tF = CMTimeGetSeconds(frame.time)
   // Simple linear ramp function from (0, tP, duration) to (-1, 0, +1)
   let dt = (tF < tP) ? CGFloat((tF - tP) / tP) : CGFloat((tF - tP) / (duration - tP))
  // Animate crop rect
   image = image.cropping(to: rect.offsetBy(dx: dt * rect.minX, dy: dt * rect.minY))
   return image
}
```

```swift
// Applying a time-based adjustment

let tP = CMTimeGetSeconds(self.livePhotoEditingContext.photoTime)
let duration = CMTimeGetSeconds(self.livePhotoEditingContext.duration)
self.livePhotoEditingContext.frameProcessor = {
   (frame: PHLivePhotoFrame, error: NSErrorPointer) -> CIImage? in
   var image = frame.image
   let tF = CMTimeGetSeconds(frame.time)
   // Simple linear ramp function from (0, tP, duration) to (-1, 0, +1)
   let dt = (tF < tP) ? CGFloat((tF - tP) / tP) : CGFloat((tF - tP) / (duration - tP))
  // Animate crop rect
   image = image.cropping(to: rect.offsetBy(dx: dt * rect.minX, dy: dt * rect.minY))
   return image
}
```

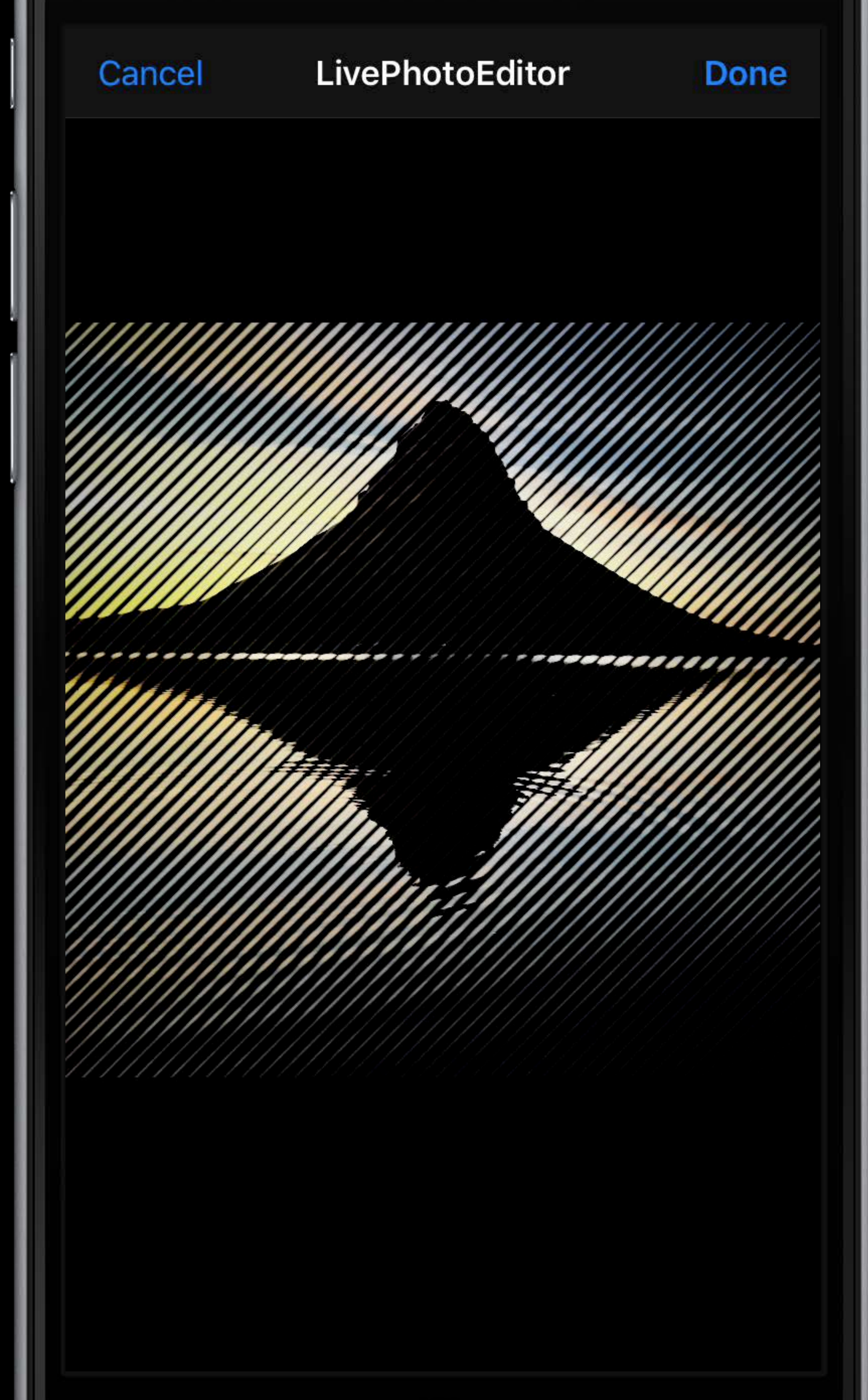
```swift
// Applying a time-based adjustment

let tP = CMTimeGetSeconds(self.livePhotoEditingContext.photoTime)
let duration = CMTimeGetSeconds(self.livePhotoEditingContext.duration)
self.livePhotoEditingContext.frameProcessor = {
   (frame: PHLivePhotoFrame, error: NSErrorPointer) -> CIImage? in
   var image = frame.image
   let tF = CMTimeGetSeconds(frame.time)
   // Simple linear ramp function from (0, tP, duration) to (-1, 0, +1)
   let dt = (tF < tP) ? CGFloat((tF - tP) / tP) : CGFloat((tF - tP) / (duration - tP))
  // Animate crop rect
   image = image.cropping(to: rect.offsetBy(dx: dt * rect.minX, dy: dt * rect.minY))
   return image
}
```

```swift
// Applying a time-based adjustment

let tP = CMTimeGetSeconds(self.livePhotoEditingContext.photoTime)
let duration = CMTimeGetSeconds(self.livePhotoEditingContext.duration)
self.livePhotoEditingContext.frameProcessor = {
   (frame: PHLivePhotoFrame, error: NSErrorPointer) -> CIImage? in
   var image = frame.image
   let tF = CMTimeGetSeconds(frame.time)
   // Simple linear ramp function from (0, tP, duration) to (-1, 0, +1)
   let dt = (tF < tP) ? CGFloat((tF - tP) / tP) : CGFloat((tF - tP) / (duration - tP))
  // Animate crop rect
   image = image.cropping(to: rect.offsetBy(dx: dt * rect.minX, dy: dt * rect.minY))
   return image
}
```


Cancel
LivePhotoEditor
Done


Cancel
LivePhotoEditor
Done

```swift
// Applying a resolution-dependent adjustment

livePhotoEditingContext.frameProcessor = {
   (frame: PHLivePhotoFrame, error: NSErrorPointer) -> CIImage? in
   var image = frame.image
   // Apply screen effect
   let scale = frame.renderScale
   image = image.applyingFilter("CILineScreen", withInputParameters:
      [ "inputAngle" : 3 * Double.pi / 4,
        "inputWidth" : 50 * scale,
        "inputCenter" : CIVector(x: image.extent.midX, y: image.extent.midY)
      ])
   return image
}
```

```swift
// Applying a resolution-dependent adjustment

livePhotoEditingContext.frameProcessor = {
    (frame: PHLivePhotoFrame, error: NSErrorPointer) -> CIImage? in
    var image = frame.image
    // Apply screen effect
    let scale = frame.renderScale
    image = image.applyingFilter("CILineScreen", withInputParameters:
        [ "inputAngle" : 3 * Double.pi / 4,
          "inputWidth" : 50 * scale,
          "inputCenter" : CIVector(x: image.extent.midX, y: image.extent.midY)
        ])
    return image
}
```

```
// Applying a resolution-dependent adjustment

livePhotoEditingContext.frameProcessor = {
   (frame: PHLivePhotoFrame, error: NSErrorPointer) -> CIImage? in
   var image = frame.image
   // Apply screen effect
   let scale = frame.renderScale
   image = image.applyingFilter("CILineScreen", withInputParameters:
      [ "inputAngle" : 3 * Double.pi / 4,
        "inputWidth" : 50 * scale,
        "inputCenter" : CIVector(x: image.extent.midX, y: image.extent.midY)
      ])
   return image
}
```

```swift
// Applying a resolution-dependent adjustment

livePhotoEditingContext.frameProcessor = {
   (frame: PHLivePhotoFrame, error: NSErrorPointer) -> CIImage? in
   var image = frame.image
   // Apply screen effect
   let scale = frame.renderScale
   image = image.applyingFilter("CILineScreen", withInputParameters:
      [ "inputAngle" : 3 * Double.pi / 4,
        "inputWidth" : 50 * scale,
        "inputCenter" : CIVector(x: image.extent.midX, y: image.extent.midY)
      ])
   return image
}
```


Cancel
LivePhotoEditor
Done
WWDC16

```swift
// Applying an adjustment to the photo only

livePhotoEditingContext.frameProcessor = {
   (frame: PHLivePhotoFrame, error: NSErrorPointer) -> CIImage? in
   var image = frame.image
   // Add watermark to the photo only
  if frame.type == .photo {
     // Composite logo
      image = logo.applyingFilter("CILinearDodgeBlendMode",
         withInputParameters: ["inputBackgroundImage" : image])
   }
   return image
}
```

```
// Applying an adjustment to the photo only

livePhotoEditingContext.frameProcessor = {
   (frame: PHLivePhotoFrame, error: NSErrorPointer) -> CIImage? in
   var image = frame.image
   // Add watermark to the photo only
   if frame.type == .photo {
      // Composite logo
      image = logo.applyingFilter("CILinearDodgeBlendMode",
         withInputParameters: ["inputBackgroundImage" : image])
   }
   return image
}
```

Previewing a Live Photo

PHLivePhotoView

