Developing Core Image Filters for iOS

Session 515

Alexandre Naaman
Lead Customizer

Tony Chu
General Specialist

New to iOS 8

Write your own kernels

Introduced on OS X 10.4, now on iOS

- Main motivation—unable to write kernels as a combination of existing filters
- Use cases
  - “Per pixel effects” (e.g., hot pixels, vignette)
  - Distortions that cannot be represented as a matrix (Droste)

Agenda

Concepts

Agenda

Concepts

Examples of writing custom kernels

Agenda

Concepts

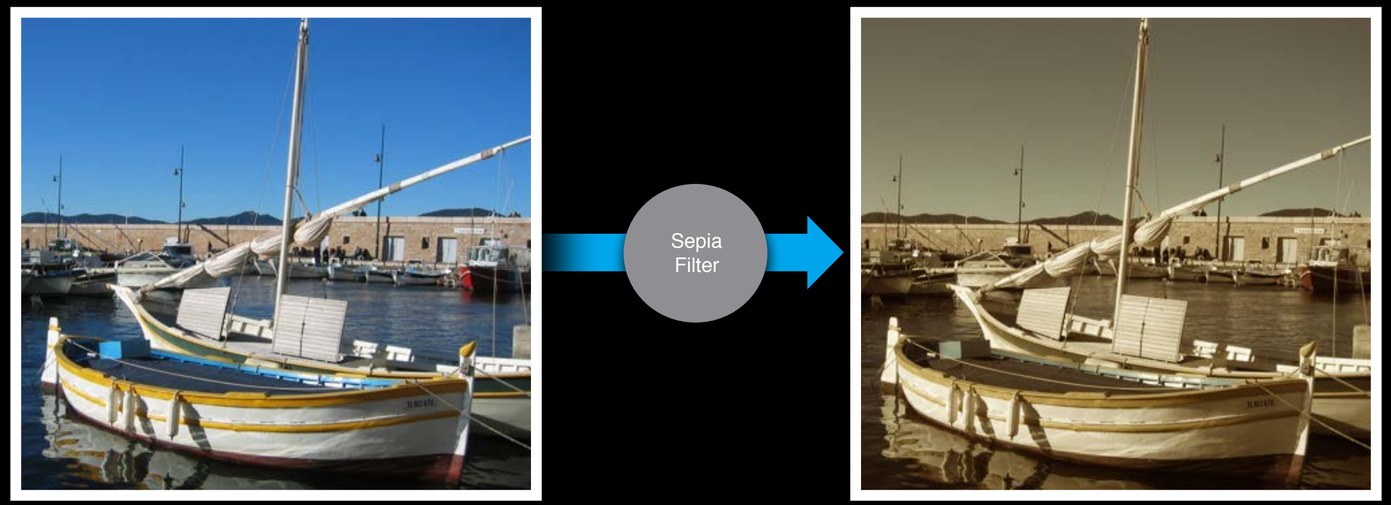
Examples of writing custom kernels

Platform differences

Key Concepts

Key Concepts

Filters perform per pixel operations on an image

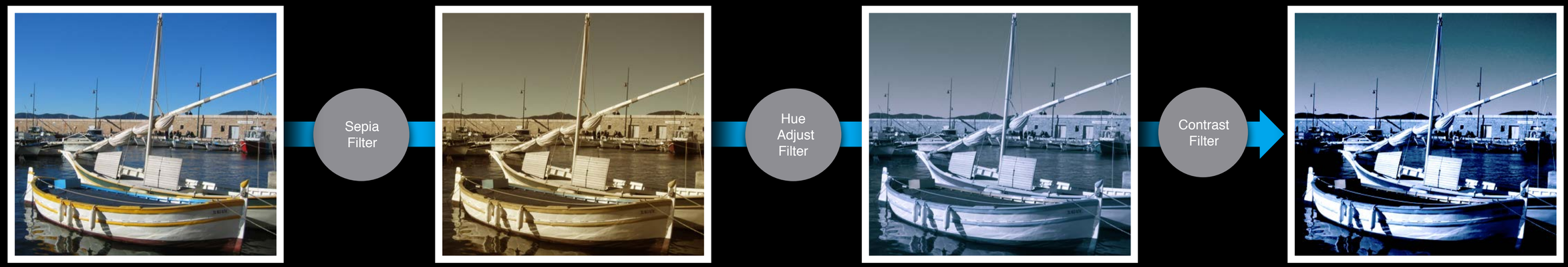


Original

Result

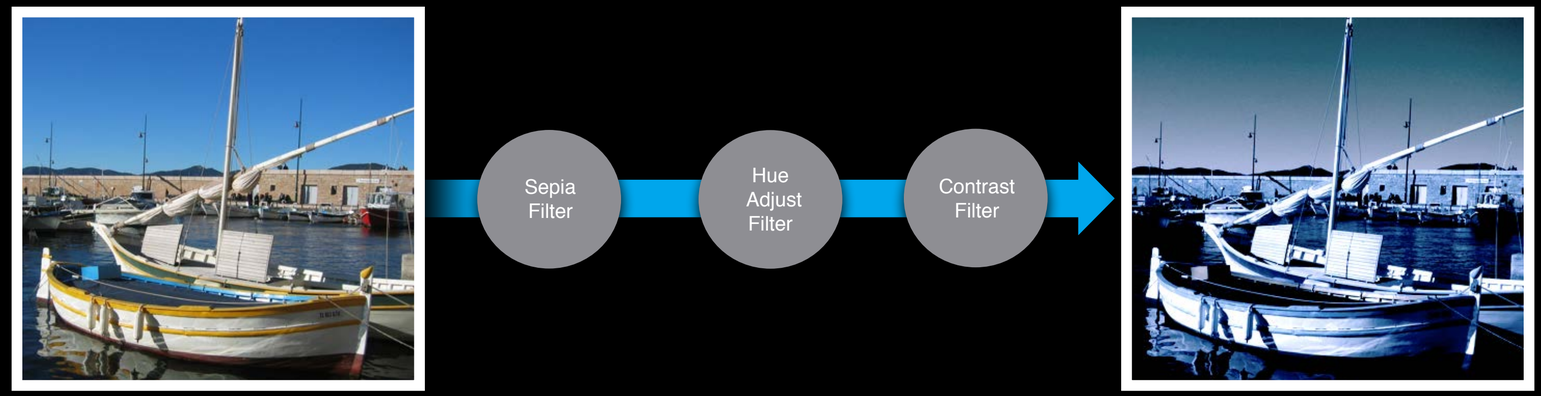
Key Concepts

Filters can be chained together for complex effects



Key Concepts

Intermediate images are lightweight objects

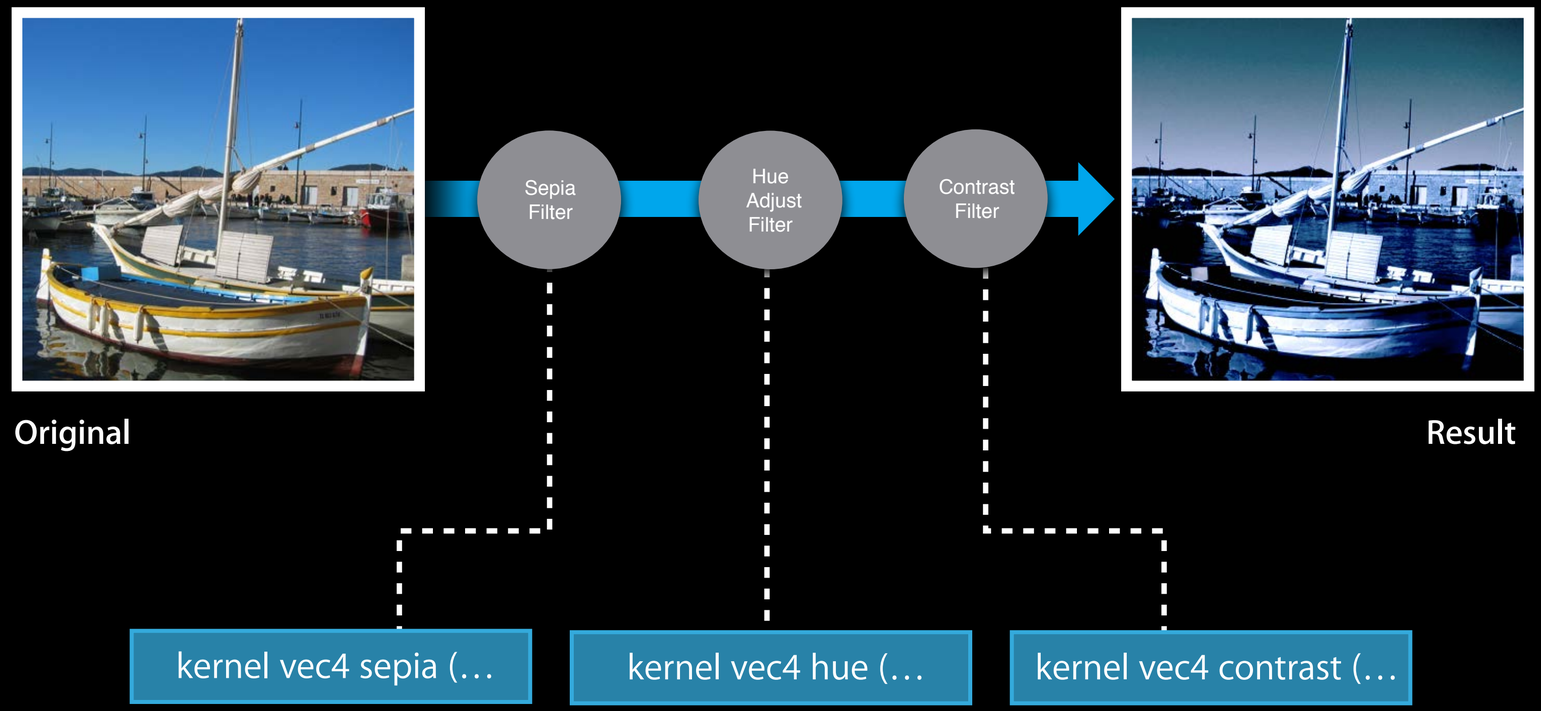


Original

Result

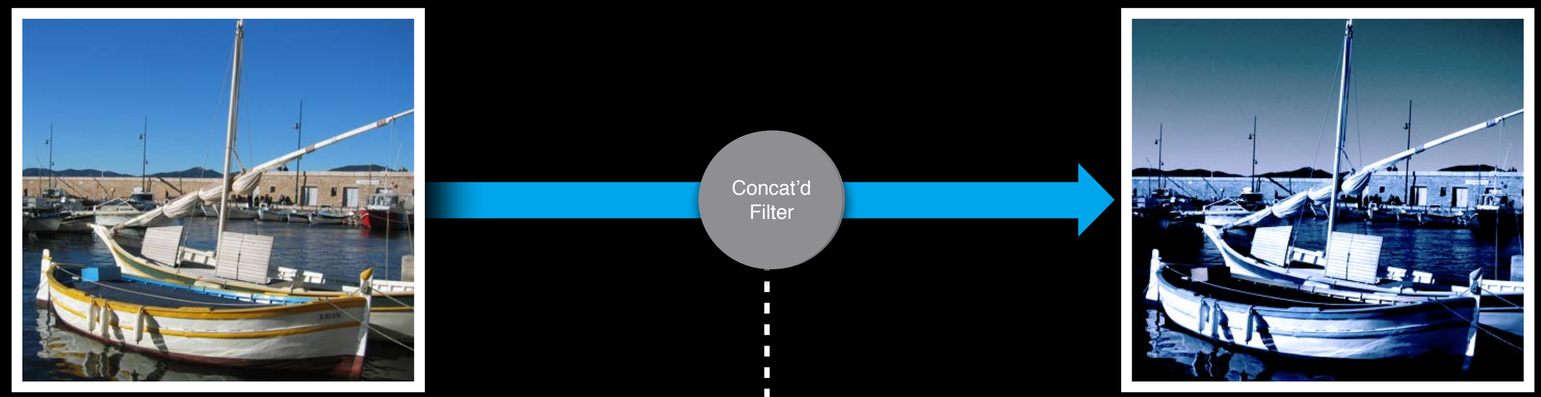
Key Concepts

Each filter has one or more kernel functions



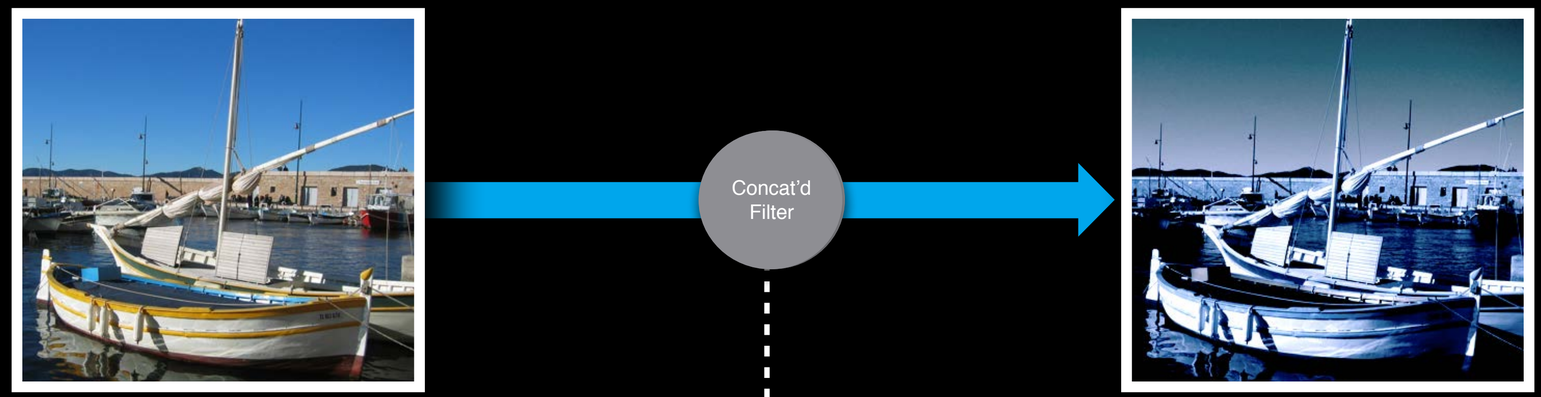
Key Concepts

Kernels concatenated into programs to minimize buffers



Key Concepts

Kernels concatenated into programs to minimize buffers




```
kernel vec4 sepia (…
kernel vec4 hue (…
kernel vec4 contrast (…
```

Core Image Classes

Core Image Classes

CIKernel

- Represents a program written in Core Image's kernel language

Core Image Classes

CIKernel

- Represents a program written in Core Image's kernel language

CIFilter

- Has mutable input parameters
- Uses one or more CIKernels to make a new image based on inputs

Core Image Classes

CIKernel

- Represents a program written in Core Image's kernel language

CIFilter

- Has mutable input parameters
- Uses one or more CIKernels to make a new image based on inputs

CIImage

- An immutable object that represents the recipe for an image
- Non zero origin (lower left) and possibly infinite bounds

Core Image Classes

CIKernel

- Represents a program written in Core Image's kernel language

CIFilter

- Has mutable input parameters
- Uses one or more CIKernels to make a new image based on inputs

CIImage

- An immutable object that represents the recipe for an image
- Non zero origin (lower left) and possibly infinite bounds

CIContext

- A object through which Core Image draws results

Concepts

Starting assumptions

```
void *inputBuffer={
```

```
};
```

Concepts

Starting assumptions

```
void *inputBuffer={
```

```
};
```

```
void *outputBuffer={
```

```
};
```

Concepts

Starting assumptions

```
void *inputBuffer={
```

```
};
```

```
for (j=0;j<height;j++){

}
```

```
void *outputBuffer={
```

```
};
```

Concepts

Starting assumptions

```
void *inputBuffer={
```

```
};
```

```
for (j=0;j<height;j++){
  for (i=0;i<width;i++){


  }
}
```

```
void *outputBuffer={
```

```
};
```

Concepts

Starting assumptions

```
void *inputBuffer={
```

```
};
```

```
for (j=0;j<height;j++){
  for (i=0;i<width;i++){
    outputBuffer[i][j]=
       processPixel (inputBuffer[i][j]);
  }
}
```

```
void *outputBuffer={
```

```
};
```

Concepts

Starting assumptions

```
void *inputBuffer={
```

```
};
```

```
for (j=0;j<height;j++){
  for (i=0;i<width;i++){
    outputBuffer[i][j]=
      processPixel (inputBuffer[i][j]);
  }
}
```

```
void *outputBuffer={
```

```
};
```

Concepts

Starting assumptions

```
void *inputBuffer={
```

```
};
```

```
for (j=0;j<height;j++){
  for (i=0;i<width;i++){
    outputBuffer[i][j]=
      processPixel (inputBuffer[i][j]);
  }
}
```

```
void *outputBuffer={
```

```
};
```


CIKernel: processPixel

Concepts

Starting assumptions

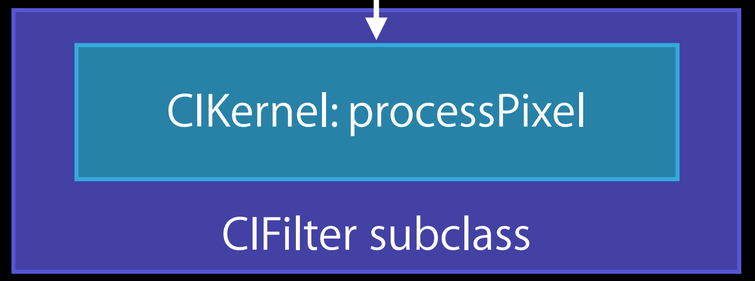
```
void *inputBuffer={
```

```
};
```

```
for (j=0;j<height;j++){
  for (i=0;i<width;i++){
    outputBuffer[i][j]=
      processPixel (inputBuffer[i][j]);
  }
}
```

```
void *outputBuffer={
```

```
};
```



Concepts

Starting assumptions

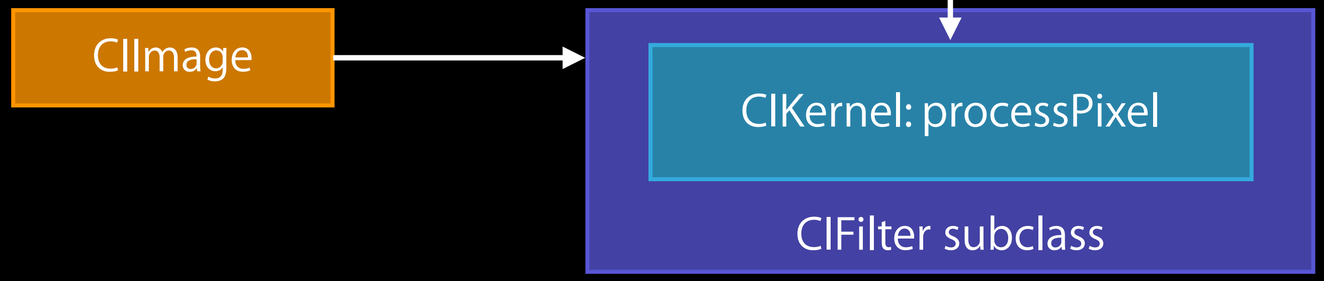
```
void *inputBuffer={
```

```
};
```

```
for (j=0;j<height;j++){
  for (i=0;i<width;i++){
    outputBuffer[i][j]=
      processPixel (inputBuffer[i][j]);
  }
}
```

```
void *outputBuffer={
```

```
};
```



Concepts

Starting assumptions

```
void *inputBuffer={
```

```
};
```

```
for (j=0;j<height;j++){
  for (i=0;i<width;i++){
    outputBuffer[i][j]=
      processPixel (inputBuffer[i][j]);
  }
}
```

```
void *outputBuffer={
```

```
};
```

CIImage

other input params

CIKernel: processPixel

CIFilter subclass

Concepts

Starting assumptions

```
void *inputBuffer={
};
```

```
for (j=0;j<height;j++){
  for (i=0;i<width;i++){
    outputBuffer[i][j]=
      processPixel (inputBuffer[i][j]);
  }
}
```

```
void *outputBuffer={
};
```

CIImage

other input params

CIKernel: processPixel

CIFilter subclass

CIImage

Workflow

Using Core Image on iOS

Workflow

Using Core Image on iOS

Create input CIImages

Workflow

Using Core Image on iOS

Create input CIImages

Subclass CIFilter

Workflow

Using Core Image on iOS

Create input CIImages

Subclass CIFilter

Get output CIImage

Workflow

Using Core Image on iOS

Create input CIImages

Subclass CIFilter

Get output CIImage

Use CIContext to render output image (to CGImageRef, EAGL context, etc.)

Workflow

Using Core Image on iOS

Create input CIImages

Subclass CIFilter

- Create kernel
- Apply kernel to input image(s) and pass parameters from filter

Get output CIImage

Use CIContext to render output image (to CGImageRef, EAGL context, etc.)

115 Built-in CIFilters on iOS

CIAccordianFoldTransition
CIAdditionCompositing
CIAffineClamp
CIAffineTile
CIAffineTransform
CIAreaHistogram
CIBarsSwipeTransition
CIBlendWithMask
CIBloom
CIBumpDistortion
CIBumpDistortionLinear
CICheckerboardGenerator
CICircleSplashDistortion
CICircularScreen
CICode128BarcodeGenerator
CIColorBlendMode
CIColorBurnBlendMode
CIColorControls
CIColorCube
CIColorDodgeBlendMode
CIColorInvert
CIColorMap
CIColorMatrix
CIColorMonochrome
CIColorClamp
CIColorCrossPolynomial
CIColorPolynomial
CIColorPosterize
CIConstantColorGenerator
CIConvolution3X3
CIConvolution5X5
CIConvolution9Horizontal
CIConvolution9Vertical
CICopyMachineTransition
CICrop
CIDarkenBlendMode
CIDifferenceBlendMode
CIDisintegrateWithMask
CIDissolveTransition
CIDivideBlendMode
CIDotScreen
CIEightfoldReflectedTile
CIExclusionBlendMode
CIExposureAdjust
CIFalseColor
CIFlashTransition
CIFourfoldReflectedTile
CIFourfoldTranslatedTile
CIGammaAdjust
CIGaussianBlur
CIGaussianGradient
CIGlassDistortion
CIGlideReflectedTile
CIGloom
CIHardLightBlendMode
CIHatchedScreen
CIHighlightShadowAdjust
CIHistogramDisplayFilter
CIHoleDistortion
CIHueAdjust
CIHueBlendMode
CILanczosScaleTransform
CILightenBlendMode
CILightTunnel
CILinearBurnBlendMode
CILinearDodgeBlendMode
CILinearGradient
CILineScreen
CILuminosityBlendMode
CIMaskToAlpha
CIMaximumComponent
CIMaximumCompositing
CIMinimumComponent
CIMinimumCompositing
CIModTransition
CIMultiplyBlendMode
CIMultiplyCompositing
CIOverlayBlendMode
CIPerspectiveCorrection
CIPerspectiveTile
CIPerspectiveTransform
CIPinchDistortion
CIPinLightBlendMode
CIPixellate
CIRadialGradient
CIRandomGenerator
CISaturationBlendMode
CIScreenBlendMode
CISepiaTone
CISharpenLuminance
CISixfoldReflectedTile
CISixfoldRotatedTile
CISoftLightBlendMode
CISourceAtopCompositing
CILinearToSRGBToneCurve
CISRGBToneCurveToLinear
CISourceInCompositing
CISourceOutCompositing
CISourceOverCompositing
CIStarShineGenerator
CIStraightenFilter
CIStripesGenerator
CISubtractBlendMode
CISwipeTransition
CITemperatureAndTint
CIToneCurve
CITriangleKaleidoscope
CITwelvefoldReflectedTile
CITwirlDistortion
CIUnsharpMask
CIVibrance
CIVignette
CIVortexDistortion
CIWhitePointAdjust
CIQRCodeGenerator

115 Built-in CIFilters on iOS (78 Color)

CIAccordianFoldTransition
CIAdditionCompositing
CIAffineClamp
CIAffineTile
CIAffineTransform
CIAreaHistogram
CIBarsSwipeTransition
CIBlendWithMask
CIBloom
CIBumpDistortion
CIBumpDistortionLinear
CICheckerboardGenerator
CICircleSplashDistortion
CICircularScreen
CICode128BarcodeGenerator
CIColorBlendMode
CIColorBurnBlendMode
CIColorControls
CIColorCube
CIColorDodgeBlendMode
CIColorInvert
CIColorMap
CIColorMatrix
CIColorMonochrome
CIColorClamp
CIColorCrossPolynomial
CIColorPolynomial
CIColorPosterize
CIConstantColorGenerator
CIConvolution3X3
CIConvolution5X5
CIConvolution9Horizontal
CIConvolution9Vertical
CICopyMachineTransition
CICrop
CIDarkenBlendMode
CIDifferenceBlendMode
CIDisintegrateWithMask
CIDissolveTransition
CIDivideBlendMode
CIDotScreen
CIEightfoldReflectedTile
CIExclusionBlendMode
CIExposureAdjust
CIFalseColor
CIFlashTransition
CIFourfoldReflectedTile
CIFourfoldTranslatedTile
CIGammaAdjust
CIGaussianBlur
CIGaussianGradient
CIGlassDistortion
CIGlideReflectedTile
CIGloom
CIHardLightBlendMode
CIHatchedScreen
CIHighlightShadowAdjust
CIHistogramDisplayFilter
CIHoleDistortion
CIHueAdjust
CIHueBlendMode
CILanczosScaleTransform
CILightenBlendMode
CILightTunnel
CILinearBurnBlendMode
CILinearDodgeBlendMode
CILinearGradient
CILineScreen
CILuminosityBlendMode
CIMaskToAlpha
CIMaximumComponent
CIMaximumCompositing
CIMinimumComponent
CIMinimumCompositing
CIModTransition
CIMultiplyBlendMode
CIMultiplyCompositing
CIOverlayBlendMode
CIPerspectiveCorrection
CIPerspectiveTile
CIPerspectiveTransform
CIPinchDistortion
CIPinLightBlendMode
CIPixellate
CIRadialGradient
CIRandomGenerator
CISaturationBlendMode
CIScreenBlendMode
CISepiaTone
CISharpenLuminance
CISixfoldReflectedTile
CISixfoldRotatedTile
CISoftLightBlendMode
CISourceAtopCompositing
CILinearToSRGBToneCurve
CISRGBToneCurveToLinear
CISourceInCompositing
CISourceOutCompositing
CISourceOverCompositing
CIStarShineGenerator
CIStraightenFilter
CIStripesGenerator
CISubtractBlendMode
CISwipeTransition
CITemperatureAndTint
CIToneCurve
CITriangleKaleidoscope
CITwelvefoldReflectedTile
CITwirlDistortion
CIUnsharpMask
CIVibrance
CIVignette
CIVortexDistortion
CIWhitePointAdjust
CIQRCodeGenerator

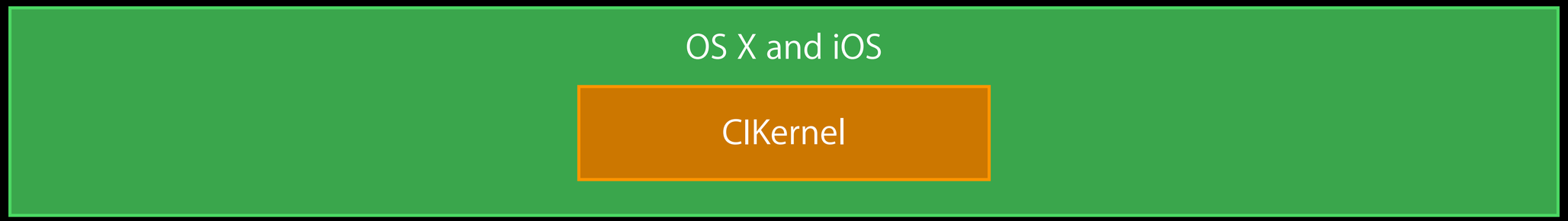
115 Built-in CIFilters on iOS (27 Warps)

CIAccordianFoldTransition
CIAdditionCompositing
CIAffineClamp
CIAffineTile
CIAffineTransform
CIAreaHistogram
CIBarsSwipeTransition
CIBlendWithMask
CIBloom
CIBumpDistortion
CIBumpDistortionLinear
CICheckerboardGenerator
CICircleSplashDistortion
CICircularScreen
CICode128BarcodeGenerator
CIColorBlendMode
CIColorBurnBlendMode
CIColorControls
CIColorCube
CIColorDodgeBlendMode
CIColorInvert
CIColorMap
CIColorMatrix
CIColorMonochrome
CIColorClamp
CIColorCrossPolynomial
CIColorPolynomial
CIColorPosterize
CIConstantColorGenerator
CIConvolution3X3
CIConvolution5X5
CIConvolution9Horizontal
CIConvolution9Vertical
CICopyMachineTransition
CICrop
CIDarkenBlendMode
CIDifferenceBlendMode
CIDisintegrateWithMask
CIDissolveTransition
CIDivideBlendMode
CIDotScreen
CIEightfoldReflectedTile
CIExclusionBlendMode
CIExposureAdjust
CIFalseColor
CIFlashTransition
CIFourfoldReflectedTile
CIFourfoldTranslatedTile
CIGammaAdjust
CIGaussianBlur
CIGaussianGradient
CIGlassDistortion
CIGlideReflectedTile
CIGloom
CIHardLightBlendMode
CIHatchedScreen
CIHighlightShadowAdjust
CIHistogramDisplayFilter
CIHoleDistortion
CIHueAdjust
CIHueBlendMode
CILanczosScaleTransform
CILightenBlendMode
CILightTunnel
CILinearBurnBlendMode
CILinearDodgeBlendMode
CILinearGradient
CILineScreen
CILuminosityBlendMode
CIMaskToAlpha
CIMaximumComponent
CIMaximumCompositing
CIMinimumComponent
CIMinimumCompositing
CIModTransition
CIMultiplyBlendMode
CIMultiplyCompositing
CIOverlayBlendMode
CIPerspectiveCorrection
CIPerspectiveTile
CIPerspectiveTransform
CIPinchDistortion
CIPinLightBlendMode
CIPixellate
CIRadialGradient
CIRandomGenerator
CISaturationBlendMode
CIScreenBlendMode
CISepiaTone
CISharpenLuminance
CISixfoldReflectedTile
CISixfoldRotatedTile
CISoftLightBlendMode
CISourceAtopCompositing
CILinearToSRGBToneCurve
CISRGBToneCurveToLinear
CISourceInCompositing
CISourceOutCompositing
CISourceOverCompositing
CIStarShineGenerator
CIStraightenFilter
CIStripesGenerator
CISubtractBlendMode
CISwipeTransition
CITemperatureAndTint
CIToneCurve
CITriangleKaleidoscope
CITwelvefoldReflectedTile
CITwirlDistortion
CIUnsharpMask
CIVibrance
CIVignette
CIVortexDistortion
CIWhitePointAdjust
CIQRCodeGenerator

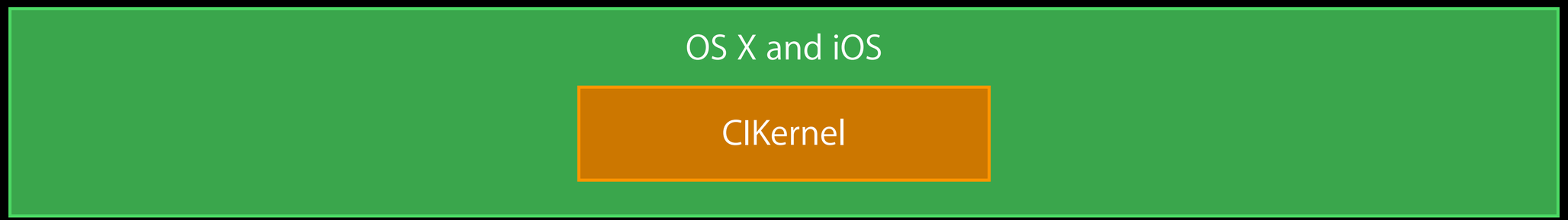
115 Built-in CIFilters on iOS (7 Convolution)

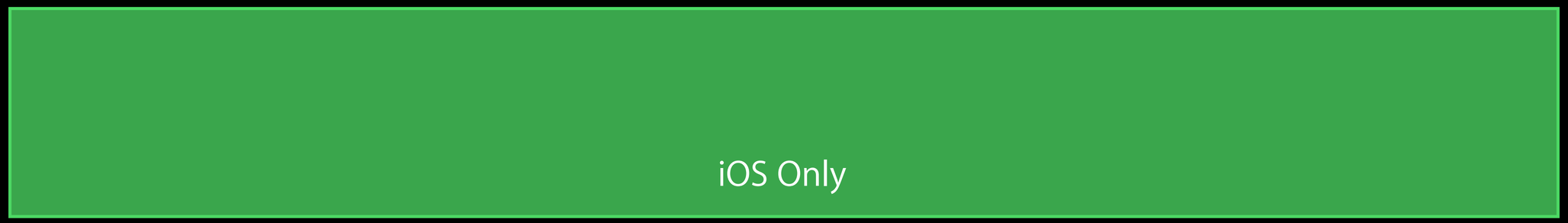
CIAccordianFoldTransition
CIAdditionCompositing
CIAffineClamp
CIAffineTile
CIAffineTransform
CIAreaHistogram
CIBarsSwipeTransition
CIBlendWithMask
CIBloom
CIBumpDistortion
CIBumpDistortionLinear
CICheckerboardGenerator
CICircleSplashDistortion
CICircularScreen
CICode128BarcodeGenerator
CIColorBlendMode
CIColorBurnBlendMode
CIColorControls
CIColorCube
CIColorDodgeBlendMode
CIColorInvert
CIColorMap
CIColorMatrix
CIColorMonochrome
CIColorClamp
CIColorCrossPolynomial
CIColorPolynomial
CIColorPosterize
CIConstantColorGenerator
CIConvolution3X3
CIConvolution5X5
CIConvolution9Horizontal
CIConvolution9Vertical
CICopyMachineTransition
CICrop
CIDarkenBlendMode
CIDifferenceBlendMode
CIDisintegrateWithMask
CIDissolveTransition
CIDivideBlendMode
CIDotScreen
CIEightfoldReflectedTile
CIExclusionBlendMode
CIExposureAdjust
CIFalseColor
CIFlashTransition
CIFourfoldReflectedTile
CIFourfoldTranslatedTile
CIGammaAdjust
CIGaussianBlur
CIGaussianGradient
CIGlassDistortion
CIGlideReflectedTile
CIGloom
CIHardLightBlendMode
CIHatchedScreen
CIHighlightShadowAdjust
CIHistogramDisplayFilter
CIHoleDistortion
CIHueAdjust
CIHueBlendMode
CILanczosScaleTransform
CILightenBlendMode
CILightTunnel
CILinearBurnBlendMode
CILinearDodgeBlendMode
CILinearGradient
CILineScreen
CILuminosityBlendMode
CIMaskToAlpha
CIMaximumComponent
CIMaximumCompositing
CIMinimumComponent
CIMinimumCompositing
CIModTransition
CIMultiplyBlendMode
CIMultiplyCompositing
CIOverlayBlendMode
CIPerspectiveCorrection
CIPerspectiveTile
CIPerspectiveTransform
CIPinchDistortion
CIPinLightBlendMode
CIPixellate
CIRadialGradient
CIRandomGenerator
CISaturationBlendMode
CIScreenBlendMode
CISepiaTone
CISharpenLuminance
CISixfoldReflectedTile
CISixfoldRotatedTile
CISoftLightBlendMode
CISourceAtopCompositing
CILinearToSRGBToneCurve
CISRGBToneCurveToLinear
CISourceInCompositing
CISourceOutCompositing
CISourceOverCompositing
CIStarShineGenerator
CIStraightenFilter
CIStripesGenerator
CISubtractBlendMode
CISwipeTransition
CITemperatureAndTint
CIToneCurve
CITriangleKaleidoscope
CITwelvefoldReflectedTile
CITwirlDistortion
CIUnsharpMask
CIVibrance
CIVignette
CIVortexDistortion
CIWhitePointAdjust
CIQRCodeGenerator