```
// Prepare Live Photo for playback
self.livePhotoEditingContext.prepareLivePhotoForPlayback(withTargetSize: targetSize,
   options: nil) {
   (livePhoto: PHLivePhoto?, error: NSError?) in
   guard let livePhoto = livePhoto else { print("Prepare error: \(error)"); return }
   // Update live photo view
   self.livePhotoView.livePhoto = livePhoto
}
```

Previewing a Live Photo

PHLivePhotoView

```
// Prepare Live Photo for playback
self.livePhotoEditingContext.prepareLivePhotoForPlayback(withTargetSize: targetSize,
    options: nil) {
    (livePhoto: PHLivePhoto?, error: NSError?) in
    guard let livePhoto = livePhoto else { print("Prepare error: \(error)"); return }
    // Update live photo view
    self.livePhotoView.livePhoto = livePhoto
}
```

Previewing a Live Photo

PHLivePhotoView

```
// Prepare Live Photo for playback
self.livePhotoEditingContext.prepareLivePhotoForPlayback(withTargetSize: targetSize,
    options: nil) {
    (livePhoto: PHLivePhoto?, error: NSError?) in
    guard let livePhoto = livePhoto else { print("Prepare error: \(error)"); return }
    // Update live photo view
    self.livePhotoView.livePhoto = livePhoto
}
```

Previewing a Live Photo

PHLivePhotoView

```
// Prepare Live Photo for playback
self.livePhotoEditingContext.prepareLivePhotoForPlayback(withTargetSize: targetSize,
   options: nil) {
   (livePhoto: PHLivePhoto?, error: NSError?) in
   guard let livePhoto = livePhoto else { print("Prepare error: \(error)"); return }
   // Update live photo view
   self.livePhotoView.livePhoto = livePhoto
}
```

Saving to the Photo Library

Photo editing extension

```swift
// Called automatically by Photos to save the edits
func finishContentEditing(completionHandler: (PHContentEditingOutput?) -> Void) {
    let output = PHContentEditingOutput(contentEditingInput: self.contentEditingInput)
    self.livePhotoEditingContext.saveLivePhoto(to: output, options: nil) {
        (success: Bool, error: NSError?) in
        if success {
            output.adjustmentData = PHAdjustmentData(/* Your adjustment data */)
            completionHandler(output)
        }
    }
}
```

Saving to the Photo Library

Photo editing extension

```
// Called automatically by Photos to save the edits
func finishContentEditing(completionHandler: (PHContentEditingOutput?) -> Void) {
    let output = PHContentEditingOutput(contentEditingInput: self.contentEditingInput)
    self.livePhotoEditingContext.saveLivePhoto(to: output, options: nil) {
        (success: Bool, error: NSError?) in
        if success {
            output.adjustmentData = PHAdjustmentData(/* Your adjustment data */)
            completionHandler(output)
        }
    }
}
```

Saving to the Photo Library

Photo editing extension

```
// Called automatically by Photos to save the edits
func finishContentEditing(completionHandler: (PHContentEditingOutput?) -> Void) {
    let output = PHContentEditingOutput(contentEditingInput: self.contentEditingInput)
    self.livePhotoEditingContext.saveLivePhoto(to: output, options: nil) {
        (success: Bool, error: NSError?) in
        if success {
            output.adjustmentData = PHAdjustmentData(/* Your adjustment data */)
            completionHandler(output)
        }
    }
}
```

Saving to the Photo Library

Photo editing extension

```
// Called automatically by Photos to save the edits
func finishContentEditing(completionHandler: (PHContentEditingOutput?) -> Void) {
    let output = PHContentEditingOutput(contentEditingInput: self.contentEditingInput)
    self.livePhotoEditingContext.saveLivePhoto(to: output, options: nil) {
        (success: Bool, error: NSError?) in
        if success {
            output.adjustmentData = PHAdjustmentData(/* Your adjustment data */)
            completionHandler(output)
        }
    }
}
```

Saving to the Photo Library

Photo editing extension

```
// Called automatically by Photos to save the edits
func finishContentEditing(completionHandler: (PHContentEditingOutput?) -> Void) {
    let output = PHContentEditingOutput(contentEditingInput: self.contentEditingInput)
    self.livePhotoEditingContext.saveLivePhoto(to: output, options: nil) {
        (success: Bool, error: NSError?) in
        if success {
            output.adjustmentData = PHAdjustmentData(/* Your adjustment data */)
            completionHandler(output)
        }
    }
}
```

Saving to the Photo Library

Photo editing extension

```
// Called automatically by Photos to save the edits
func finishContentEditing(completionHandler: (PHContentEditingOutput?) -> Void) {
   let output = PHContentEditingOutput(contentEditingInput: self.contentEditingInput)
   self.livePhotoEditingContext.saveLivePhoto(to: output, options: nil) {
      (success: Bool, error: NSError?) in
      if success {
         output.adjustmentData = PHAdjustmentData(/* Your adjustment data */)
         completionHandler(output)
      }
   }
}
```

Saving to the Photo Library

PhotoKit App

```
let output = PHContentEditingOutput(contentEditingInput: self.contentEditingInput)
self.livePhotoEditingContext.saveLivePhoto(to: output, options: nil) {
   (success: Bool, error: NSError?) in
   if success {
      output.adjustmentData = PHAdjustmentData(/* Your adjustment data */)
      PHPhotoLibrary.shared().performChanges({
         PHAssetChangeRequest(for: asset).contentEditingOutput = output
      }) { (success: Bool, error: NSError?) in
         // Completion handler
      }
   }
}
```

Saving to the Photo Library

PhotoKit App