Anatomy of CIKernel

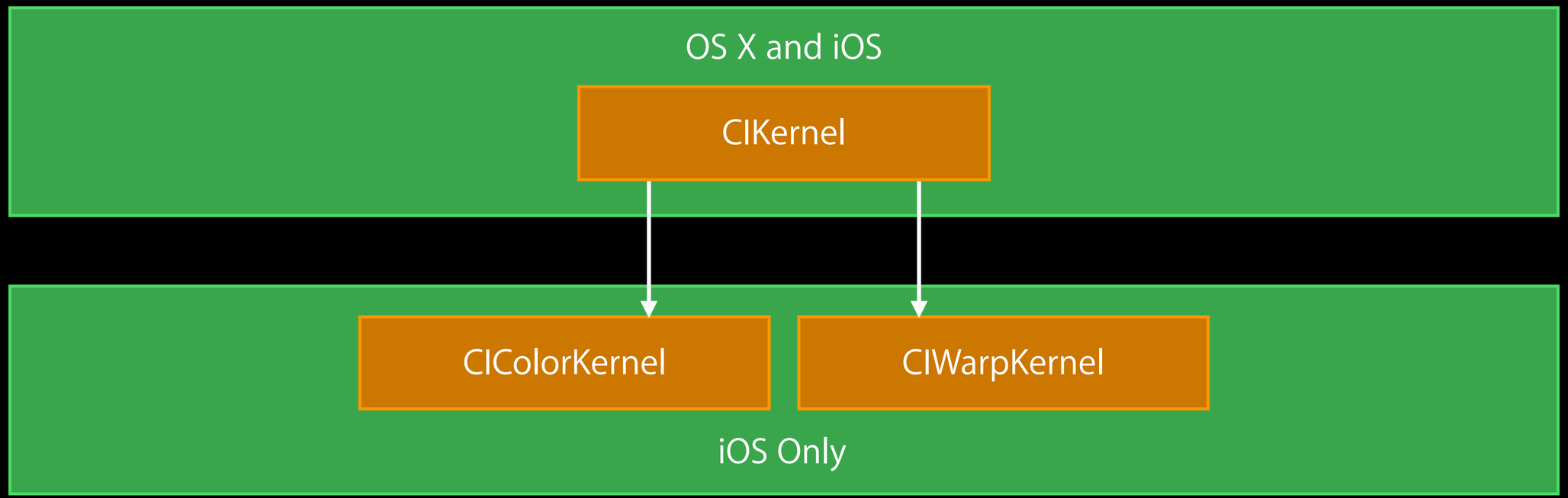


Anatomy of CIKernel





Anatomy of CIKernel



Anatomy of CIKernel

```
@interface CIKernel
+ (CIKernel *) kernelWithString:(NSString *);
- (CIImage *) applyWithExtent:(CGRect)extent
                  roiCallback:(CIKernelROICallback)callback
                  arguments:(NSArray *)arguments;
@end
```

Performs no actual work

Anatomy of CIKernel

```
@interface CIKernel
+ (CIKernel *) kernelWithString:(NSString *);
- (CIImage *) applyWithExtent:(CGRect)extent
                  roiCallback:(CIKernelROICallback)callback
                  arguments:(NSArray *)arguments;
@end
```

Performs no actual work

GLSL + extensions for imaging

Inputs and outputs are floats

Color Kernels

CIColorKernel

Basic Principles of Color Kernels

CIColorKernel

Color Kernel Example

```
kernel vec4       doNothing ( __sample s ) { return s.rgba; }
```

Color Kernel Example

```
kernel vec4       doNothing ( __sample s ) { return s.rgba; }
```

Color Kernel Example

```
kernel vec4       doNothing ( __sample s ) { return s.rgba; }
kernel vec4 swapRedAndGreen ( __sample s ) { return s.grba; }
```

Color Kernel Example

```
kernel vec4       doNothing ( __sample s ) { return s.rgba; }
kernel vec4 swapRedAndGreen ( __sample s ) { return s.grba; }
```

Color Kernel Example

```
kernel vec4       doNothing ( __sample s ) { return s.rgba; }
kernel vec4 swapRedAndGreen ( __sample s ) { return s.grba; }
```

Color Kernel Example

```
kernel vec4       doNothing ( __sample s ) { return s.rgba; }
kernel vec4 swapRedAndGreen ( __sample s ) { return s.grba; }
```

Color Kernel Example

```
kernel vec4       doNothing ( __sample s ) { return s.rgba; }
kernel vec4 swapRedAndGreen ( __sample s ) { return s.grba; }
kernel vec4 swapRedAndGreenAmount ( __sample s, float amount )
                          { return mix(s.rgba, s.grba, amount); }
```

Color Kernel Example

```
kernel vec4       doNothing ( __sample s ) { return s.rgba; }
kernel vec4 swapRedAndGreen ( __sample s ) { return s.grba; }
kernel vec4 swapRedAndGreenAmount ( __sample s, float amount )
                          { return mix(s.rgba, s.grba, amount); }
```

Color Kernel Example

```
kernel vec4       doNothing ( __sample s ) { return s.rgba; }
kernel vec4 swapRedAndGreen ( __sample s ) { return s.grba; }
kernel vec4 swapRedAndGreenAmount ( __sample s, float amount )
                          { return mix(s.rgba, s.grba, amount); }
```

Color Kernel Example

```
kernel vec4       doNothing ( __sample s ) { return s.rgba; }
kernel vec4 swapRedAndGreen ( __sample s ) { return s.grba; }
kernel vec4 swapRedAndGreenAmount ( __sample s, float amount )
                          { return mix(s.rgba, s.grba, amount); }
```

Color Kernel Example

```
kernel vec4       doNothing ( __sample s ) { return s.rgba; }
kernel vec4 swapRedAndGreen ( __sample s ) { return s.grba; }
kernel vec4 swapRedAndGreenAmount ( __sample s, float amount )
                          { return mix(s.rgba, s.grba, amount); }
```

Color Kernel Example

```
kernel vec4       doNothing ( __sample s ) { return s.rgba; }
kernel vec4 swapRedAndGreen ( __sample s ) { return s.grba; }
kernel vec4 swapRedAndGreenAmount ( __sample s, float amount )
                          { return mix(s.rgba, s.grba, amount); }
```

Color Kernel Example

```
kernel vec4       doNothing ( __sample s ) { return s.rgba; }
kernel vec4 swapRedAndGreen ( __sample s ) { return s.grba; }
kernel vec4 swapRedAndGreenAmount ( __sample s, float amount )
                          { return mix(s.rgba, s.grba, amount); }
```

Color Kernel Example

Subclassing CIFilter

Color Kernel Example

Subclassing CIFilter

```
@interface SwapRedGreenFilter : CIFilter
@property (retain, nonatomic) CIImage *inputImage;
@property (copy, nonatomic) NSNumber *inputAmount;
@end
```

Color Kernel Example

Subclassing CIFilter

```
@interface SwapRedGreenFilter : CIFilter
@property (retain, nonatomic) CIImage *inputImage;
@property (copy, nonatomic) NSNumber *inputAmount;
@end

@implementation SwapRedGreenFilter

-(CIKernel *)myKernel { … } // convenience
+(NSDictionary *)customAttributes { … }
-(CIImage *)outputImage { … }

@end
```

Color Kernel Example

Applying the kernel inside of a CIFilter

```
- (CIKernel *) myKernel {
    static CIColorKernel *kernel = nil;
    static dispatch_once_t once;
    dispatch_once(&once, ^{
          kernel = [CIColorKernel kernelWithString:
          @“kernel vec4 swapRedAndGreenAmount ( __sample s, float amount )
                              { return mix(s.rgba, s.grba, amount); }” ]; });
    return kernel;
}
- (CIImage *)outputImage
{
    CGRect dod = inputImage.extent ;
    return [ [self myKernel] applyWithExtent : dod
                             arguments : @[inputImage, inputAmount]];
}
```

Color Kernel Example

Applying the kernel inside of a CIFilter

```
- (CIKernel *) myKernel {
    static CIColorKernel *kernel = nil;
    static dispatch_once_t once;
    dispatch_once(&once, ^{
          kernel = [CIColorKernel kernelWithString:
              @“kernel vec4 swapRedAndGreenAmount ( __sample s, float amount )
                        { return mix(s.rgba, s.grba, amount); }” ]; });
    return kernel;
}
- (CIImage *)outputImage
{
    CGRect dod = inputImage.extent ;
    return [ [self myKernel] applyWithExtent : dod
                          arguments : @[inputImage, inputAmount]];
}
```

Color Kernel Example

Applying the kernel inside of a CIFilter

```
- (CIKernel *) myKernel {
    static CIColorKernel *kernel = nil;
    static dispatch_once_t once;
    dispatch_once(&once, ^{
          kernel = [CIColorKernel kernelWithString:
          @“kernel vec4 swapRedAndGreenAmount ( __sample s, float amount )
                              { return mix(s.rgba, s.grba, amount); }” ]; });
    return kernel;
}
- (CIImage *)outputImage
{
    CGRect dod = inputImage.extent ;
    return [ [self myKernel] applyWithExtent : dod
                             arguments : @[inputImage, inputAmount]];
}
```

Color Kernel Example

Applying the kernel inside of a CIFilter

```
- (CIKernel *) myKernel {
    static CIColorKernel *kernel = nil;
    static dispatch_once_t once;
    dispatch_once(&once, ^{
          kernel = [CIColorKernel kernelWithString:
          @“kernel vec4 swapRedAndGreenAmount ( __sample s, float amount )
                              { return mix(s.rgba, s.grba, amount); }” ]; });
    return kernel;
}
- (CIImage *)outputImage
{
    CGRect dod = inputImage.extent ;
    return [ [self myKernel] applyWithExtent : dod
                             arguments : @[inputImage, inputAmount]];
}
```

Color Kernel Example

Applying the kernel inside of a CIFilter

```
- (CIKernel *) myKernel {
    static CIColorKernel *kernel = nil;
    static dispatch_once_t once;
    dispatch_once(&once, ^{
          kernel = [CIColorKernel kernelWithString:
          @“kernel vec4 swapRedAndGreenAmount ( __sample s, float amount )
                              { return mix(s.rgba, s.grba, amount); }” ]; });
    return kernel;
}
- (CIImage *)outputImage
{
    CGRect dod = inputImage.extent ;
    return [ [self myKernel] applyWithExtent : dod
                             arguments : @[inputImage, inputAmount]];
}
```

Color Kernel Which Depends on Position

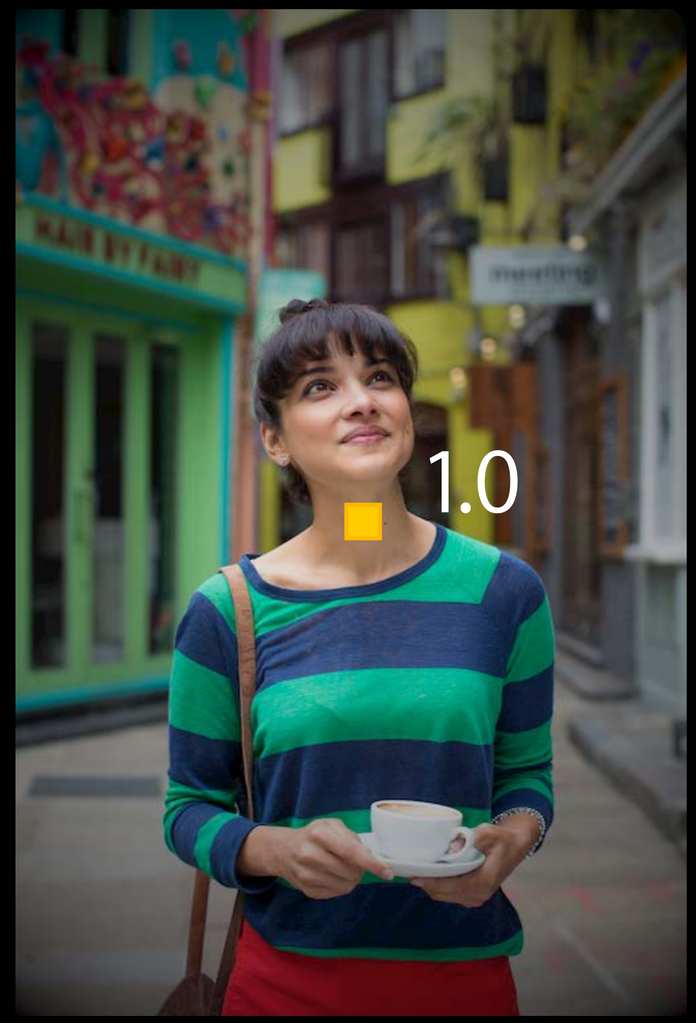
Vignette effect

Color Kernel Which Depends on Position

Vignette effect

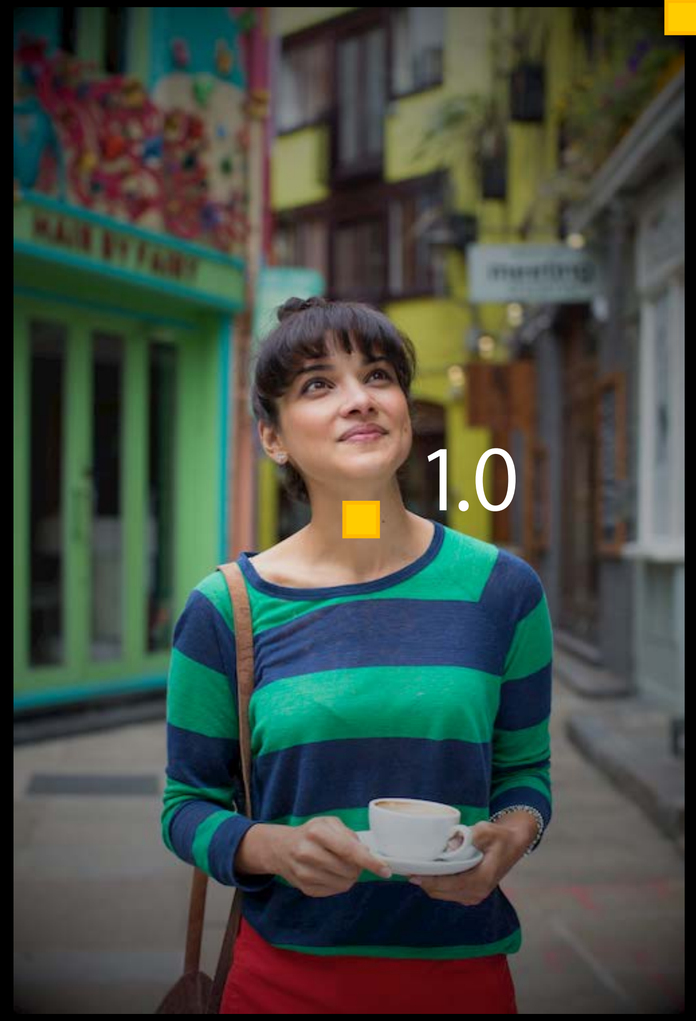
Color Kernel Which Depends on Position

Vignette effect



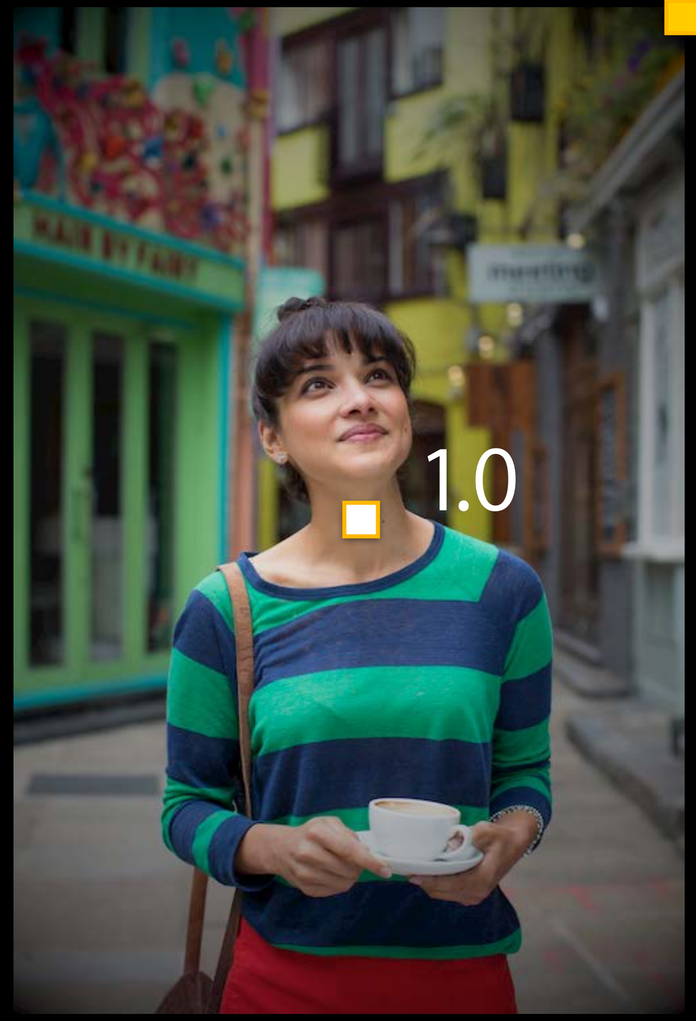
Color Kernel Which Depends on Position

Vignette effect



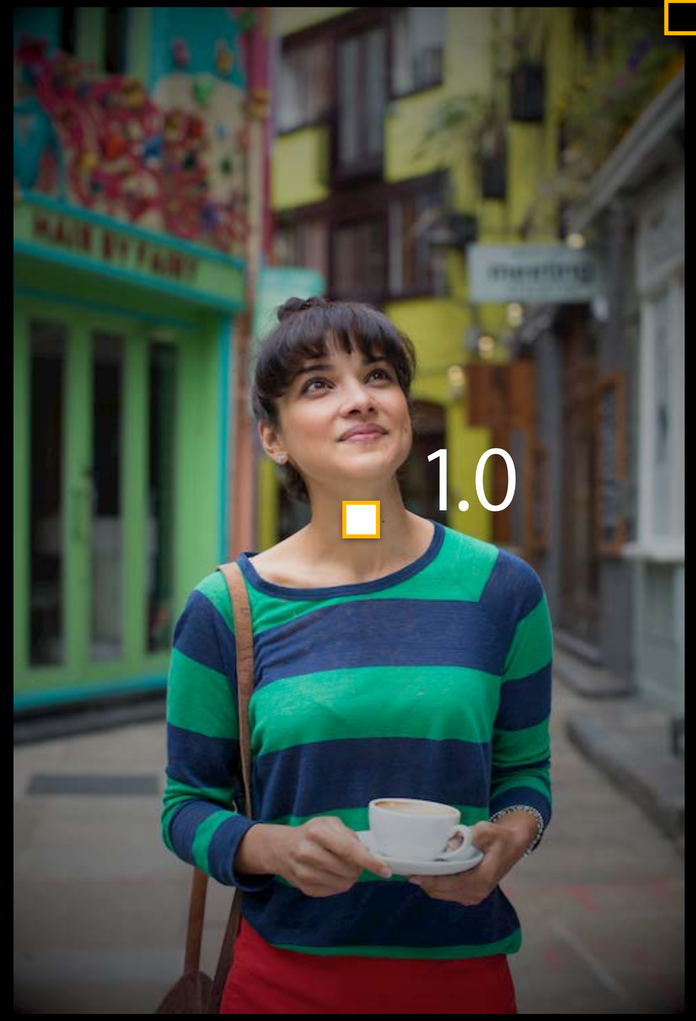
Color Kernel Which Depends on Position

Vignette effect



Color Kernel Which Depends on Position

Vignette effect

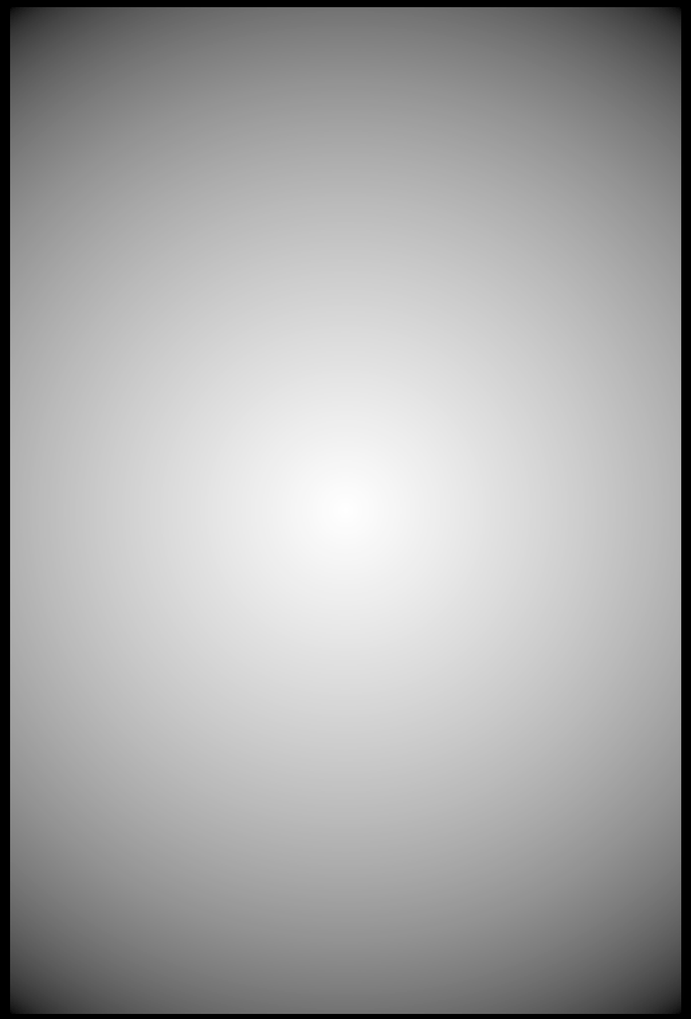


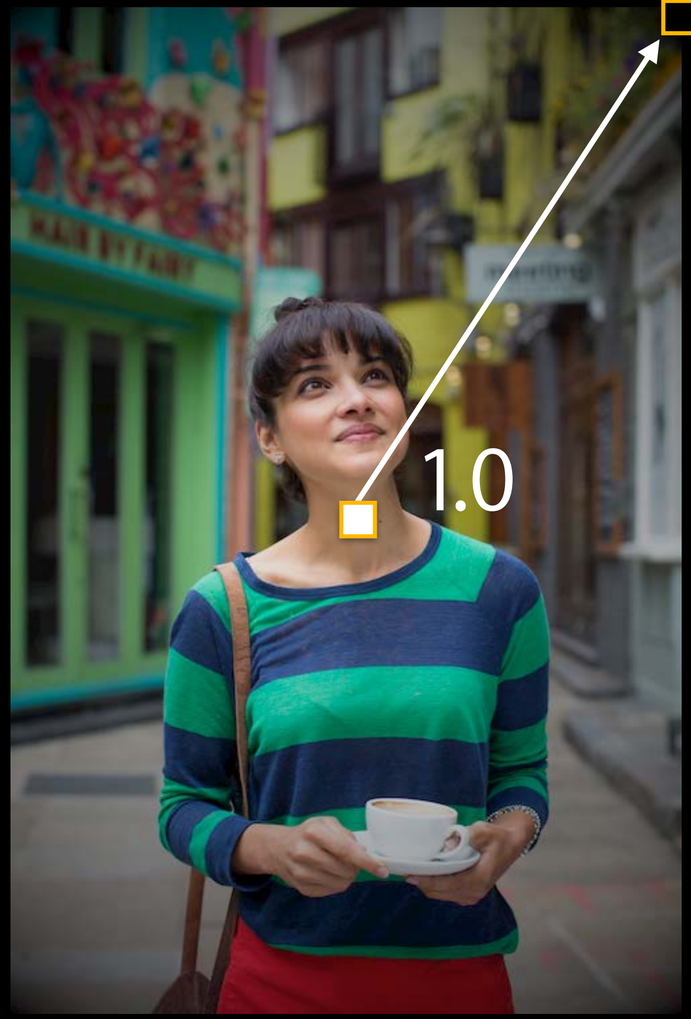
Color Kernel Which Depends on Position

Vignette effect

Color Kernel Which Depends on Position

Vignette effect

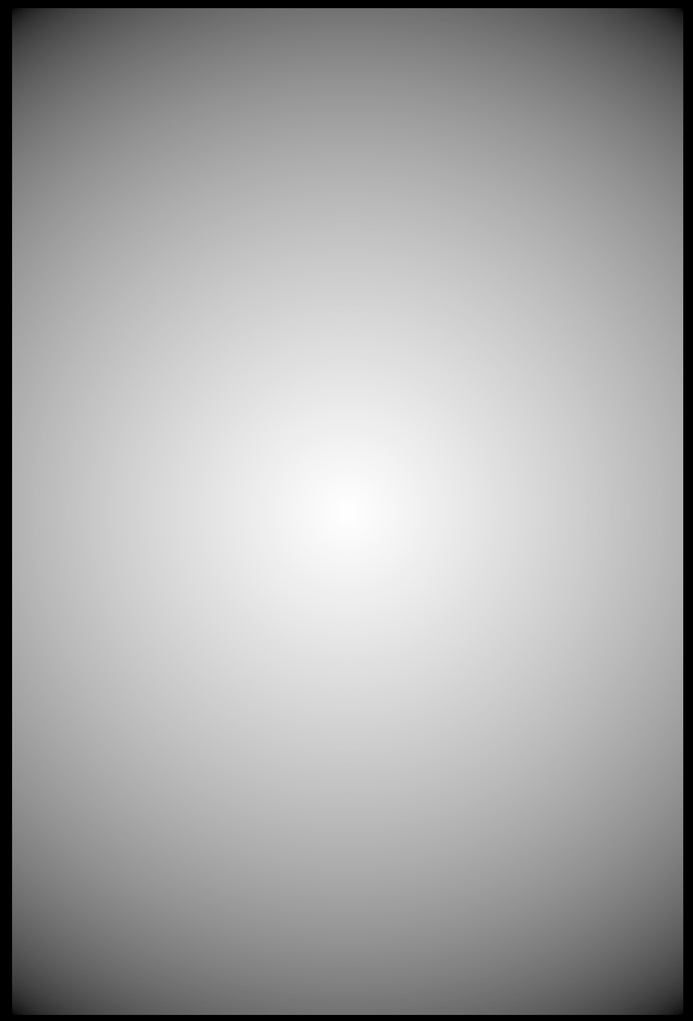




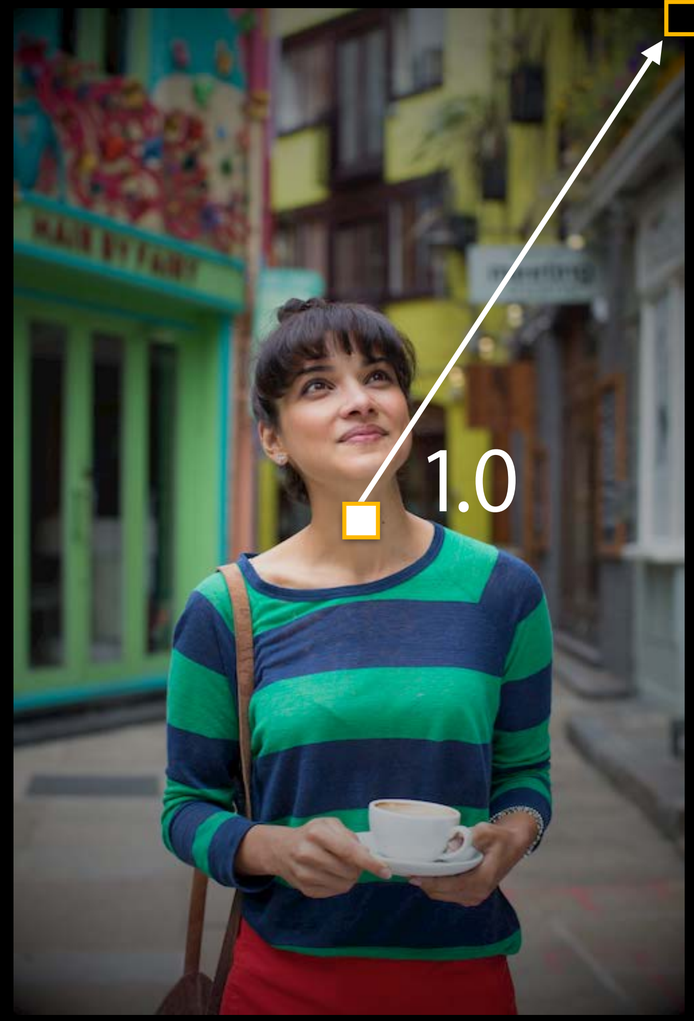
Color Kernel Which Depends on Position

Vignette effect

x



=



Color Kernel Which Depends on Position

Vignette kernel code

```
kernel vec4 vignette ( __sample s ,
                       vec2 centerOffset ,
                       float radius )

{




}
```

Color Kernel Which Depends on Position

Vignette kernel code

```
kernel vec4 vignette ( __sample s ,
                       vec2 centerOffset ,
                       float radius )

{




}
```

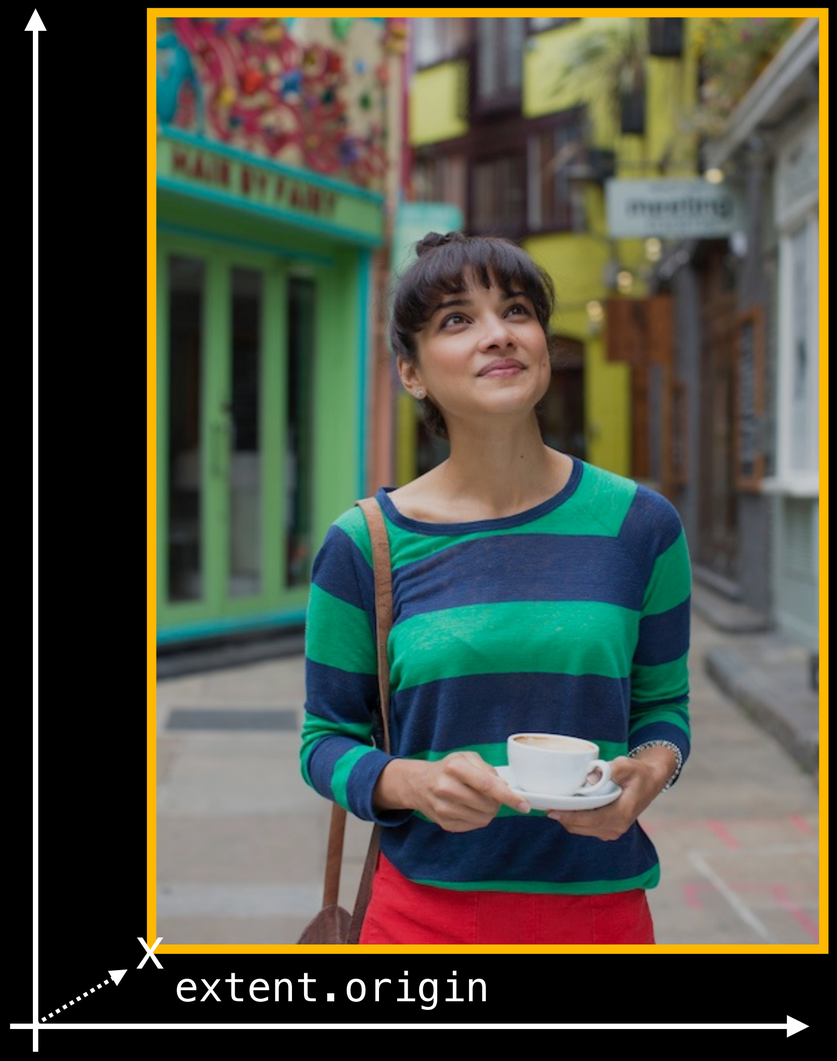
Color Kernel Which Depends on Position

Vignette kernel code


```
kernel vec4 vignette ( __sample s ,
                       vec2 centerOffset ,
                       float radius )

{




}
```

Color Kernel Which Depends on Position

Vignette kernel code




```
kernel vec4 vignette ( __sample s ,
                       vec2 centerOffset ,
                       float radius )
{




}
```

Color Kernel Which Depends on Position

Vignette kernel code


```
kernel vec4 vignette ( __sample s ,
                       vec2 centerOffset ,
                       float radius )

{




}
```

Color Kernel Which Depends on Position

Vignette kernel code


```
kernel vec4 vignette ( __sample s ,
                       vec2 centerOffset ,
                       float radius )
{




}
```

Color Kernel Which Depends on Position

Vignette kernel code


```
kernel vec4 vignette ( __sample s ,
                       vec2 centerOffset ,
                       float radius )
{



}
```

Color Kernel Which Depends on Position

Vignette kernel code


```
centerOffset = extent.origin + extent.size/2.0
```

```
kernel vec4 vignette ( __sample s ,
                       vec2 centerOffset ,
                       float radius )
{

}
```