```
let output = PHContentEditingOutput(contentEditingInput: self.contentEditingInput)
self.livePhotoEditingContext.saveLivePhoto(to: output, options: nil) {
   (success: Bool, error: NSError?) in
   if success {
      output.adjustmentData = PHAdjustmentData(/* Your adjustment data */)
      PHPhotoLibrary.shared().performChanges({
         PHAssetChangeRequest(for: asset).contentEditingOutput = output
      }) { (success: Bool, error: NSError?) in
         // Completion handler
      }
   }
}
```

Demo

Live Photo editing extension

Editing Live Photos

Summary

Editing Live Photos

Summary

What you’ve learned so far

Editing Live Photos

Summary

What you've learned so far

- How to use the Live Photo editing context and the frame processor

Editing Live Photos

Summary

What you've learned so far

- How to use the Live Photo editing context and the frame processor
- How to preview a Live Photo using a Live Photo view

Editing Live Photos

Summary

What you've learned so far

- How to use the Live Photo editing context and the frame processor
- How to preview a Live Photo using a Live Photo view
- How to save a Live Photo back to the Photo Library

Editing Live Photos

Summary

What you've learned so far

- How to use the Live Photo editing context and the frame processor
- How to preview a Live Photo using a Live Photo view
- How to save a Live Photo back to the Photo Library

Remember

Editing Live Photos

Summary

What you've learned so far

- How to use the Live Photo editing context and the frame processor
- How to preview a Live Photo using a Live Photo view
- How to save a Live Photo back to the Photo Library

Remember

- Don't forget to opt-in to Live Photo Editing in your extension's Info.plist

Editing Live Photos

Summary

What you've learned so far

- How to use the Live Photo editing context and the frame processor
- How to preview a Live Photo using a Live Photo view
- How to save a Live Photo back to the Photo Library

Remember

- Don't forget to opt-in to Live Photo Editing in your extension's Info.plist
- Make sure to save your adjustment data to the Photo Library

Editing Live Photos

Summary

What you’ve learned so far

- How to use the Live Photo editing context and the frame processor
- How to preview a Live Photo using a Live Photo view
- How to save a Live Photo back to the Photo Library

Remember

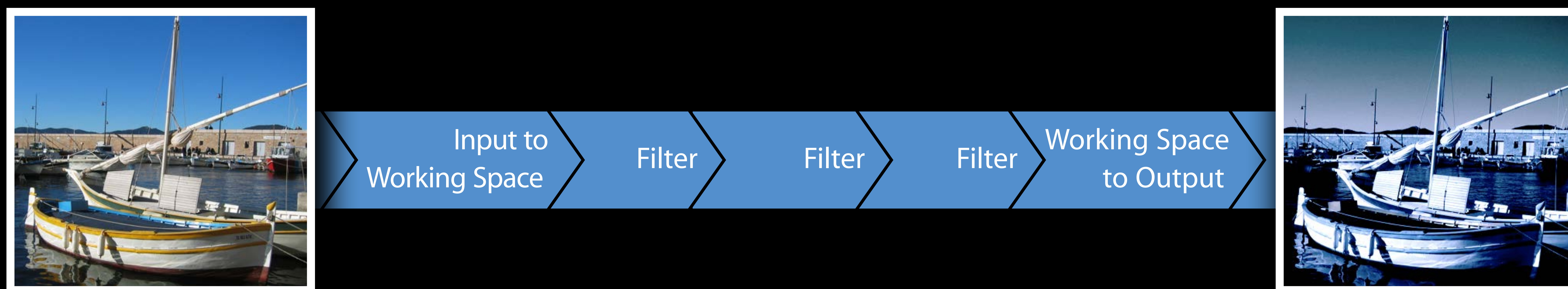
- Don’t forget to opt-in to Live Photo Editing in your extension’s Info.plist
- Make sure to save your adjustment data to the Photo Library
- Live Photo Editing support should be easy to add to your existing app/extension

Extending Core Image Using CIImageProcessor

Alexandre Naaman Lord of Pixelland

Using CIImageProcessor

You can do lots with built-in CIFilters and custom CIKernels

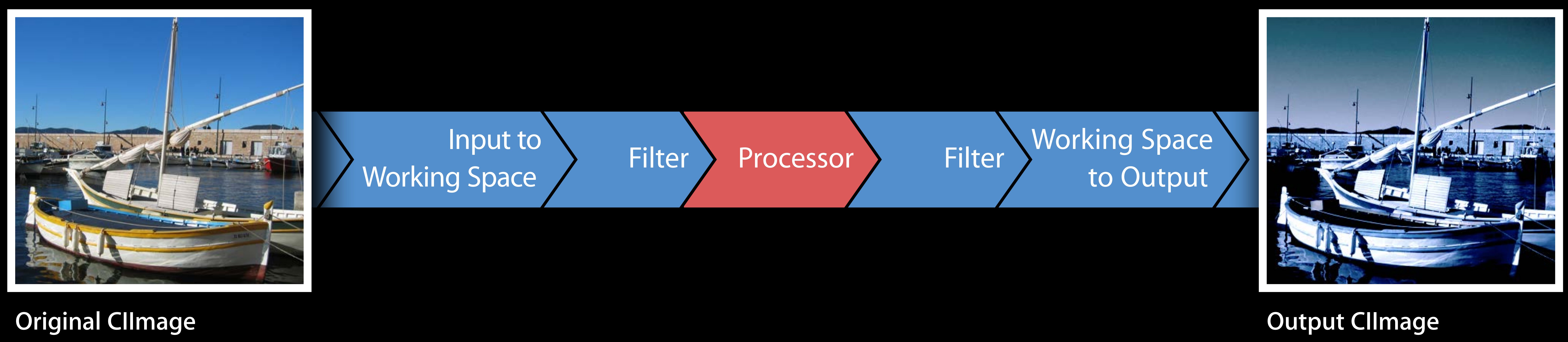


Original CIImage

Output CIImage

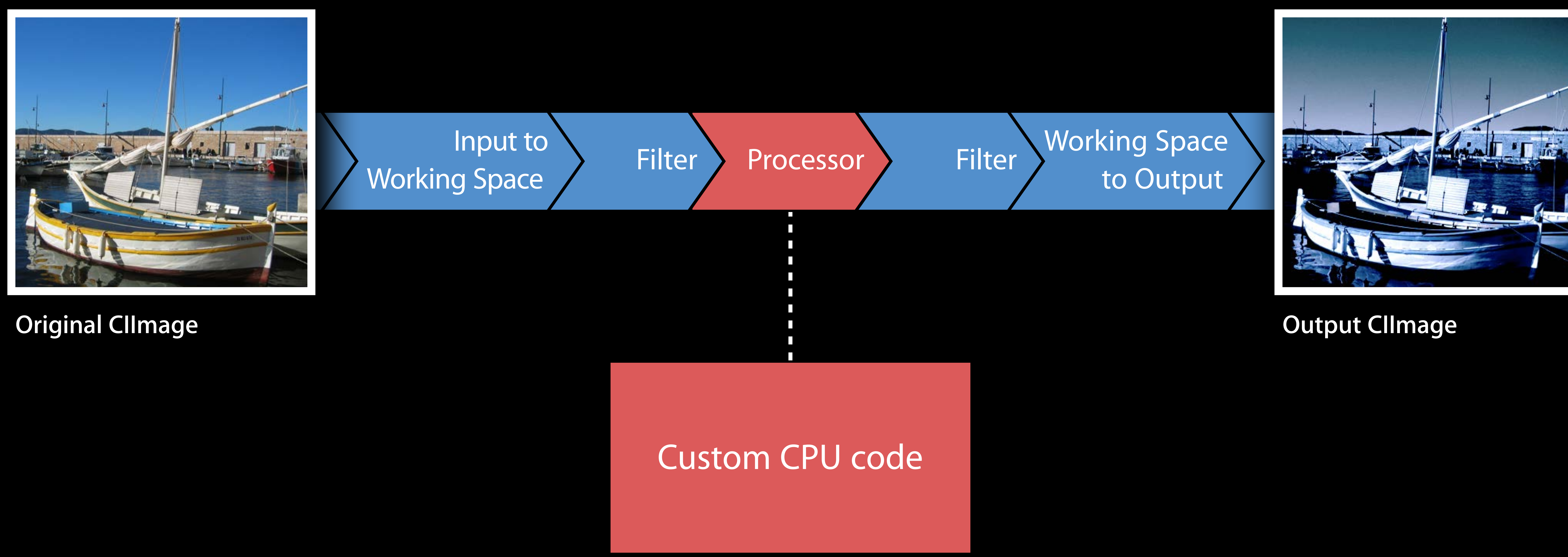
Using CIImageProcessor

Now part of the graph can use something different



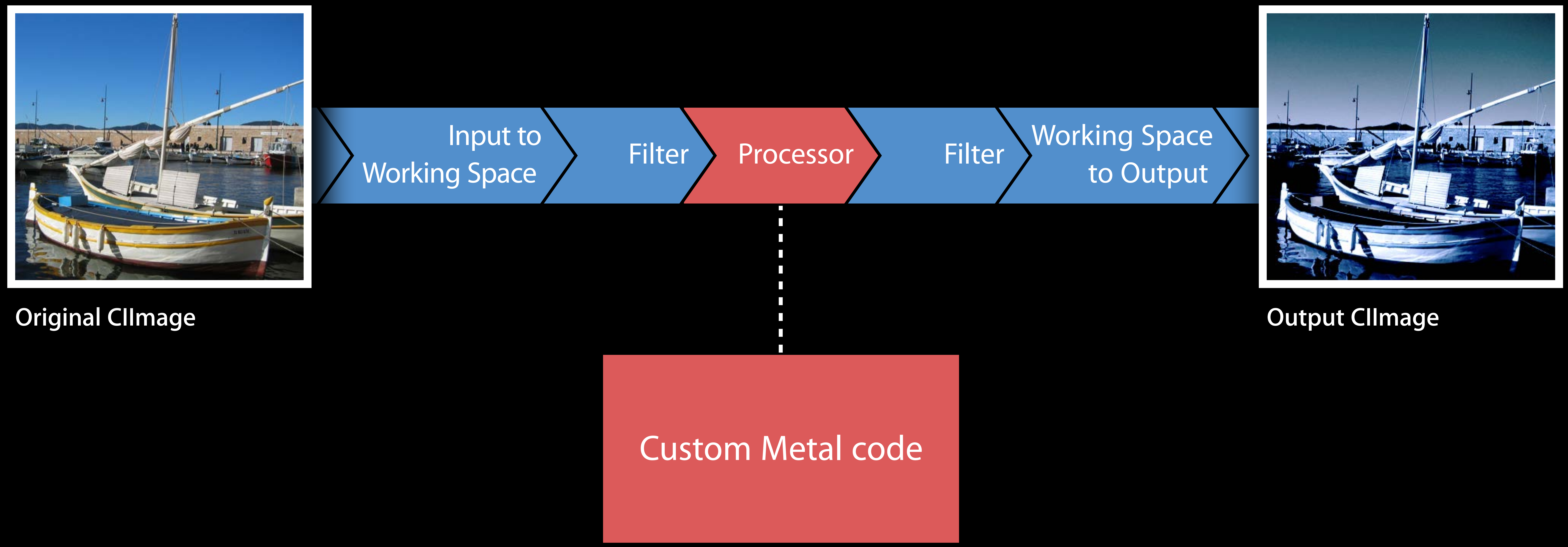
Using CIImageProcessor

Now part of the graph can use something different



Using CIImageProcessor

Now part of the graph can use something different



```swift
// Applying a CIKernel in a CIFilter subclass

// Only create the kernel once
static let yourKernel = CIKernel(string:"kernel vec4 your_code_here …")!

override var outputImage: CIImage!
{
    return yourKernel.apply(withExtent: calcExtent(),
                            roiCallback: { (index, rect) -> CGRect in
                                           return calcROI(rect) },
                            arguments: [inputImage!, inputArgument!])
}
```

```swift
// Applying a CIKernel in a CIFilter subclass

// Only create the kernel once
static let yourKernel = CIKernel(string:"kernel vec4 your_code_here …”)!

override var outputImage: CIImage!
{
    return yourKernel.apply(withExtent: calcExtent(),
                            roiCallback: { (index, rect) -> CGRect in
                                           return calcROI(rect) },
                            arguments: [inputImage!, inputArgument!])
}
```

```
// Applying a CIKernel in a CIFilter subclass

// Only create the kernel once
static let yourKernel = CIKernel(string:"kernel vec4 your_code_here …")!

override var outputImage: CIImage!
{
    return yourKernel.apply(withExtent: calcExtent(),
                            roiCallback: { (index, rect) -> CGRect in
                                           return calcROI(rect) },
                            arguments: [inputImage!, inputArgument!])
}
```

```
// Applying a CIKernel in a CIFilter subclass

// Only create the kernel once
static let yourKernel = CIKernel(string:"kernel vec4 your_code_here …”)!

override var outputImage: CIImage!
{
    return yourKernel.apply(withExtent: calcExtent(),
                            roiCallback: { (index, rect) -> CGRect in
                                                return calcROI(rect) },
                            arguments: [inputImage!, inputArgument!])
}
```

```swift
// Applying a CIKernel in a CIFilter subclass

// Only create the kernel once
static let yourKernel = CIKernel(string:"kernel vec4 your_code_here …”)!

override var outputImage: CIImage!
{
    return yourKernel.apply(withExtent: calcExtent(),
                            roiCallback: { (index, rect) -> CGRect in
                                           return calcROI(rect) },
                            arguments: [inputImage!, inputArgument!])
}
```

```swift
// Applying a CIImageProcessor in a CIFilter subclass

override var outputImage: CIImage!
{
    return inputImage.withExtent( calcExtent(),
                              processorDescription: "myProcessor",
        argumentDigest: calcDigest(inputArgument: inputArgument),
        inputFormat:  kCIFormatBGRA8,
        outputFormat: kCIFormatRGBAf,
        options: [:],
        roiCallback: { (rect) -> CGRect in calcROI(rect) },
        processor: { ( input: CIImageProcessorInput,
                       output: CIImageProcessorOutput ) in
            // do what you want here
            // read from input,
            // use inputArgument,
            // write to output
    })
}
```

```swift
// Applying a CIImageProcessor in a CIFilter subclass

override var outputImage: CIImage!
{
    return inputImage.withExtent( calcExtent(),
                                  processorDescription: "myProcessor",
        argumentDigest: calcDigest(inputArgument: inputArgument),
        inputFormat:  kCIFormatBGRA8,
        outputFormat: kCIFormatRGBAf,
        options: [:],
        roiCallback: { (rect) -> CGRect in calcROI(rect) },
        processor: { ( input: CIImageProcessorInput,
                       output: CIImageProcessorOutput ) in
            // do what you want here
            // read from input,
            // use inputArgument,
            // write to output
    })
}
```

```swift
// Applying a CIImageProcessor in a CIFilter subclass

override var outputImage: CIImage!
{
    return inputImage.withExtent( calcExtent(),
                                   processorDescription: "myProcessor",
        argumentDigest: calcDigest(inputArgument: inputArgument),
        inputFormat:  kCIFormatBGRA8,
        outputFormat: kCIFormatRGBAf,
        options: [:],
        roiCallback: { (rect) -> CGRect in calcROI(rect) },
        processor: { ( input: CIImageProcessorInput,
                       output: CIImageProcessorOutput ) in
            // do what you want here
            // read from input,
            // use inputArgument,
            // write to output
    })
}
```

```swift
// Applying a CIImageProcessor in a CIFilter subclass

override var outputImage: CIImage!
{
    return inputImage.withExtent( calcExtent(),
                              processorDescription: "myProcessor",
        argumentDigest: calcDigest(inputArgument: inputArgument),
        inputFormat:  kCIFormatBGRA8,
        outputFormat: kCIFormatRGBAf,
        options: [:],
        roiCallback: { (rect) -> CGRect in calcROI(rect) },
        processor: { ( input: CIImageProcessorInput,
                       output: CIImageProcessorOutput ) in
            // do what you want here
            // read from input,
            // use inputArgument,
            // write to output
    })
}
```

```swift
// Applying a CIImageProcessor in a CIFilter subclass

override var outputImage: CIImage!
{
    return inputImage.withExtent( calcExtent(),
                                  processorDescription: "myProcessor",
        argumentDigest: calcDigest(inputArgument: inputArgument),
        inputFormat: kCIFormatBGRA8,
        outputFormat: kCIFormatRGBAf,
        options: [:],
        roiCallback: { (rect) -> CGRect in calcROI(rect) },
        processor: { ( input: CIImageProcessorInput,
                       output: CIImageProcessorOutput ) in
            // do what you want here
            // read from input,
            // use inputArgument,
            // write to output
    })
}
```

```swift
// Applying a CIImageProcessor in a CIFilter subclass

override var outputImage: CIImage!
{
    return inputImage.withExtent( calcExtent(),
                              processorDescription: "myProcessor",
        argumentDigest: calcDigest(inputArgument: inputArgument),
        inputFormat:  kCIFormatBGRA8,
        outputFormat: kCIFormatRGBAf,
        options: [:],
        roiCallback: { (rect) -> CGRect in calcROI(rect) },
        processor: { ( input: CIImageProcessorInput,
                       output: CIImageProcessorOutput ) in
            // do what you want here
            // read from input,
            // use inputArgument,
            // write to output
    })
}
```

```swift
// Applying a CIImageProcessor in a CIFilter subclass

override var outputImage: CIImage!
{
    return inputImage.withExtent( calcExtent(),
                                  processorDescription: "myProcessor",
        argumentDigest: calcDigest(inputArgument: inputArgument),
        inputFormat:  kCIFormatBGRA8,
        outputFormat: kCIFormatRGBAf,
        options: [:],
        roiCallback: { (rect) -> CGRect in calcROI(rect) },
        processor: { ( input: CIImageProcessorInput,
                       output: CIImageProcessorOutput ) in
            // do what you want here
            // read from input,
            // use inputArgument,
            // write to output
    })
}
```

Using CIImageProcessor

Using CIImageProcessor

Useful when you have an algorithm that isn’t suitable for CIKernel language

Using CIImageProcessor

Useful when you have an algorithm that isn't suitable for CIKernel language

A good example of this is an integral image

Using CIImageProcessor

Useful when you have an algorithm that isn't suitable for CIKernel language

A good example of this is an integral image

- Each output pixel contains the sum of all input pixels above and to the left

Using CIImageProcessor

Useful when you have an algorithm that isn't suitable for CIKernel language

A good example of this is an integral image

- Each output pixel contains the sum of all input pixels above and to the left
- This cannot be calculated as a traditional data-parallel pixel shader

Using CIImageProcessor

What's an integral image?

Input Image

1	4	5	3	2
0	2	4	6	3
3	7	8	2	1
6	8	3	4	7
7	2	1	0	3

Integral Image

1	5	10	13	15
1	7	16	25	30
4	17	34	45	51
10	31	51	66	79
17	40	61	76	92

Using CIImageProcessor

What's an integral image?

Input Image

1	4	5	3	2
0	2	4	6	3
3	7	8	2	1
6	8	3	4	7
7	2	1	0	3

Integral Image

1	5	10	13	15
1	7	16	25	30
4	17	34	45	51
10	31	51	66	79
17	40	61	76	92

Using CIImageProcessor

What’s an integral image?

Input Image

1	4	5	3	2
0	2	4	6	3
3	7	8	2	1
6	8	3	4	7
7	2	1	0	3

Integral Image

1	5	10	13	15
1	7	16	25	30
4	17	34	45	51
10	31	51	66	79
17	40	61	76	92

Using CIImageProcessor

What's an integral image?

Input Image

1	4	5	3	2
0	2	4	6	3
3	7	8	2	1
6	8	3	4	7
7	2	1	0	3

Integral Image

1	5	10	13	15
1	7	16	25	30
4	17	34	45	51
10	31	51	66	79
17	40	61	76	92

Using CIImageProcessor

What's an integral image?

Input Image

1	4	5	3	2
0	2	4	6	3
3	7	8	2	1
6	8	3	4	7
7	2	1	0	3

Integral Image

1	5	10	13	15
1	7	16	25	30
4	17	34	45	51
10	31	51	66	79
17	40	61	76	92

```swift
// CIImageProcessor block of integral image

processor: { ( input:  CIImageProcessorInput, output: CIImageProcessorOutput ) in
    let inputPointer = UnsafeMutablePointer <UInt8>(input.baseAddress)
    let outputPointer = UnsafeMutablePointer <Float>(output.baseAddress)

    let outputHeight = UInt(output.region.height)
    let outputWidth = UInt(output.region.width)
    let xShift = UInt(output.region.minX - input.region.minX)
    let yShift = UInt(output.region.minY - input.region.minY)

   for j in 0..<outputHeight {
        for i in 0..<outputWidth {
           // ... compute value of output(i,j) from input(i,j,xShift,yShift)
       }
    }
}
```

```swift
// CIImageProcessor block of integral image

processor: { ( input:  CIImageProcessorInput, output: CIImageProcessorOutput ) in
    let inputPointer = UnsafeMutablePointer <UInt8>(input.baseAddress)
    let outputPointer = UnsafeMutablePointer <Float>(output.baseAddress)

    let outputHeight = UInt(output.region.height)
    let outputWidth = UInt(output.region.width)
    let xShift = UInt(output.region.minX - input.region.minX)
    let yShift = UInt(output.region.minY - input.region.minY)

   for j in 0..<outputHeight {
        for i in 0..<outputWidth {
           // ... compute value of output(i,j) from input(i,j,xShift,yShift)
       }
    }
}
```

```swift
// CIImageProcessor block of integral image

processor: { ( input:  CIImageProcessorInput, output: CIImageProcessorOutput ) in
    let inputPointer = UnsafeMutablePointer <UInt8>(input.baseAddress)
    let outputPointer = UnsafeMutablePointer <Float>(output.baseAddress)

    let outputHeight = UInt(output.region.height)
    let outputWidth = UInt(output.region.width)
    let xShift = UInt(output.region.minX - input.region.minX)
    let yShift = UInt(output.region.minY - input.region.minY)

   for j in 0..<outputHeight {
        for i in 0..<outputWidth {
           // ... compute value of output(i,j) from input(i,j,xShift,yShift)
        }
    }
}
```

```swift
// CIImageProcessor block of integral image

processor: { ( input:  CIImageProcessorInput, output: CIImageProcessorOutput ) in
    let inputPointer = UnsafeMutablePointer <UInt8>(input.baseAddress)
    let outputPointer = UnsafeMutablePointer <Float>(output.baseAddress)

    let outputHeight = UInt(output.region.height)
    let outputWidth = UInt(output.region.width)
    let xShift = UInt(output.region.minX - input.region.minX)
    let yShift = UInt(output.region.minY - input.region.minY)

    for j in 0..<outputHeight {
        for i in 0..<outputWidth {
            // ... compute value of output(i,j) from input(i,j,xShift,yShift)
        }
    }
}
```

```swift
// CIImageProcessor block of integral image

processor: { ( input:  CIImageProcessorInput, output: CIImageProcessorOutput ) in
    let inputPointer = UnsafeMutablePointer <UInt8>(input.baseAddress)
    let outputPointer = UnsafeMutablePointer <Float>(output.baseAddress)

    let outputHeight = UInt(output.region.height)
    let outputWidth = UInt(output.region.width)
    let xShift = UInt(output.region.minX - input.region.minX)
    let yShift = UInt(output.region.minY - input.region.minY)

   for j in 0..<outputHeight {
        for i in 0..<outputWidth {
           // ... compute value of output(i,j) from input(i,j,xShift,yShift)
       }
    }
}
```