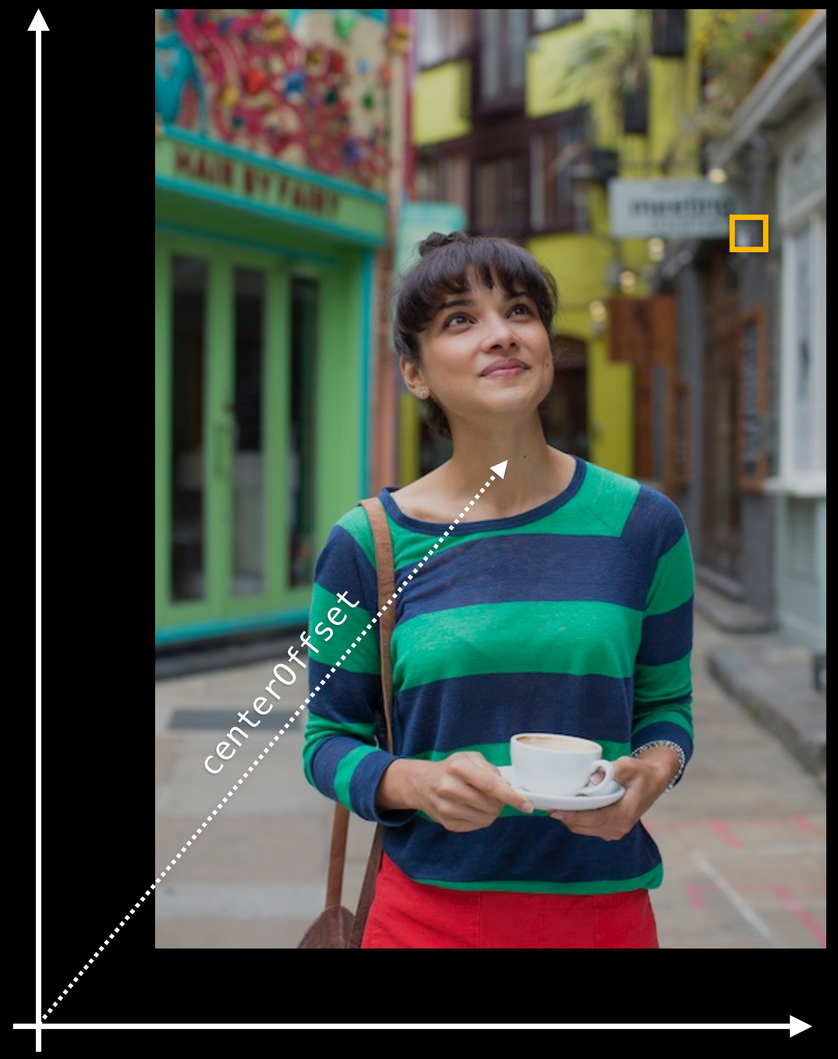
Color Kernel Which Depends on Position

Vignette kernel code


```
centerOffset = extent.origin + extent.size/2.0
      radius = length ( extent.size )/2.0

kernel vec4 vignette ( __sample s ,
                       vec2 centerOffset ,
                       float radius )
{



}
```

Color Kernel Which Depends on Position

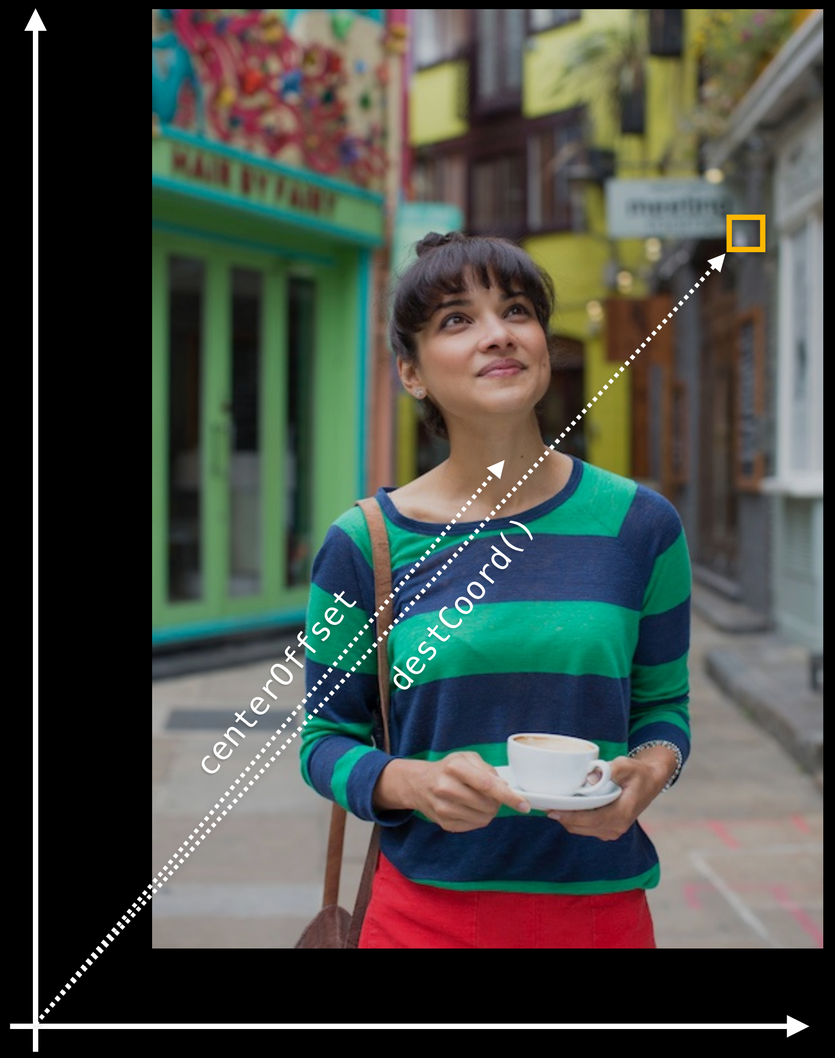
Vignette kernel code




```
centerOffset = extent.origin + extent.size/2.0
      radius = length ( extent.size )/2.0

kernel vec4 vignette ( __sample s ,
                       vec2 centerOffset ,
                       float radius )
{




}
```

Color Kernel Which Depends on Position

Vignette kernel code




```
centerOffset = extent.origin + extent.size/2.0
      radius = length ( extent.size )/2.0

kernel vec4 vignette ( __sample s ,
                       vec2 centerOffset ,
                       float radius )
{




}
```

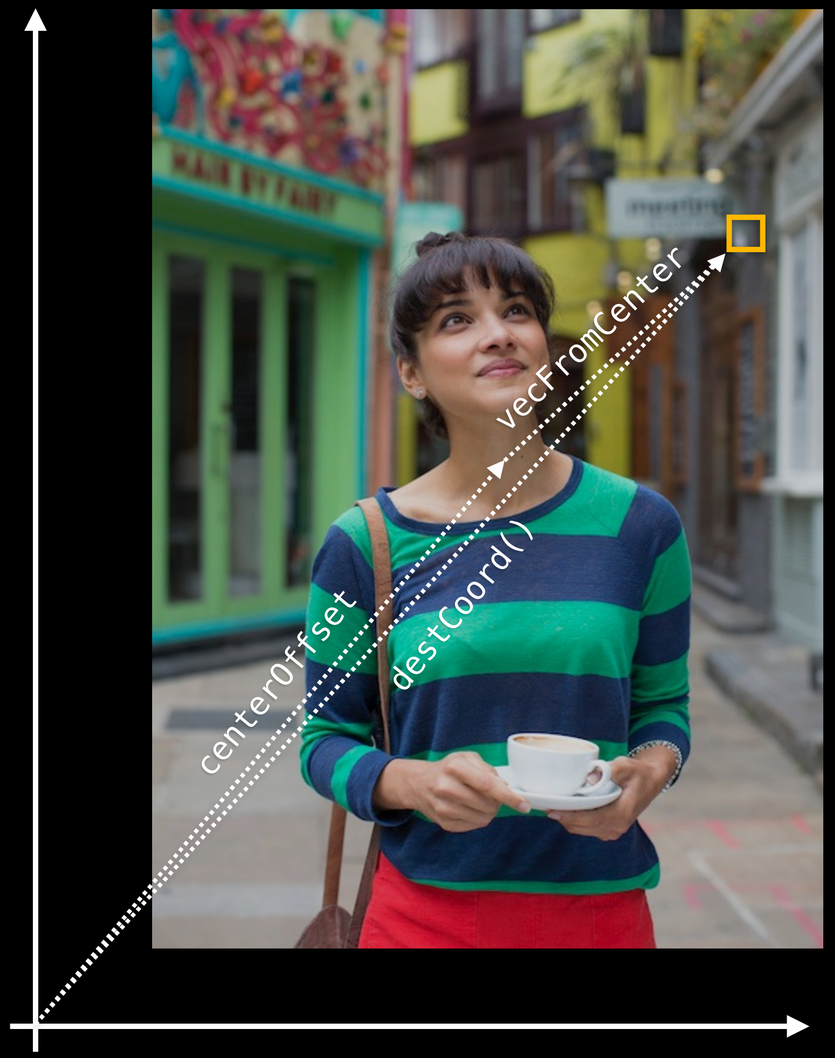
Color Kernel Which Depends on Position

Vignette kernel code


```
centerOffset = extent.origin + extent.size/2.0
      radius = length ( extent.size )/2.0

kernel vec4 vignette ( __sample s ,
                       vec2 centerOffset ,
                       float radius )
{



}
```

Color Kernel Which Depends on Position

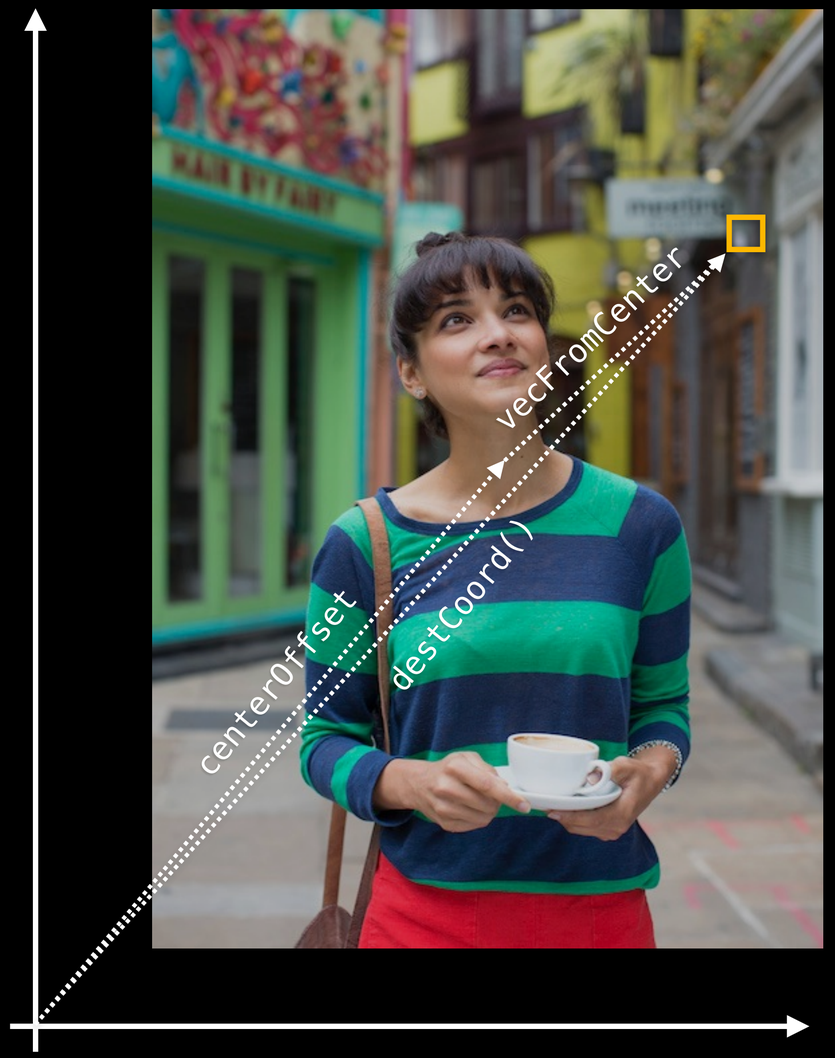
Vignette kernel code




```
centerOffset = extent.origin + extent.size/2.0
      radius = length ( extent.size )/2.0

kernel vec4 vignette ( __sample s ,
                       vec2 centerOffset ,
                       float radius )
{
  vec2 vecFromCenter =
                    destCoord () - centerOffset;



}
```

Color Kernel Which Depends on Position

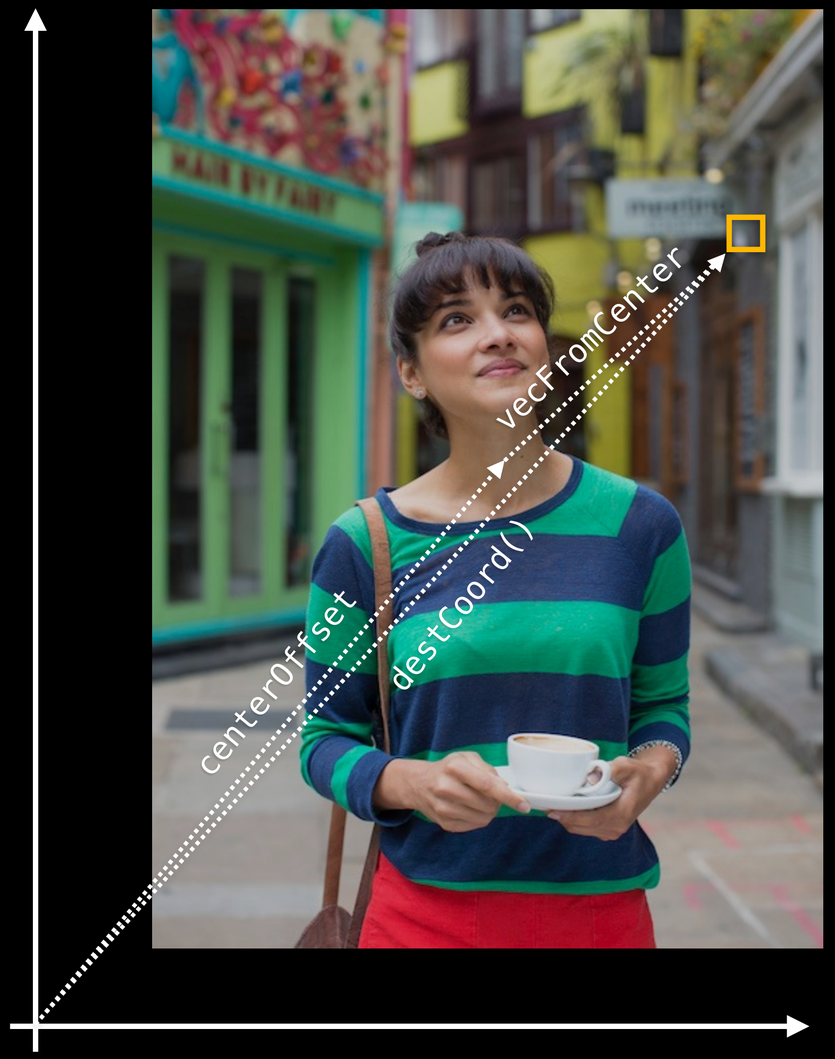
Vignette kernel code




```
centerOffset = extent.origin + extent.size/2.0
      radius = length ( extent.size )/2.0

kernel vec4 vignette ( __sample s ,
                       vec2 centerOffset ,
                       float radius )
{
  vec2 vecFromCenter =
                  destCoord () - centerOffset;
  float distance  = length ( vecFromCenter ) ;


}
```