```swift
// CIImageProcessor block of integral image using MPS

processor: { ( input:  CIImageProcessorInput, output: CIImageProcessorOutput ) in
    let kernel = MPSImageIntegral(device: output.metalCommandBuffer?.device)

    let offsetX = output.region.minX - input.region.minX
    let offsetY = output.region.minY - input.region.minY
    kernel.offset = MPSOffset(x:Int(offsetX), y: Int(offsetY), z: 0)

    kernel.encodeToCommandBuffer(output.metalCommandBuffer?,
                                 sourceTexture: input.metalTexture,
                                 destinationTexture: output.metalTexture)
}
```

```swift
// CIImageProcessor block of integral image using MPS

processor: { ( input:  CIImageProcessorInput, output: CIImageProcessorOutput ) in
    let kernel = MPSImageIntegral(device: output.metalCommandBuffer?.device)

    let offsetX = output.region.minX - input.region.minX
    let offsetY = output.region.minY - input.region.minY
    kernel.offset = MPSOffset(x:Int(offsetX), y: Int(offsetY), z: 0)

    kernel.encodeToCommandBuffer(output.metalCommandBuffer?,
                                 sourceTexture: input.metalTexture,
                                 destinationTexture: output.metalTexture)
}
```

```swift
// CIImageProcessor block of integral image using MPS

processor: { ( input:  CIImageProcessorInput, output: CIImageProcessorOutput ) in
    let kernel = MPSImageIntegral(device: output.metalCommandBuffer?.device)

    let offsetX = output.region.minX – input.region.minX
    let offsetY = output.region.minY – input.region.minY
    kernel.offset = MPSOffset(x:Int(offsetX), y: Int(offsetY), z: 0)

    kernel.encodeToCommandBuffer(output.metalCommandBuffer?,
                                 sourceTexture: input.metalTexture,
                                 destinationTexture: output.metalTexture)
}
```

```swift
// CIImageProcessor block of integral image using MPS

processor: { ( input:  CIImageProcessorInput, output: CIImageProcessorOutput ) in
    let kernel = MPSImageIntegral(device: output.metalCommandBuffer?.device)

    let offsetX = output.region.minX - input.region.minX
    let offsetY = output.region.minY - input.region.minY
    kernel.offset = MPSOffset(x:Int(offsetX), y: Int(offsetY), z: 0)

    kernel.encodeToCommandBuffer(output.metalCommandBuffer?,
                                 sourceTexture: input.metalTexture,
                                 destinationTexture: output.metalTexture)
}
```

Use Integral Image to Do Fast Variable Box Blur

Use Integral Image to Do Fast Variable Box Blur

How Can You Use an Integral Image

Very fast box sums

Input Image

1	4	5	3	2
0	2	4	6	3
3	7	8	2	1
6	8	3	4	7
7	2	1	0	3

How Can You Use an Integral Image

Very fast box sums

Input Image

1	4	5	3	2
0	2	4	6	3
3	7	8	2	1
6	8	3	4	7
7	2	1	0	3

How Can You Use an Integral Image

Very fast box sums

Input Image

1	4	5	3	2
0	2	4	6	3
3	7	8	2	1
6	8	3	4	7
7	2	1	0	3

How Can You Use an Integral Image

Very fast box sums

Input Image

1	4	5	3	2
0	2	4	6	3
3	7	8	2	1
6	8	3	4	7
7	2	1	0	3

n^2 Reads

How Can You Use an Integral Image

Very fast box sums

Input Image

1	4	5	3	2
0	2	4	6	3
3	7	8	2	1
6	8	3	4	7
7	2	1	0	3

2n Reads

How Can You Use an Integral Image

Very fast box sums

Input Image

1	4	5	3	2
0	2	4	6	3
3	7	8	2	1
6	8	3	4	7
7	2	1	0	3

2n Reads

Integral Image

1	5	10	13	15
1	7	16	25	30
4	17	34	45	51
10	31	51	66	79
17	40	61	76	92

How Can You Use an Integral Image

Very fast box sums

Input Image

1	4	5	3	2
0	2	4	6	3
3	7	8	2	1
6	8	3	4	7
7	2	1	0	3

2n Reads

Integral Image

1	5	10	13	15
1	7	16	25	30
4	17	34	45	51
10	31	51	66	79
17	40	61	76	92

How Can You Use an Integral Image

Very fast box sums

Input Image

1	4	5	3	2
0	2	4	6	3
3	7	8	2	1
6	8	3	4	7
7	2	1	0	3

2n Reads

Integral Image

1	5	10	13	15
1	7	16	25	30
4	17	34	45	51
10	31	51	66	79
17	40	61	76	92

How Can You Use an Integral Image

Very fast box sums

Input Image

1	4	5	3	2
0	2	4	6	3
3	7	8	2	1
6	8	3	4	7
7	2	1	0	3

2n Reads

Integral Image

1	5	10	13	15
1	7	16	25	30
4	17	34	45	51
10	31	51	66	79
17	40	61	76	92

How Can You Use an Integral Image

Very fast box sums

Input Image

1	4	5	3	2
0	2	4	6	3
3	7	8	2	1
6	8	3	4	7
7	2	1	0	3

2n Reads

Integral Image

1	5	10	13	15
1	7	16	25	30
4	17	34	45	51
10	31	51	66	79
17	40	61	76	92

How Can You Use an Integral Image

Very fast box sums

Input Image

1	4	5	3	2
0	2	4	6	3
3	7	8	2	1
6	8	3	4	7
7	2	1	0	3

2n Reads

Integral Image

1	5	10	13	15
1	7	16	25	30
4	17	34	45	51
10	31	51	66	79
17	40	61	76	92

4 Reads

How Can You Use an Integral Image

Very fast box sums

$2 + 4 + 6 + 7 + 8 + 2 + 8 + 3 + 4 \quad == \quad 66 - 10 - 13 + 1$

Input Image

1	4	5	3	2
0	2	4	6	3
3	7	8	2	1
6	8	3	4	7
7	2	1	0	3

2n Reads

Integral Image

1	5	10	13	15
1	7	16	25	30
4	17	34	45	51
10	31	51	66	79
17	40	61	76	92

4 Reads

```
// CIKernel box blur from integral image

kernel vec4 boxBlur(sampler image, float radius, vec4 e) {
  vec2 c = destCoord();
  vec2 lowerLeft = clampToRect(c + vec2(-radius-1.0, -radius), e);
  vec2 upperRight = clampToRect(c + vec2(radius, radius+1.0), e);

  vec2 diagonal      = upperRight - lowerLeft;
  float usedArea     = abs(diagonal.x * diagonal.y);
  float originalArea = (2.0*radius+1.0) * (2.0*radius+1.0);

  vec4 ul = sample(image, samplerTransform(image, vec2(lowerLeft.x, upperRight.y)));
  vec4 ur = sample(image, samplerTransform(image, upperRight));
  vec4 ll = sample(image, samplerTransform(image, lowerLeft));
  vec4 lr = sample(image, samplerTransform(image, vec2(upperRight.x, lowerLeft.y)));

  return ( ul + lr - ur - ll ) * usedArea / originalArea;
}
```

```
// CIKernel box blur from integral image

kernel vec4 boxBlur(sampler image, float radius, vec4 e) {
  vec2 c = destCoord();
  vec2 lowerLeft = clampToRect(c + vec2(-radius-1.0, -radius), e);
  vec2 upperRight = clampToRect(c + vec2(radius, radius+1.0), e);

  vec2 diagonal      = upperRight - lowerLeft;
  float usedArea     = abs(diagonal.x * diagonal.y);
  float originalArea = (2.0*radius+1.0) * (2.0*radius+1.0);

  vec4 ul = sample(image, samplerTransform(image, vec2(lowerLeft.x, upperRight.y)));
  vec4 ur = sample(image, samplerTransform(image, upperRight));
  vec4 ll = sample(image, samplerTransform(image, lowerLeft));
  vec4 lr = sample(image, samplerTransform(image, vec2(upperRight.x, lowerLeft.y)));

  return ( ul + lr - ur - ll ) * usedArea / originalArea;
}
```

```
// CIKernel box blur from integral image

kernel vec4 boxBlur(sampler image, float radius, vec4 e) {
  vec2 c = destCoord();
  vec2 lowerLeft = clampToRect(c + vec2(-radius-1.0, -radius), e);
  vec2 upperRight = clampToRect(c + vec2(radius, radius+1.0), e);

  vec2 diagonal      = upperRight - lowerLeft;
  float usedArea     = abs(diagonal.x * diagonal.y);
  float originalArea = (2.0*radius+1.0) * (2.0*radius+1.0);

  vec4 ul = sample(image, samplerTransform(image, vec2(lowerLeft.x, upperRight.y)));
  vec4 ur = sample(image, samplerTransform(image, upperRight));
  vec4 ll = sample(image, samplerTransform(image, lowerLeft));
  vec4 lr = sample(image, samplerTransform(image, vec2(upperRight.x, lowerLeft.y)));

  return ( ul + lr - ur - ll ) * usedArea / originalArea;
}
```

```
// CIKernel box blur from integral image

kernel vec4 boxBlur(sampler image, float radius, vec4 e) {
  vec2 c = destCoord();
  vec2 lowerLeft = clampToRect(c + vec2(-radius-1.0, -radius), e);
  vec2 upperRight = clampToRect(c + vec2(radius, radius+1.0), e);

  vec2 diagonal      = upperRight - lowerLeft;
  float usedArea     = abs(diagonal.x * diagonal.y);
  float originalArea = (2.0*radius+1.0) * (2.0*radius+1.0);

  vec4 ul = sample(image, samplerTransform(image, vec2(lowerLeft.x, upperRight.y)));
  vec4 ur = sample(image, samplerTransform(image, upperRight));
  vec4 ll = sample(image, samplerTransform(image, lowerLeft));
  vec4 lr = sample(image, samplerTransform(image, vec2(upperRight.x, lowerLeft.y)));

  return ( ul + lr - ur - ll ) * usedArea / originalArea;
}
```

```
// CIKernel box blur from integral image

kernel vec4 boxBlur(sampler image, float radius, vec4 e) {
  vec2 c = destCoord();
  vec2 lowerLeft = clampToRect(c + vec2(-radius-1.0, -radius), e);
  vec2 upperRight = clampToRect(c + vec2(radius, radius+1.0), e);

  vec2 diagonal      = upperRight - lowerLeft;
  float usedArea     = abs(diagonal.x * diagonal.y);
  float originalArea = (2.0*radius+1.0) * (2.0*radius+1.0);

  vec4 ul = sample(image, samplerTransform(image, vec2(lowerLeft.x, upperRight.y)));
  vec4 ur = sample(image, samplerTransform(image, upperRight));
  vec4 ll = sample(image, samplerTransform(image, lowerLeft));
  vec4 lr = sample(image, samplerTransform(image, vec2(upperRight.x, lowerLeft.y)));

  return ( ul + lr - ur - ll ) * usedArea / originalArea;
}
```

```
// CIKernel box blur from integral image

kernel vec4 boxBlur(sampler image, float radius, vec4 e) {
  vec2 c = destCoord();
  vec2 lowerLeft = clampToRect(c + vec2(-radius-1.0, -radius), e);
  vec2 upperRight = clampToRect(c + vec2(radius, radius+1.0), e);

  vec2 diagonal      = upperRight - lowerLeft;
  float usedArea     = abs(diagonal.x * diagonal.y);
  float originalArea = (2.0*radius+1.0) * (2.0*radius+1.0);

  vec4 ul = sample(image, samplerTransform(image, vec2(lowerLeft.x, upperRight.y)));
  vec4 ur = sample(image, samplerTransform(image, upperRight));
  vec4 ll = sample(image, samplerTransform(image, lowerLeft));
  vec4 lr = sample(image, samplerTransform(image, vec2(upperRight.x, lowerLeft.y)));

  return ( ul + lr - ur - ll ) * usedArea / originalArea;
}
```

```
// CIKernel variable box blur from integral image and mask

kernel vec4 variableBoxBlur (sampler integralImage,
                             sampler maskImage,
                             float radius,
                             vec4 e) __attribute__((outputFormat(kCIFormatRGBAf)))
{
  vec4 v = unpremultiply ( sample ( maskImage, samplerCoord ( maskImage ) ) );
  radius *= v.r;
  return boxBlur (integralImage, radius, e);
}
```

```
// CIKernel variable box blur from integral image and mask

kernel vec4 variableBoxBlur (sampler integralImage,
                             sampler maskImage,
                             float radius,
                             vec4 e) __attribute__((outputFormat(kCIFormatRGBAf)))
{
  vec4 v = unpremultiply ( sample ( maskImage, samplerCoord ( maskImage ) ) );
  radius *= v.r;
  return boxBlur (integralImage, radius, e);
}
```

```
// CIKernel variable box blur from integral image and mask

kernel vec4 variableBoxBlur (sampler integralImage,
                             sampler maskImage,
                             float radius,
                             vec4 e) __attribute__((outputFormat(kCIFormatRGBAf)))
{
  vec4 v = unpremultiply ( sample ( maskImage, samplerCoord ( maskImage ) ) );
  radius *= v.r;
  return boxBlur (integralImage, radius, e);
}
```

```
// CIKernel variable box blur from integral image and mask

kernel vec4 variableBoxBlur (sampler integralImage,
                             sampler maskImage,
                             float radius,
                             vec4 e) __attribute__((outputFormat(kCIFormatRGBAf)))
{
  vec4 v = unpremultiply ( sample ( maskImage, samplerCoord ( maskImage ) ) );
  radius *= v.r;
  return boxBlur (integralImage, radius, e);
}
```

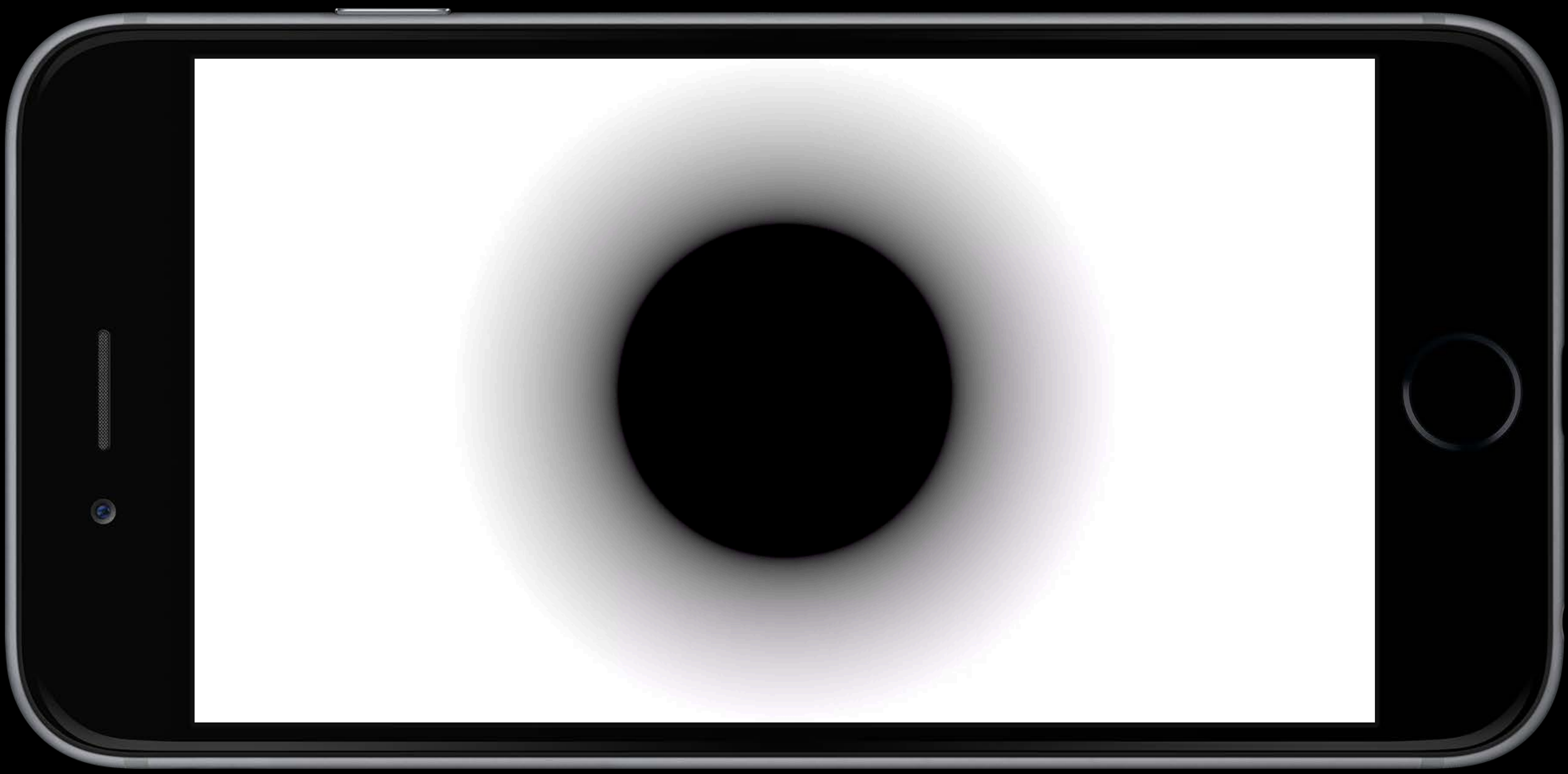
```
// CIKernel variable box blur from integral image and mask

kernel vec4 variableBoxBlur (sampler integralImage,
                             sampler maskImage,
                             float radius,
                             vec4 e) __attribute__((outputFormat(kCIFormatRGBAf)))
{
  vec4 v = unpremultiply ( sample ( maskImage, samplerCoord ( maskImage ) ) );
  radius *= v.r;
  return boxBlur (integralImage, radius, e);
}
```

```swift
// Create a mask image to control size of blur effect (0..1) -> (0..radius)

let maskImage =
   CIFilter(name: "CIRadialGradient",
      withInputParameters: [
         "inputCenter": centerOfEffect,
         "inputRadius0": innerRadius,
         "inputRadius1": outerRadius,
         "inputColor0": CIColor.black(),
         "inputColor1": CIColor.white()
       ])?.outputImage
```


8



Using CIImageProcessor

Tips and tricks

If your processor:

- Wants data in a color space other than the context working space,
  - Call `CIImage.byColorMatchingWorkingSpace(to: CGColorSpace)` on the processor input
- Returns data in a color space other than the context working space,
  - Call `CIImage.byColorMatchingColorSpace(toWorking: CGColorSpace)` on the processor output

Using CIImageProcessor

Tips and tricks

If your processor:

- Wants data in a color space other than the context working space,
  - Call `CIImage.byColorMatchingWorkingSpace(to: CGColorSpace)` on the processor input
- Returns data in a color space other than the context working space,
  - Call `CIImage.byColorMatchingColorSpace(toWorking: CGColorSpace)` on the processor output

You can see how your processor fits into a full-render graph by running with the CI_PRINT_TREE environment variable

```
// Example log with CI_PRINT_TREE=1

initial graph render_to_display (metal context 1 frame 1) extent=[0 0 1532 1032] =
  clamptoalpha roi=[0 0 1532 1032]
    colormatch workingspace-to-"Color LCD" roi=[0 0 1532 1032]
      affine [1 0 0 1 16 16] roi=[0 0 1532 1032]
        kernel variableBoxBlur(iImage,rImage,scale=16,origExtent) roi=[-16 -16 1532 1032]
          processor integralImage 0x12345678 roi=[-1 -1 1502 1002]
            clamp [0 0 1500 1000] roi=[-1 -1 1502 1002] opaque
              affine [1 0 0 -1 0 1000] roi=[0 0 1500 1000] opaque
                colormatch "sRGB IEC61966-2.1"-to-workingspace roi=[0 0 1500 1000] opaque
                  IOSurface BGRA8 alpha_one roi=[0 0 1500 1000] opaque
          colorkernel _radialGradient(params,c0,c1) roi=[0 0 1500 1000]
```