Color Kernel Which Depends on Position

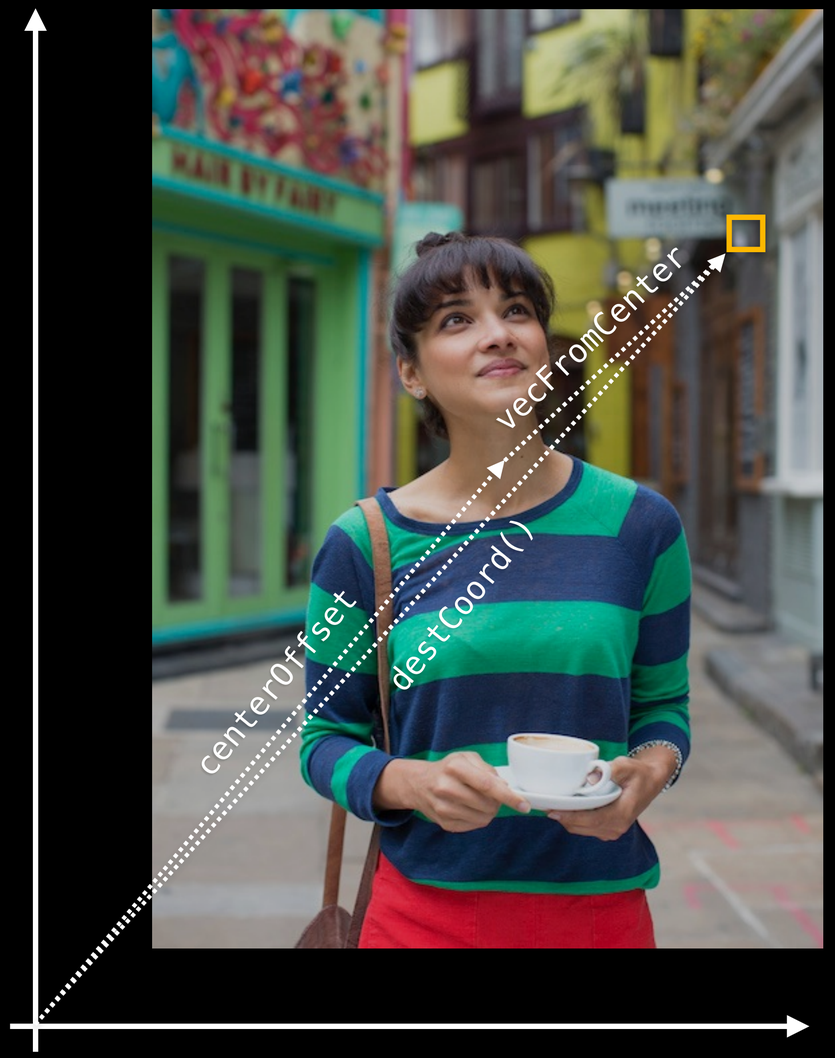
Vignette kernel code




```
centerOffset = extent.origin + extent.size/2.0
      radius = length ( extent.size )/2.0

kernel vec4 vignette ( __sample s ,
                       vec2 centerOffset ,
                       float radius )
{
  vec2 vecFromCenter =
                  destCoord () - centerOffset;
  float distance  = length ( vecFromCenter ) ;
  float darken = 1.0 - ( distance / radius ) ;

}
```

Color Kernel Which Depends on Position

Vignette kernel code




```
centerOffset = extent.origin + extent.size/2.0
      radius = length ( extent.size )/2.0

kernel vec4 vignette ( __sample s ,
                       vec2 centerOffset ,
                       float radius )
{
  vec2 vecFromCenter =
                  destCoord () - centerOffset;
  float distance  = length ( vecFromCenter ) ;
  float darken = 1.0 - ( distance / radius ) ;
  return vec4 ( s.rgb * darken, s.a ) ;
}
```

Color Kernel Which Depends on Position

Vignette outputImage method

```
- (CIImage *)outputImage {
    CGRect dod = inputImage.extent;
    double radius = 0.5 * hypot ( dod.size.width, dod.size.height );
    CIVector *centerOffset =
         [CIVector vectorWithX : dod.size.width * 0.5  + dod.origin.x
                             Y : dod.size.height * 0.5 + dod.origin.y];

    return [[self myKernel] applyWithExtent : dod
            arguments : @[inputImage, centerOffset, @(radius)]];
}
```

Color Kernel Which Depends on Position

Vignette outputImage method

```
- (CIImage *)outputImage {
    CGRect dod = inputImage.extent;
    double radius = 0.5 * hypot ( dod.size.width, dod.size.height );
    CIVector *centerOffset =
         [CIVector vectorWithX : dod.size.width * 0.5  + dod.origin.x
                             Y : dod.size.height * 0.5 + dod.origin.y];

    return [[self myKernel] applyWithExtent : dod
            arguments : @[inputImage, centerOffset, @(radius)]];
}
```

Color Kernel Which Depends on Position

Vignette outputImage method

```
- (CIImage *)outputImage {
    CGRect dod = inputImage.extent;
    double radius = 0.5 * hypot ( dod.size.width, dod.size.height );
    CIVector *centerOffset =
         [CIVector vectorWithX : dod.size.width * 0.5  + dod.origin.x
                             Y : dod.size.height * 0.5 + dod.origin.y];

    return [[self myKernel] applyWithExtent : dod
            arguments : @[inputImage, centerOffset, @(radius)]];
}
```

Color Kernel Which Depends on Position

Vignette outputImage method

```
- (CIImage *)outputImage {
    CGRect dod = inputImage.extent;
    double radius = 0.5 * hypot ( dod.size.width, dod.size.height );
    CIVector *centerOffset =
          [CIVector vectorWithX : dod.size.width * 0.5  + dod.origin.x
                              Y : dod.size.height * 0.5 + dod.origin.y];

    return [[self myKernel] applyWithExtent : dod
                  arguments : @[inputImage, centerOffset, @(radius)]];
}
```

Color Kernel Which Depends on Position

Vignette outputImage method

```
- (CIImage *)outputImage {
    CGRect dod = inputImage.extent;
    double radius = 0.5 * hypot ( dod.size.width, dod.size.height );
    CIVector *centerOffset =
          [CIVector vectorWithX : dod.size.width * 0.5  + dod.origin.x
                              Y : dod.size.height * 0.5 + dod.origin.y];

    return [[self myKernel] applyWithExtent : dod
                                  arguments : @[inputImage, centerOffset, @(radius)]];
}
```

Color Kernel Which Depends on Position

Vignette outputImage method

```
- (CIImage *)outputImage {
    CGRect dod = inputImage.extent;
    double radius = 0.5 * hypot ( dod.size.width, dod.size.height );
    CIVector *centerOffset =
         [CIVector vectorWithX : dod.size.width * 0.5  + dod.origin.x
                             Y : dod.size.height * 0.5 + dod.origin.y];

    return [[self myKernel] applyWithExtent : dod
            arguments : @[inputImage, centerOffset, @(radius)]];
}
```

```
kernel vec4 vignette ( __sample s ,vec2 centerOffset ,float radius){ … }
```

Color Kernel Which Depends on Position

Vignette outputImage method

```
- (CIImage *)outputImage {
    CGRect dod = inputImage.extent;
    double radius = 0.5 * hypot ( dod.size.width, dod.size.height );
    CIVector *centerOffset =
         [CIVector vectorWithX : dod.size.width * 0.5  + dod.origin.x
                             Y : dod.size.height * 0.5 + dod.origin.y];

    return [[self myKernel] applyWithExtent : dod
             arguments : @[inputImage, centerOffset, @(radius)]];
}

kernel vec4 vignette ( __sample s ,vec2 centerOffset ,float radius){ … }
```

Color Kernel Which Depends on Position

Vignette outputImage method

```
- (CIImage *)outputImage {
    CGRect dod = inputImage.extent;
    double radius = 0.5 * hypot ( dod.size.width, dod.size.height );
    CIVector *centerOffset =
         [CIVector vectorWithX : dod.size.width * 0.5  + dod.origin.x
                             Y : dod.size.height * 0.5 + dod.origin.y];

    return [[self myKernel] applyWithExtent : dod
            arguments : @[inputImage, centerOffset, @(radius)]];
}

kernel vec4 vignette ( __sample s ,vec2 centerOffset ,float radius){ … }
```

Color Kernel Which Depends on Position

Vignette outputImage method

```
- (CIImage *)outputImage {
    CGRect dod = inputImage.extent;
    double radius = 0.5 * hypot ( dod.size.width, dod.size.height );
    CIVector *centerOffset =
         [CIVector vectorWithX : dod.size.width * 0.5  + dod.origin.x
                             Y : dod.size.height * 0.5 + dod.origin.y];

    return [[self myKernel] applyWithExtent : dod
            arguments : @[inputImage, centerOffset, @(radius)]];
}

kernel vec4 vignette ( __sample s ,vec2 centerOffset ,float radius){ … }
```

DOD for Source over Compositing Kernel

Non zero pixels

```
kernel vec4 sourceOver( __sample src , __sample dest )
                        { return src + (1.0-src.a)*dest; }
```


```
CGRect dod = CGRectUnion ( inputImage.extent, inputBackgroundImage.extent )
```

DOD for Source over Compositing Kernel

Non zero pixels

```
kernel vec4 sourceOver( __sample src , __sample dest )
                        { return src + (1.0-src.a)*dest; }
```


```
CGRect dod = CGRectUnion ( inputImage.extent, inputBackgroundImage.extent )
```

Warp Kernels

CIWarpKernel

Basic Principles of Warp Kernels

CIWarpKernel

Specify DOD and ROI block callback

Basic Principles of Warp Kernels

CIWarpKernel—think backwards

```
kernel vec2 doNothing () { return destCoord(); }
```

Basic Principles of Warp Kernels

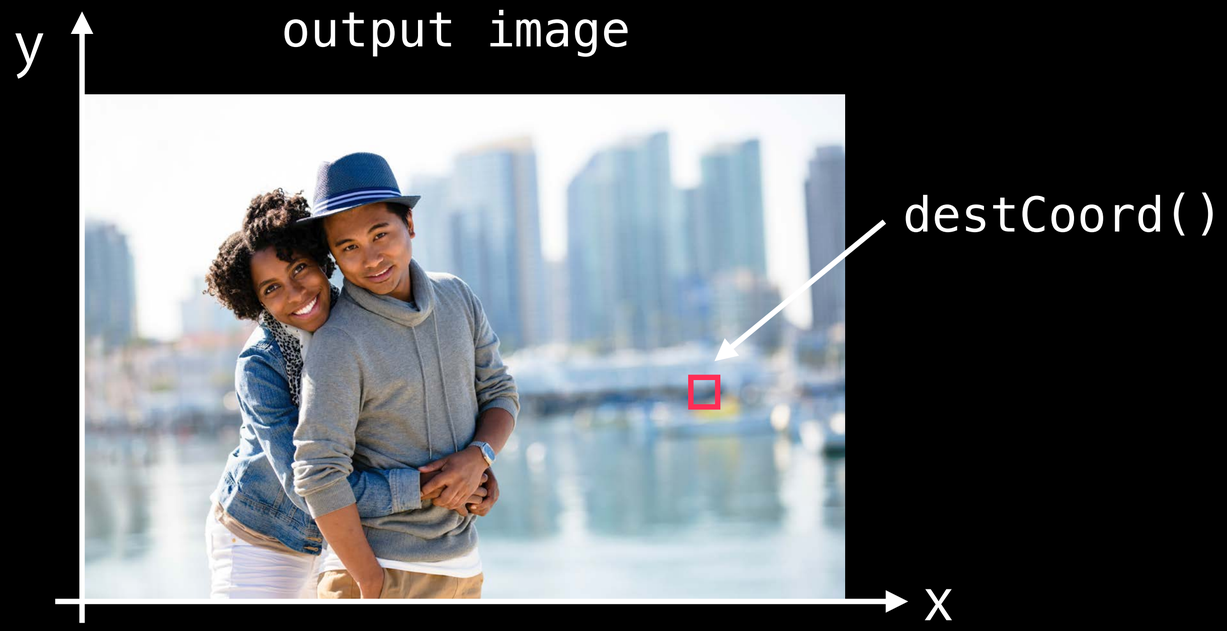
CIWarpKernel—think backwards

```
kernel vec2 doNothing () { return destCoord(); }
```

Basic Principles of Warp Kernels

CIWarpKernel—think backwards

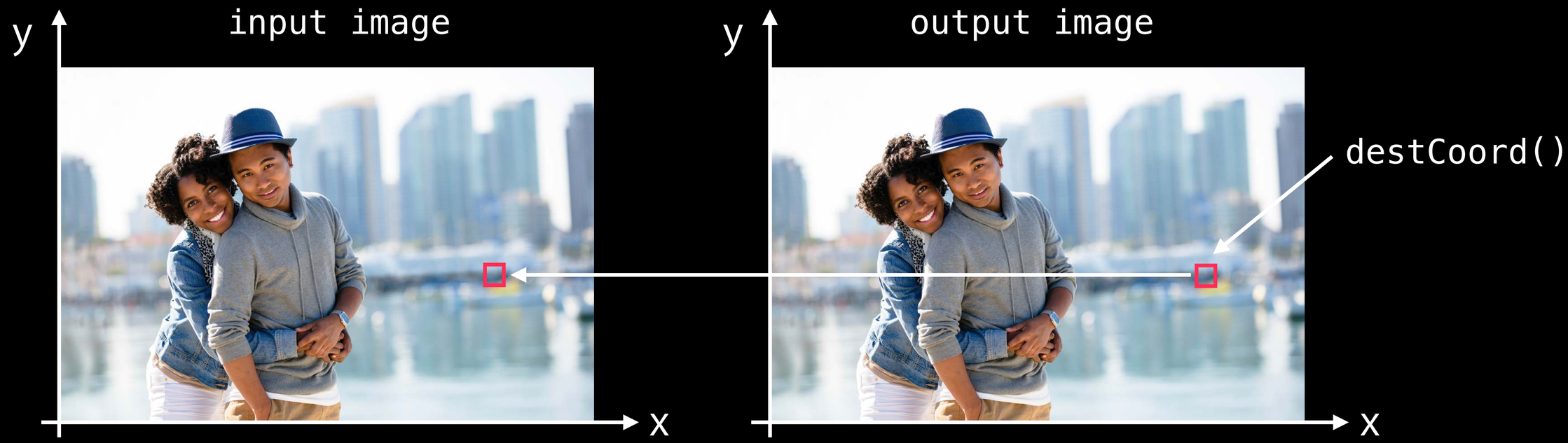
```
kernel vec2 doNothing () { return destCoord(); }
```



Basic Principles of Warp Kernels

CIWarpKernel—think backwards

```
kernel vec2 doNothing () { return destCoord(); }
```



Basic Principles of Warp Kernels

CIWarpKernel—think backwards

```
kernel vec2 mirrorX ( float imageWidth ) {
  vec2 input = destCoord();
  return vec2 ( imageWidth – input.x , input.y );
}
```

Basic Principles of Warp Kernels

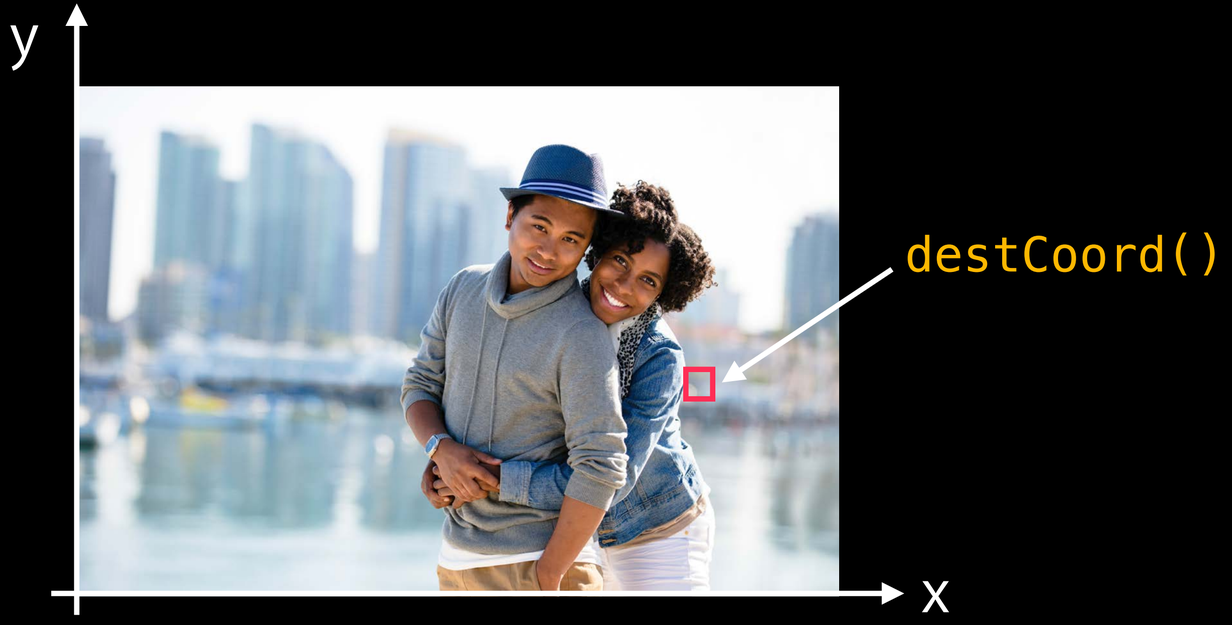
CIWarpKernel—think backwards

```
kernel vec2 mirrorX ( float imageWidth ) {
  vec2 input = destCoord();
  return vec2 ( imageWidth – input.x , input.y );
}
```

Basic Principles of Warp Kernels

CIWarpKernel—think backwards

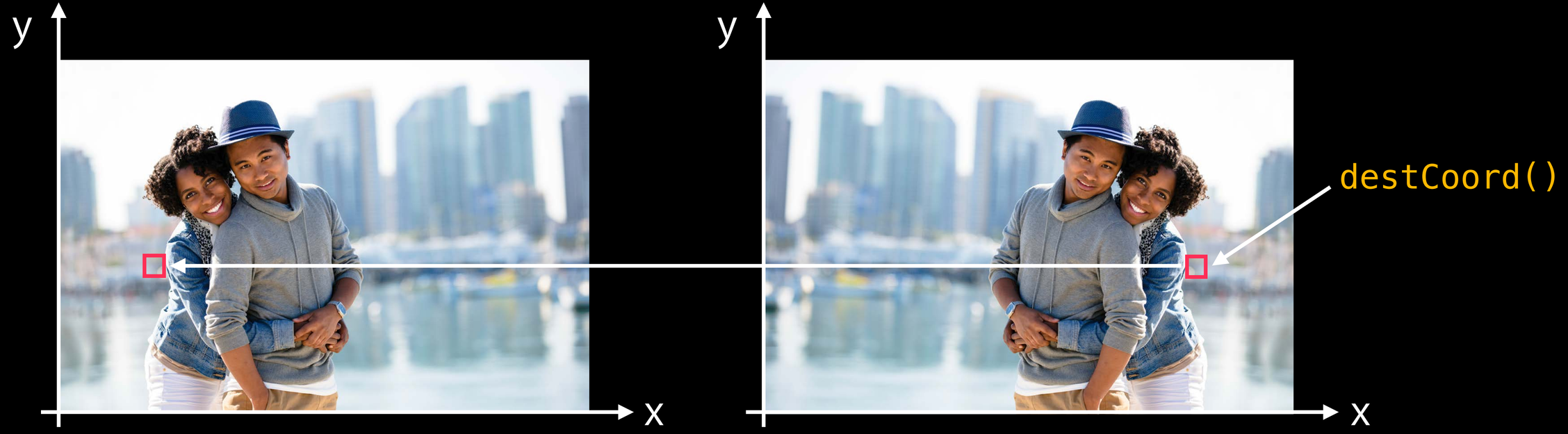
```
kernel vec2 mirrorX ( float imageWidth ) {
  vec2 input = destCoord();
  return vec2 ( imageWidth – input.x , input.y );
}
```



Basic Principles of Warp Kernels

CIWarpKernel—think backwards

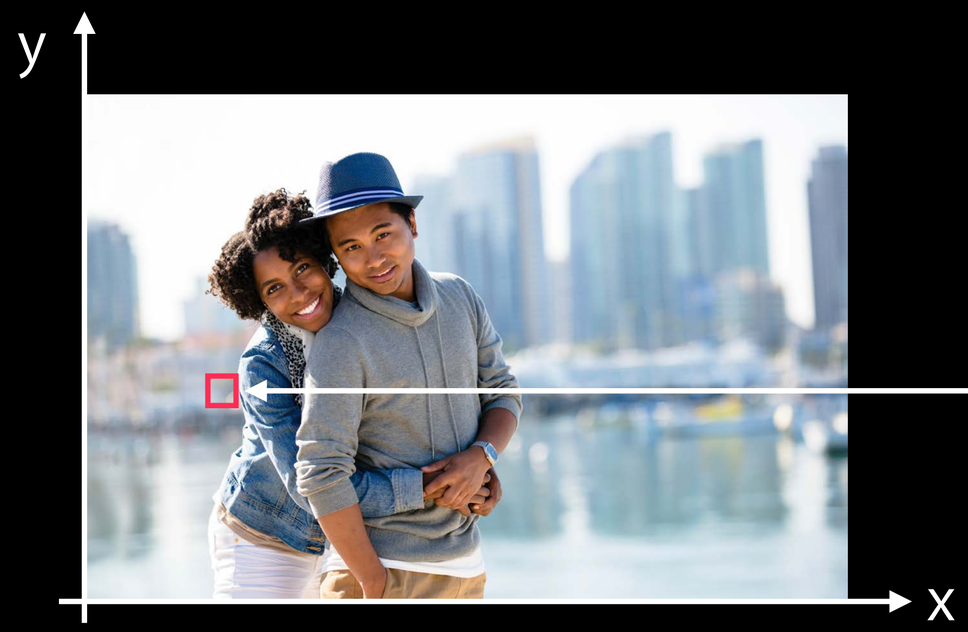
```
kernel vec2 mirrorX ( float imageWidth ) {
  vec2 input = destCoord();
  return vec2 ( imageWidth – input.x , input.y );
}
```

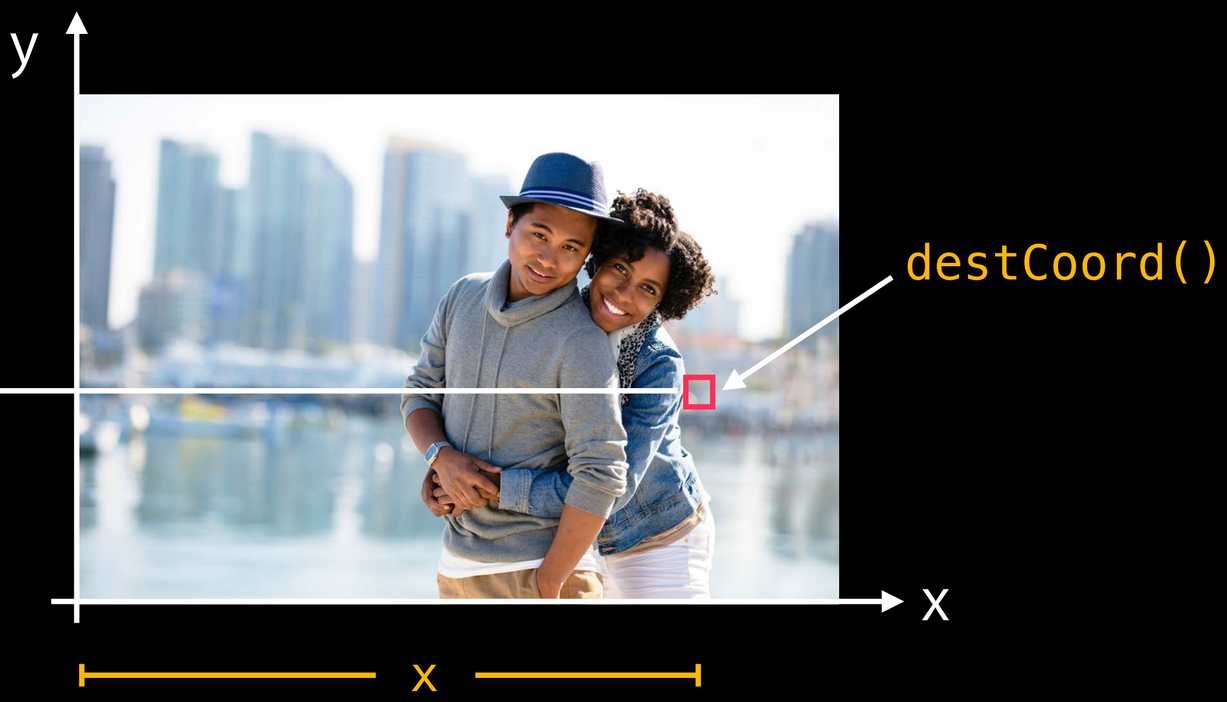


Basic Principles of Warp Kernels

CIWarpKernel—think backwards

```
kernel vec2 mirrorX ( float imageWidth ) {
  vec2 input = destCoord();
  return vec2 ( imageWidth – input.x , input.y );
}
```

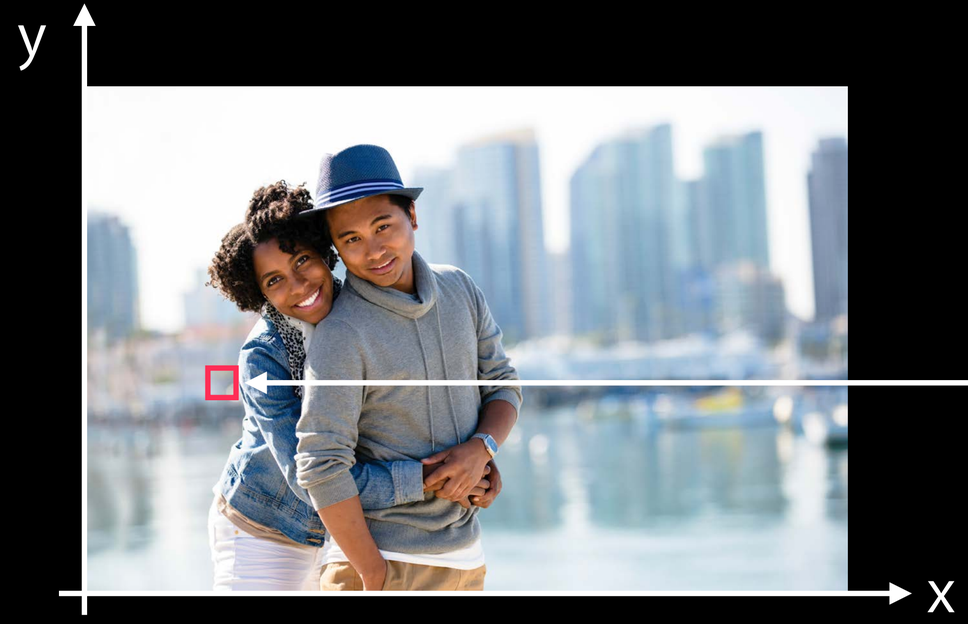


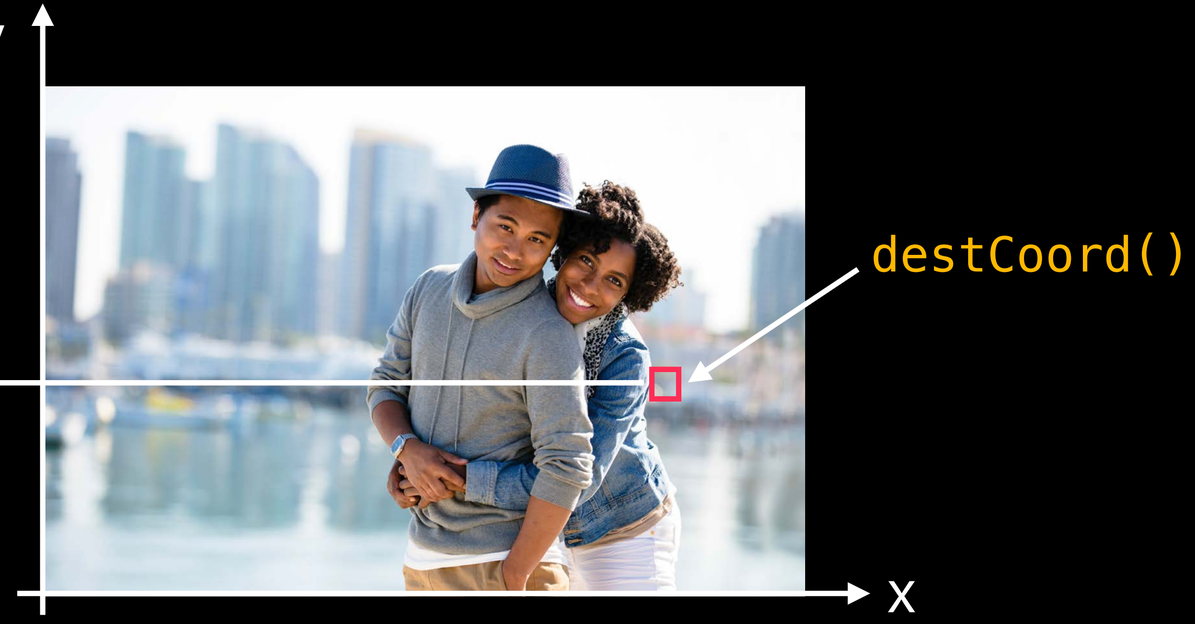


Basic Principles of Warp Kernels

CIWarpKernel—think backwards

```
kernel vec2 mirrorX ( float imageWidth ) {
  vec2 input = destCoord();
  return vec2 ( imageWidth – input.x , input.y );
}
```

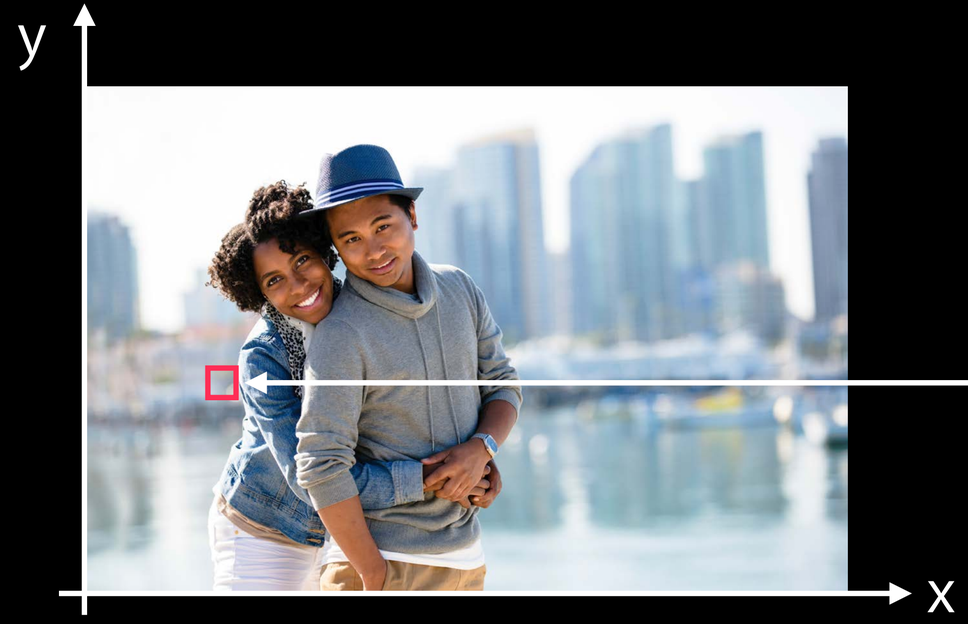


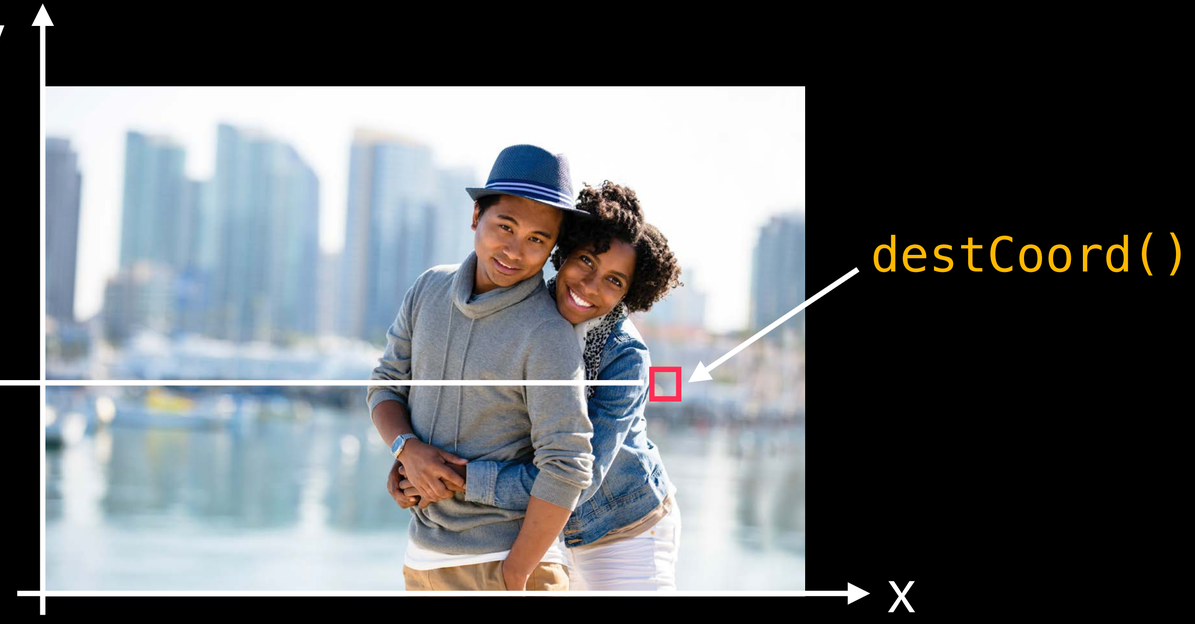


Basic Principles of Warp Kernels

CIWarpKernel—think backwards

```
kernel vec2 mirrorX ( float imageWidth ) {
  vec2 input = destCoord();
  return vec2 ( imageWidth – input.x , input.y );
}
```