```
// Example log with CI_PRINT_TREE=1

initial graph render_to_display (metal context 1 frame 1) extent=[0 0 1532 1032] =
  clamptoalpha roi=[0 0 1532 1032]
    colormatch workingspace-to-"Color LCD" roi=[0 0 1532 1032]
      affine [1 0 0 1 16 16] roi=[0 0 1532 1032]
        kernel variableBoxBlur(iImage,rImage,scale=16,origExtent) roi=[-16 -16 1532 1032]
          processor integralImage 0x12345678 roi=[-1 -1 1502 1002]
            clamp [0 0 1500 1000] roi=[-1 -1 1502 1002] opaque
              affine [1 0 0 -1 0 1000] roi=[0 0 1500 1000] opaque
                colormatch "sRGB IEC61966-2.1"-to-workingspace roi=[0 0 1500 1000] opaque
                  IOSurface BGRA8 alpha_one roi=[0 0 1500 1000] opaque
          colorkernel _radialGradient(params,c0,c1) roi=[0 0 1500 1000]
```

```
// Example log with CI_PRINT_TREE=1

initial graph render_to_display (metal context 1 frame 1) extent=[0 0 1532 1032] =
  clamptoalpha roi=[0 0 1532 1032]
    colormatch workingspace-to-"Color LCD" roi=[0 0 1532 1032]
      affine [1 0 0 1 16 16] roi=[0 0 1532 1032]
        kernel variableBoxBlur(iImage,rImage,scale=16,origExtent) roi=[-16 -16 1532 1032]
          processor integralImage 0x12345678 roi=[-1 -1 1502 1002]
            clamp [0 0 1500 1000] roi=[-1 -1 1502 1002] opaque
              affine [1 0 0 -1 0 1000] roi=[0 0 1500 1000] opaque
                colormatch "sRGB IEC61966-2.1"-to-workingspace roi=[0 0 1500 1000] opaque
                  IOSurface BGRA8 alpha_one roi=[0 0 1500 1000] opaque
          colorkernel _radialGradient(params,c0,c1) roi=[0 0 1500 1000]
```

```
// Example log with CI_PRINT_TREE=1

initial graph render_to_display (metal context 1 frame 1) extent=[0 0 1532 1032] =
  clamptoalpha roi=[0 0 1532 1032]
    colormatch workingspace-to-"Color LCD" roi=[0 0 1532 1032]
      affine [1 0 0 1 16 16] roi=[0 0 1532 1032]
        kernel variableBoxBlur(iImage,rImage,scale=16,origExtent) roi=[-16 -16 1532 1032]
          processor integralImage 0x12345678 roi=[-1 -1 1502 1002]
            clamp [0 0 1500 1000] roi=[-1 -1 1502 1002] opaque
              affine [1 0 0 -1 0 1000] roi=[0 0 1500 1000] opaque
                colormatch "sRGB IEC61966-2.1"-to-workingspace roi=[0 0 1500 1000] opaque
                  IOSurface BGRA8 alpha_one roi=[0 0 1500 1000] opaque
          colorkernel _radialGradient(params,c0,c1) roi=[0 0 1500 1000]
```

```
// Example log with CI_PRINT_TREE=1

initial graph render_to_display (metal context 1 frame 1) extent=[0 0 1532 1032] =
  clamptoalpha roi=[0 0 1532 1032]
    colormatch workingspace-to-"Color LCD" roi=[0 0 1532 1032]
      affine [1 0 0 1 16 16] roi=[0 0 1532 1032]
        kernel variableBoxBlur(iImage,rImage,scale=16,origExtent) roi=[-16 -16 1532 1032]
          processor integralImage 0x12345678 roi=[-1 -1 1502 1002]
            clamp [0 0 1500 1000] roi=[-1 -1 1502 1002] opaque
              affine [1 0 0 -1 0 1000] roi=[0 0 1500 1000] opaque
                colormatch "sRGB IEC61966-2.1"-to-workingspace roi=[0 0 1500 1000] opaque
                  IOSurface BGRA8 alpha_one roi=[0 0 1500 1000] opaque
          colorkernel _radialGradient(params,c0,c1) roi=[0 0 1500 1000]
```

```
// Example log with CI_PRINT_TREE=1

initial graph render_to_display (metal context 1 frame 1) extent=[0 0 1532 1032] =
  clamptoalpha roi=[0 0 1532 1032]
    colormatch workingspace-to-"Color LCD" roi=[0 0 1532 1032]
      affine [1 0 0 1 16 16] roi=[0 0 1532 1032]
        kernel variableBoxBlur(iImage,rImage,scale=16,origExtent) roi=[-16 -16 1532 1032]
          processor integralImage 0x12345678 roi=[-1 -1 1502 1002]
            clamp [0 0 1500 1000] roi=[-1 -1 1502 1002] opaque
              affine [1 0 0 -1 0 1000] roi=[0 0 1500 1000] opaque
                colormatch "sRGB IEC61966-2.1"-to-workingspace roi=[0 0 1500 1000] opaque
                  IOSurface BGRA8 alpha_one roi=[0 0 1500 1000] opaque
          colorkernel _radialGradient(params,c0,c1) roi=[0 0 1500 1000]
```

```
// Example log with CI_PRINT_TREE=8

programs graph render_to_display (metal context 1 frame 1 tile 1) roi=[0 0 1532 1032] =
  program affine(clamp_to_alpha(premultiply(linear_to_srgb(
        unpremultiply(color_matrix_3x3(variableBoxBlur(0,1))))))) rois=[0 0 1532 1032]
    program RGBAf processor integralImage 0x12345678 () rois=[-1 -1 1502 1002]
      program clamp(affine(srgb_to_linear())) rois=[-1 -1 1502 1002]
        IOSurface BGRA8 1500x1000 alpha_one edge_clamp rois=[0 0 1500 1000]
    program _radialGradient() rois=[0 0 1500 1000]
```

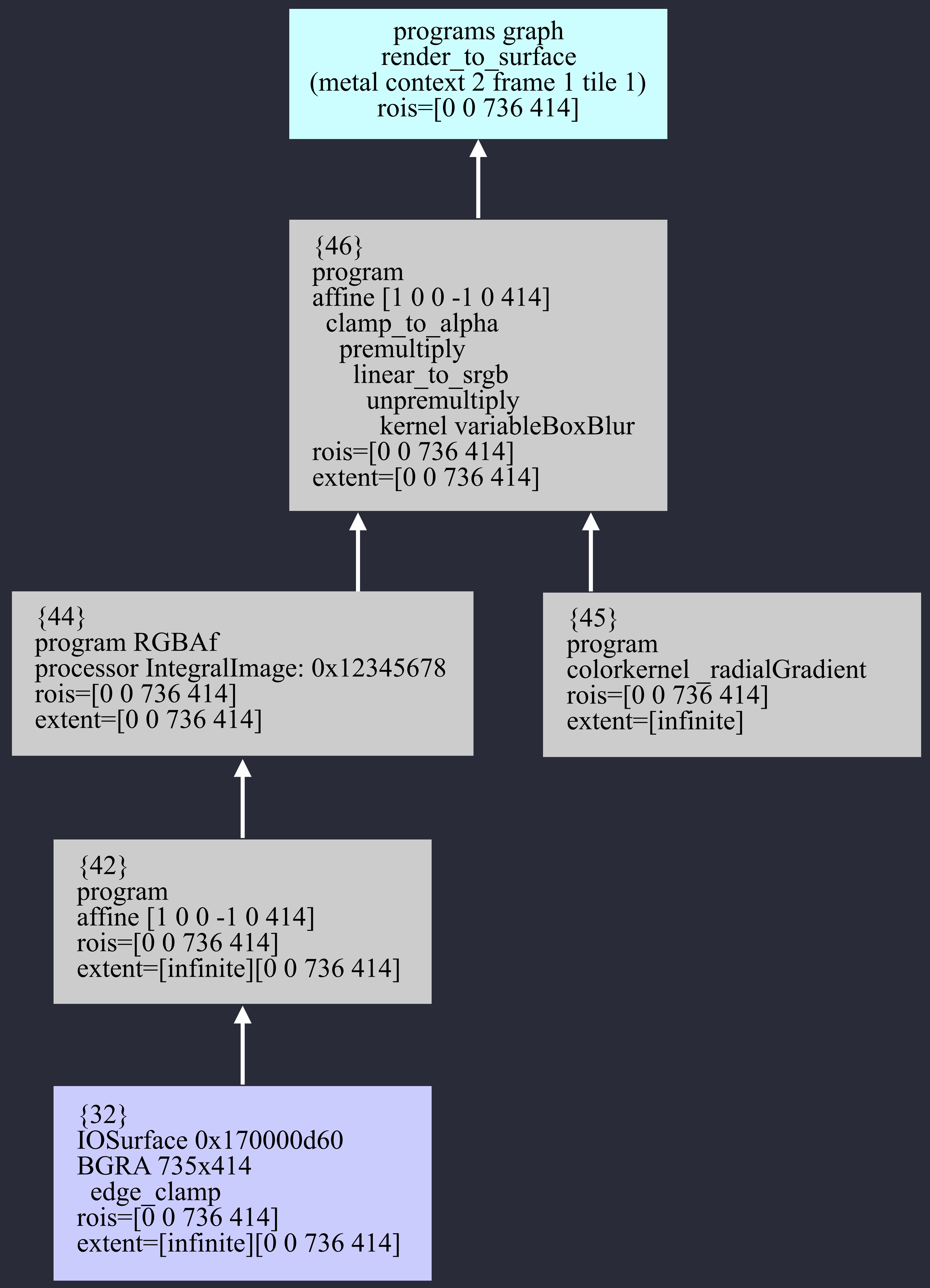
```
// Example log with CI_PRINT_TREE=8

programs graph render_to_display (metal context 1 frame 1 tile 1) roi=[0 0 1532 1032] =
  program affine(clamp_to_alpha(premultiply(linear_to_srgb(
          unpremultiply(color_matrix_3x3(variableBoxBlur(0,1))))))) rois=[0 0 1532 1032]
    program RGBAf processor integralImage 0x12345678 () rois=[-1 -1 1502 1002]
      program clamp(affine(srgb_to_linear())) rois=[-1 -1 1502 1002]
        IOSurface BGRA8 1500x1000 alpha_one edge_clamp rois=[0 0 1500 1000]
    program _radialGradient() rois=[0 0 1500 1000]
```

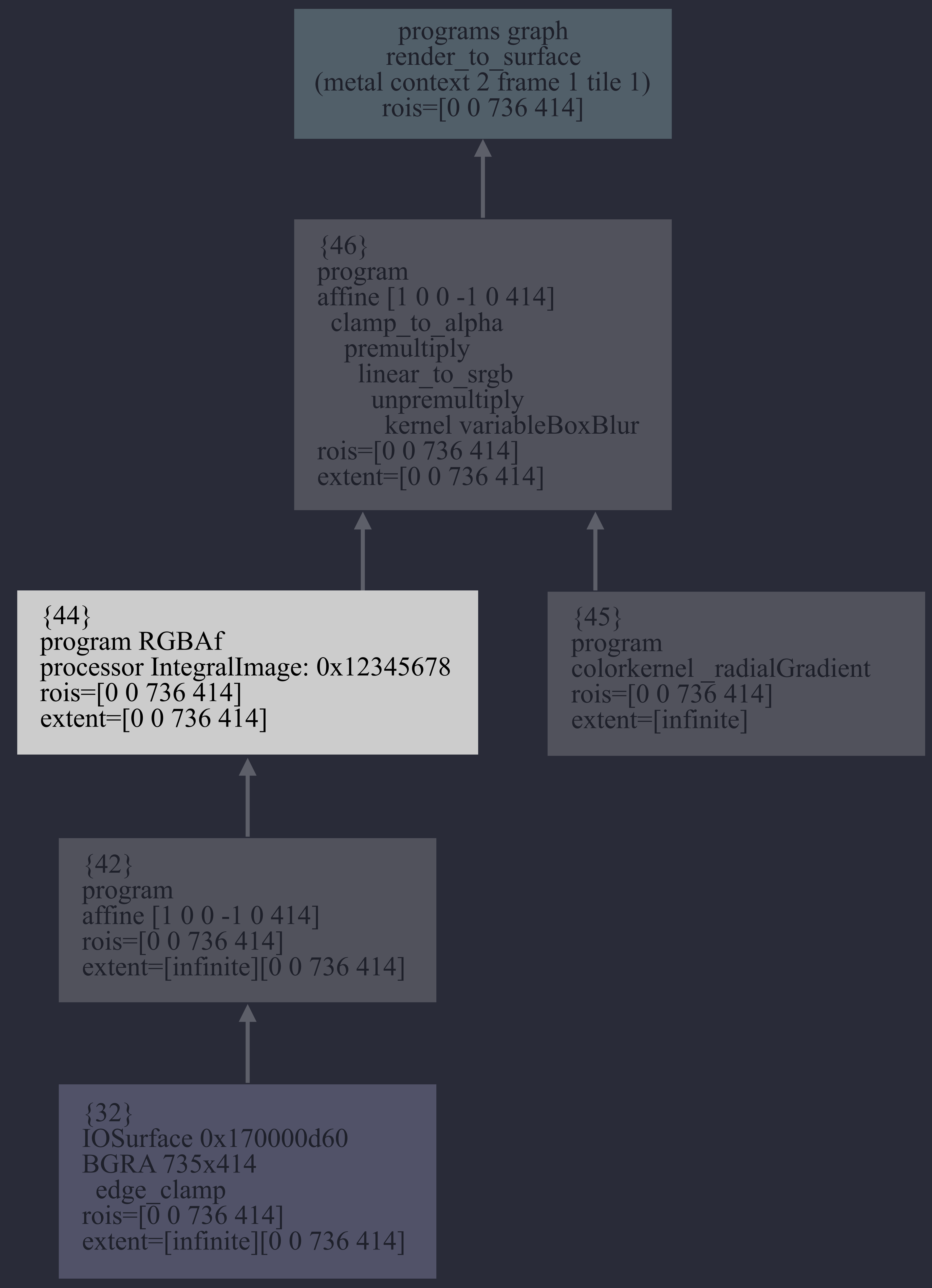
```
// Example log with CI_PRINT_TREE=8

programs graph render_to_display (metal context 1 frame 1 tile 1) roi=[0 0 1532 1032] =
  program affine(clamp_to_alpha(premultiply(linear_to_srgb(
          unpremultiply(color_matrix_3x3(variableBoxBlur(0,1))))))) rois=[0 0 1532 1032]
    program RGBAf processor integralImage 0x12345678 () rois=[-1 -1 1502 1002]
      program clamp(affine(srgb_to_linear())) rois=[-1 -1 1502 1002]
        IOSurface BGRA8 1500x1000 alpha_one edge_clamp rois=[0 0 1500 1000]
    program _radialGradient() rois=[0 0 1500 1000]
```

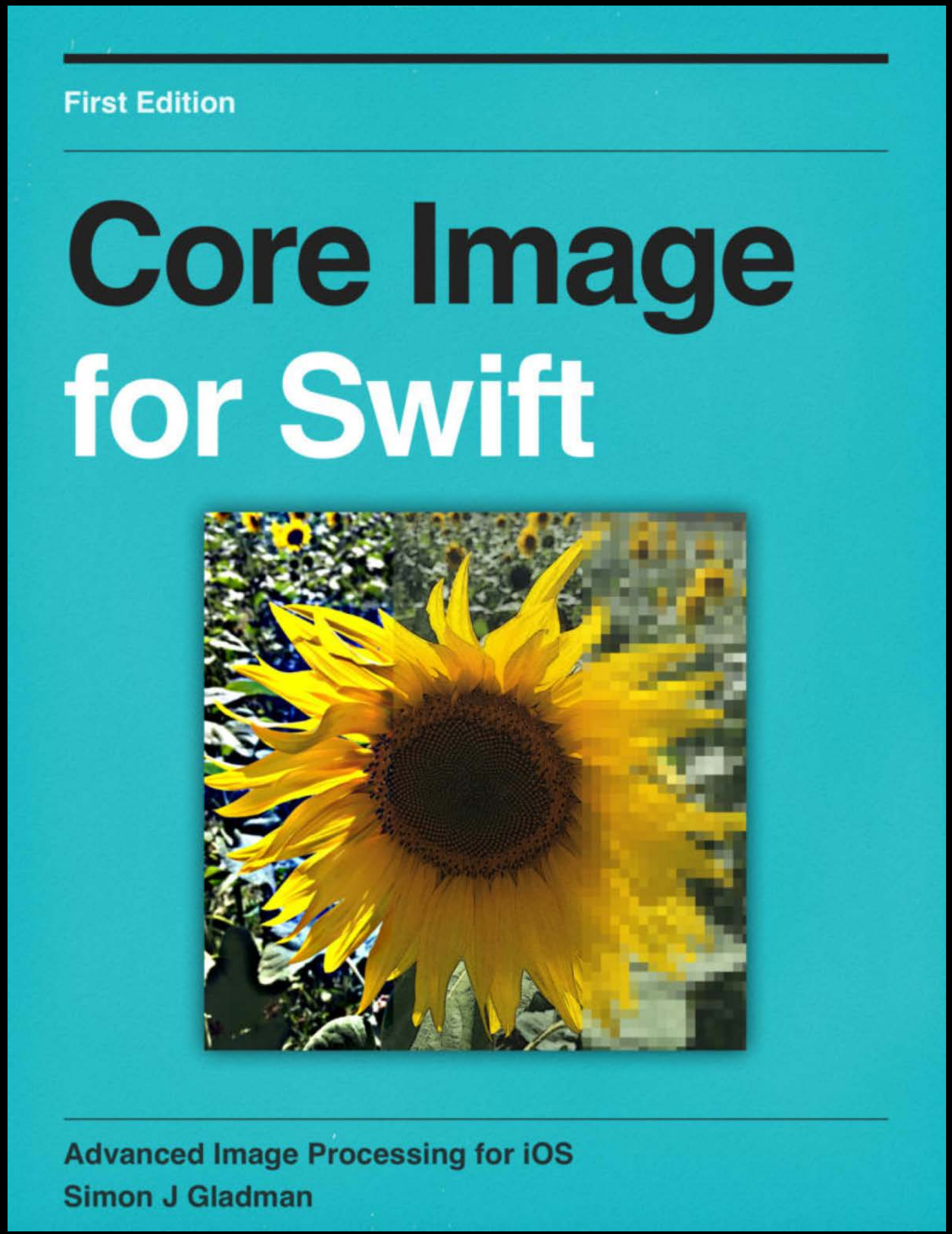
// Example log with CI_PRINT_TREE="8 graphviz"



// Example log with CI_PRINT_TREE="8 graphviz"



The Core Image Book Club Recommends



What You Learned Today

How to adjust RAW images on iOS

How to edit Live Photos

How to use CIImageProcessor

More Information

https://developer.apple.com/wwdc16/505

Related Sessions

Advances in iOS Photography	Pacific Heights	Tuesday 11:00AM
Working with Wide Color	Mission	Thursday 1:40PM

Labs

Live Photo and Core Image Lab	Graphics, Games, and Media Lab C	Thursday 1:30PM
Live Photo and Core Image Lab	Graphics, Games, and Media Lab D	Friday 9:00AM
Color Lab	Graphics, Games, and Media Lab C	Friday 4:00PM

WWDC16