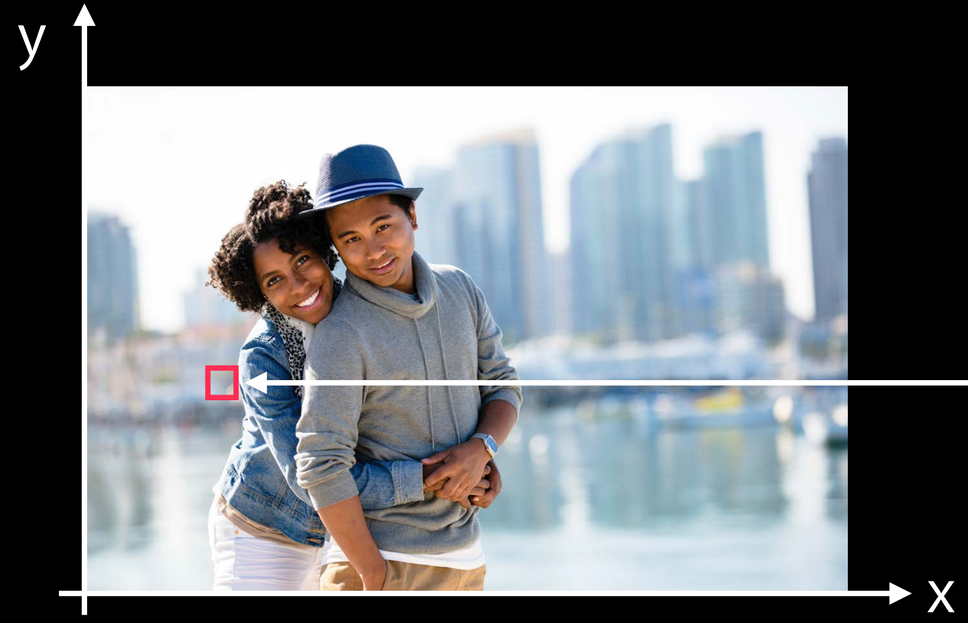


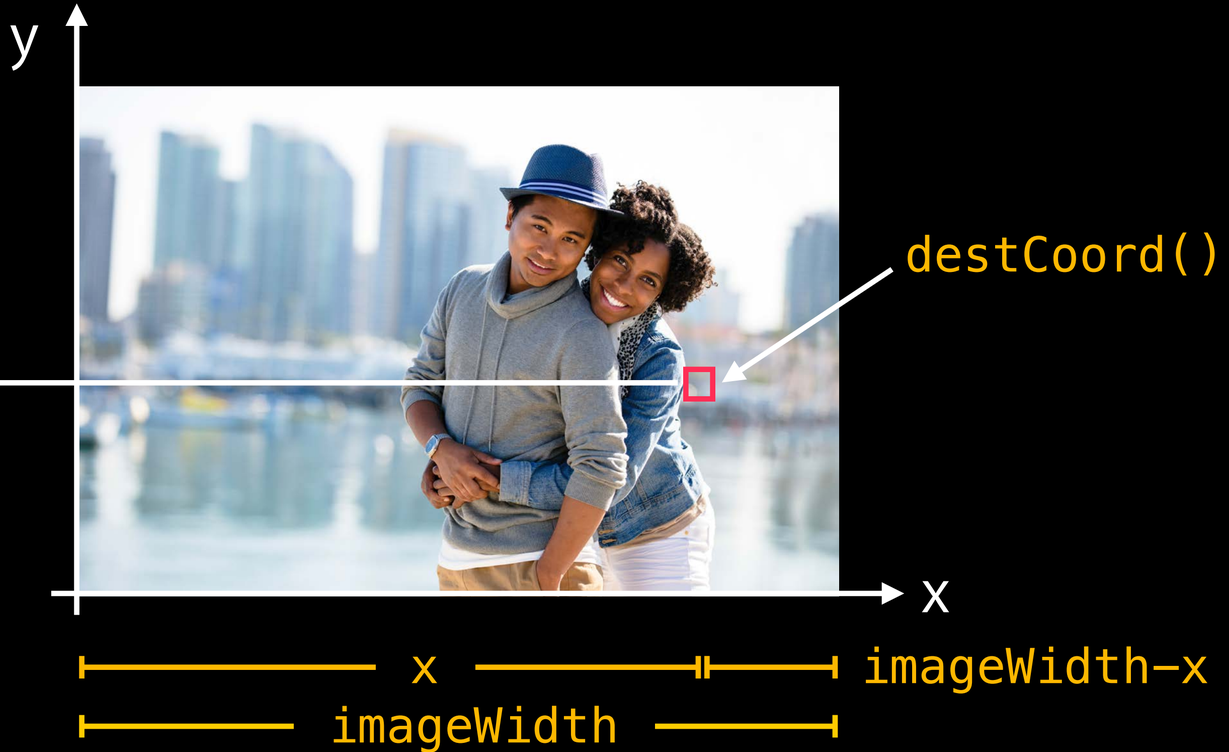


Basic Principles of Warp Kernels

CIWarpKernel—think backwards

```
kernel vec2 mirrorX ( float imageWidth ) {
  vec2 input = destCoord();
  return vec2 ( imageWidth – input.x , input.y );
}
```





Invoking MirrorX Kernel

```
CIWarpKernel *kernel = [CIWarpKernel kernelWithString:kernelSourceCode];

- (CIImage *) outputImage {
  CGFloat inputWidth = inputImage.extent.size.width;

  CIImage *result =
    [[self myKernel] applyWithExtent: inputImage.extent
                 roiCallback: ^( int index, CGRect r ) { … }
                  inputImage: inputImage
                   arguments: @[@(inputWidth)]];
  return result;
}
```

Invoking MirrorX Kernel

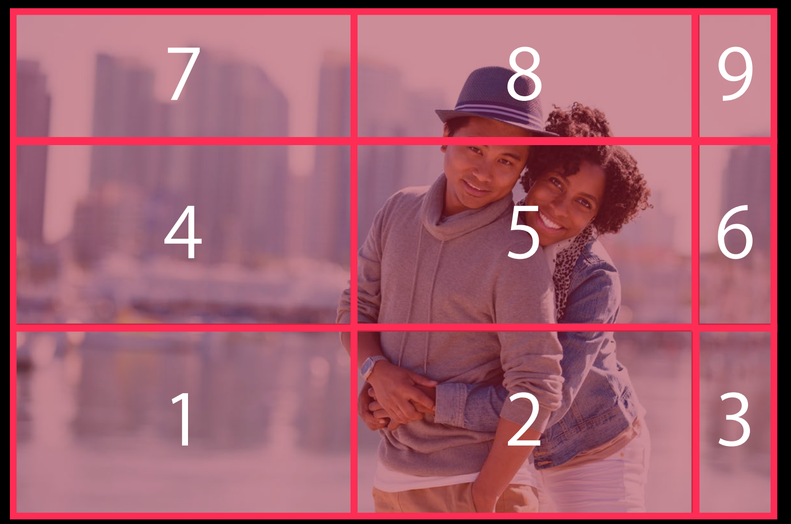
```
CIWarpKernel *kernel = [CIWarpKernel kernelWithString:kernelSourceCode];

- (CIImage *) outputImage {
  CGFloat inputWidth = inputImage.extent.size.width;

  CIImage *result =
    [[self myKernel] applyWithExtent: inputImage.extent
                 roiCallback: ^( int index, CGRect r ) { … }
                  inputImage: inputImage
                   arguments: @[@(inputWidth)]];
  return result;
}
```

Basic Principles of Warp Kernels

For output rect, what part of input rect do we need—ROI

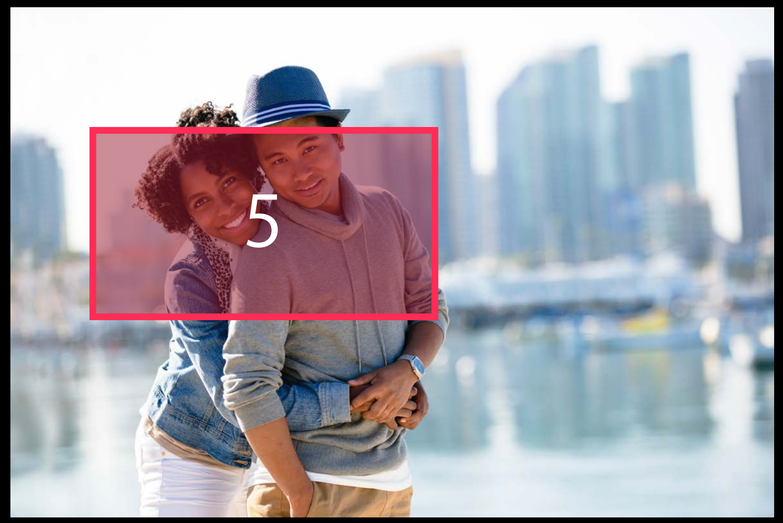
Basic Principles of Warp Kernels

For output rect, what part of input rect do we need—ROI



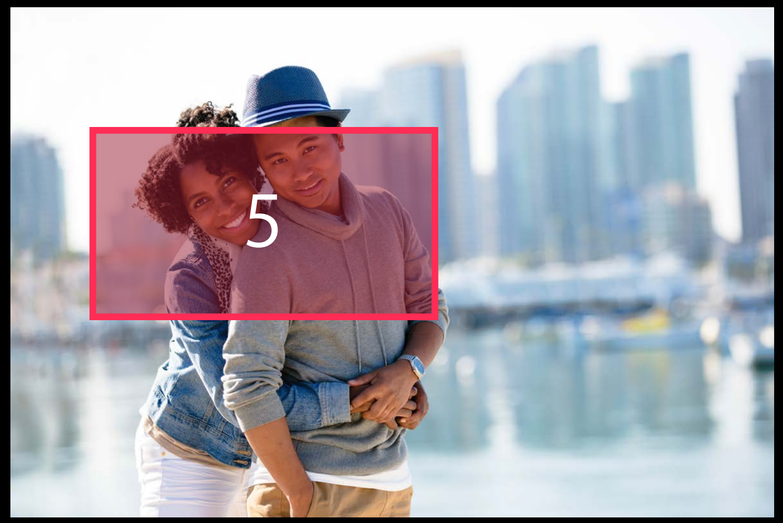
Basic Principles of Warp Kernels

For output rect, what part of input rect do we need—ROI



Basic Principles of Warp Kernels

For output rect, what part of input rect do we need—ROI




```
CGRectMake (          ,          ,          ,          );
```

Basic Principles of Warp Kernels

For output rect, what part of input rect do we need—ROI


```
CGRectMake (          ,          ,          ,          );
```

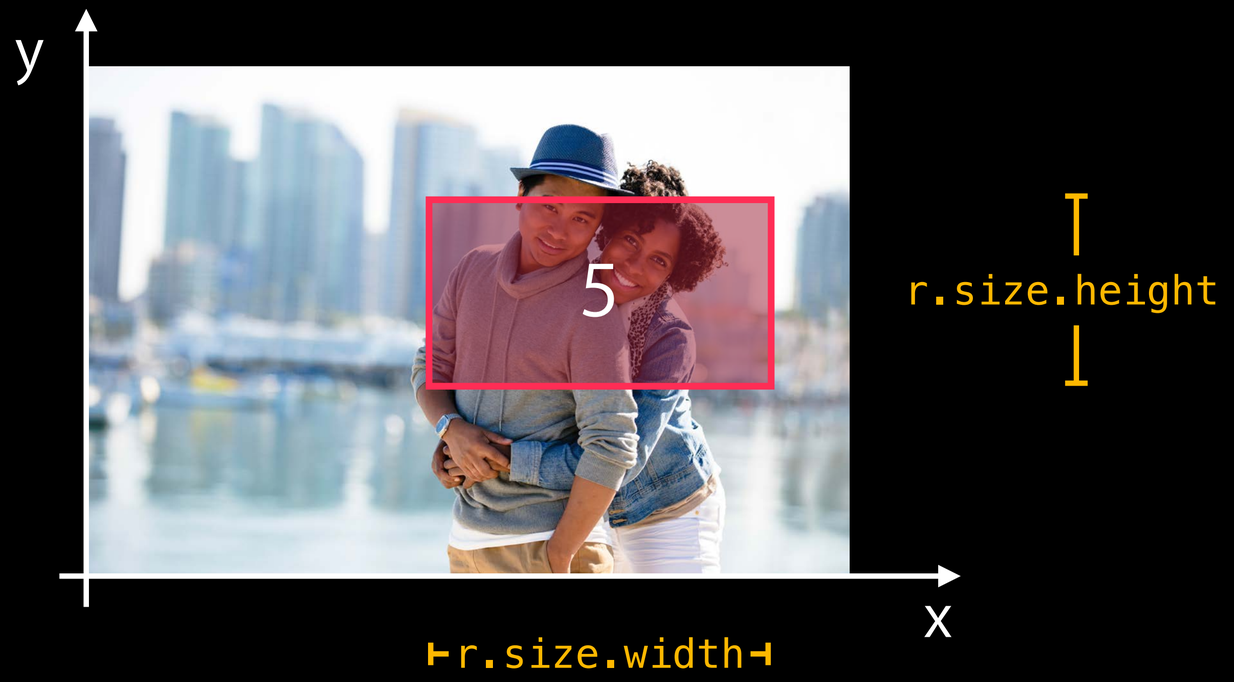
Basic Principles of Warp Kernels

For output rect, what part of input rect do we need—ROI


```
CGRectMake (          ,          ,r.size.width,                );
```

Basic Principles of Warp Kernels

For output rect, what part of input rect do we need—ROI




```
CGRectMake (          ,          ,r.size.width,r.size.height);
```

Basic Principles of Warp Kernels

For output rect, what part of input rect do we need—ROI


```
CGRectMake (          ,r.origin.y,r.size.width,r.size.height);
```

Basic Principles of Warp Kernels

For output rect, what part of input rect do we need—ROI


```
CGRectMake (newOrigin,r.origin.y,r.size.width,r.size.height);
```

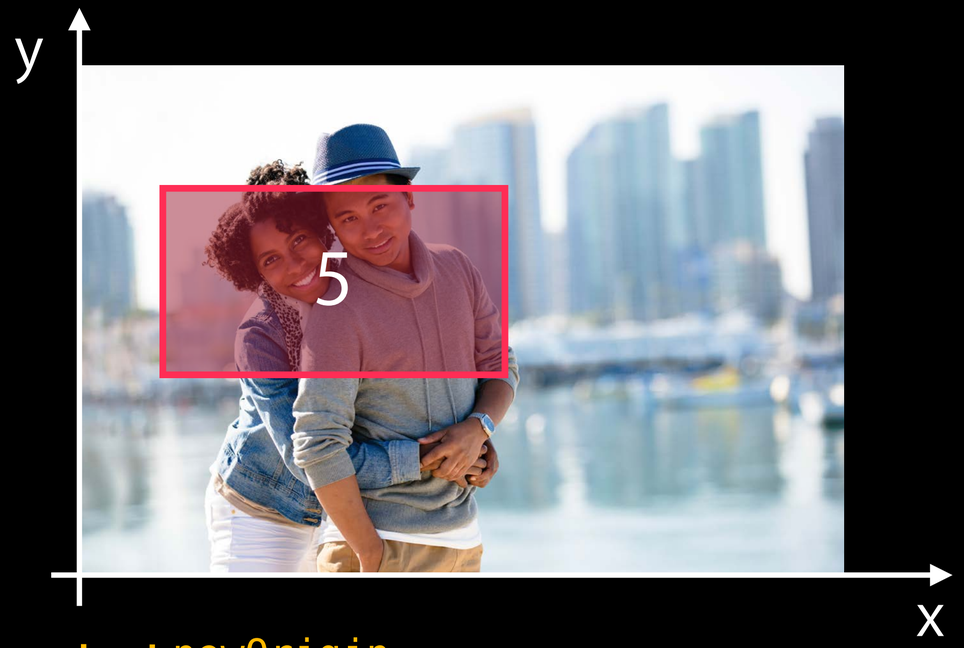
Basic Principles of Warp Kernels

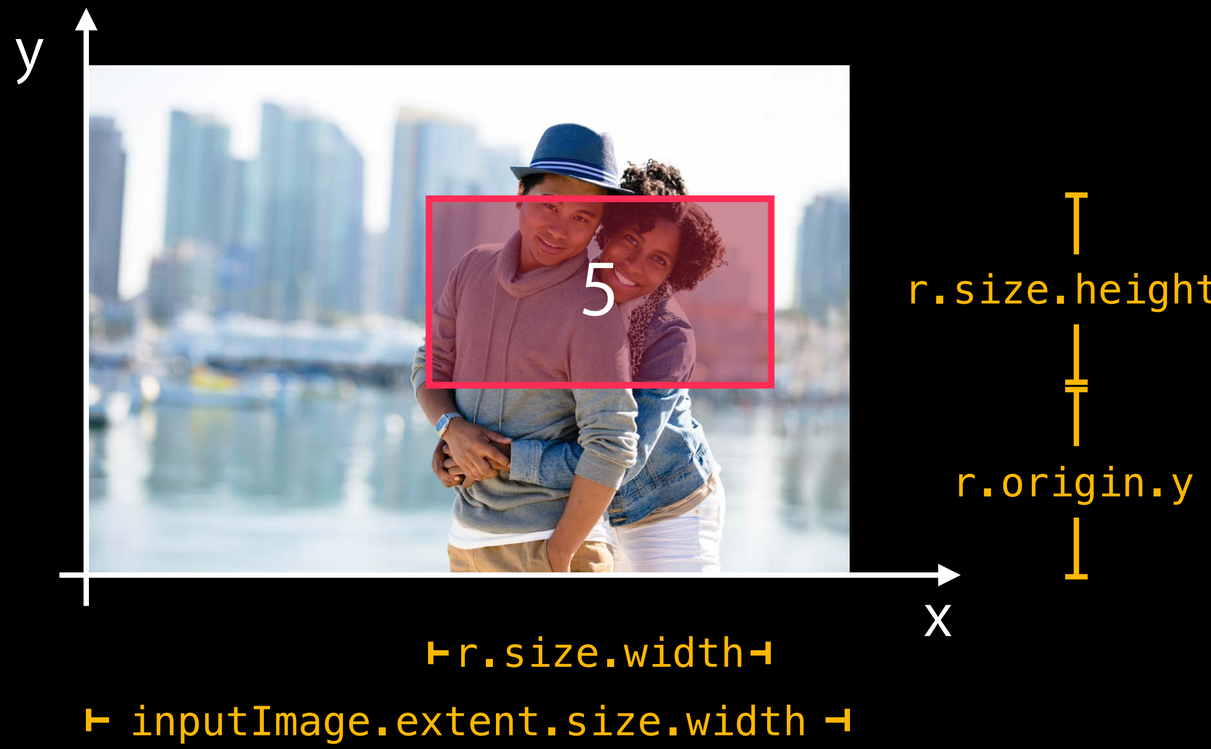
For output rect, what part of input rect do we need—ROI


```
CGRectMake (newOrigin,r.origin.y,r.size.width,r.size.height);
newOrigin =
```

Basic Principles of Warp Kernels

For output rect, what part of input rect do we need—ROI

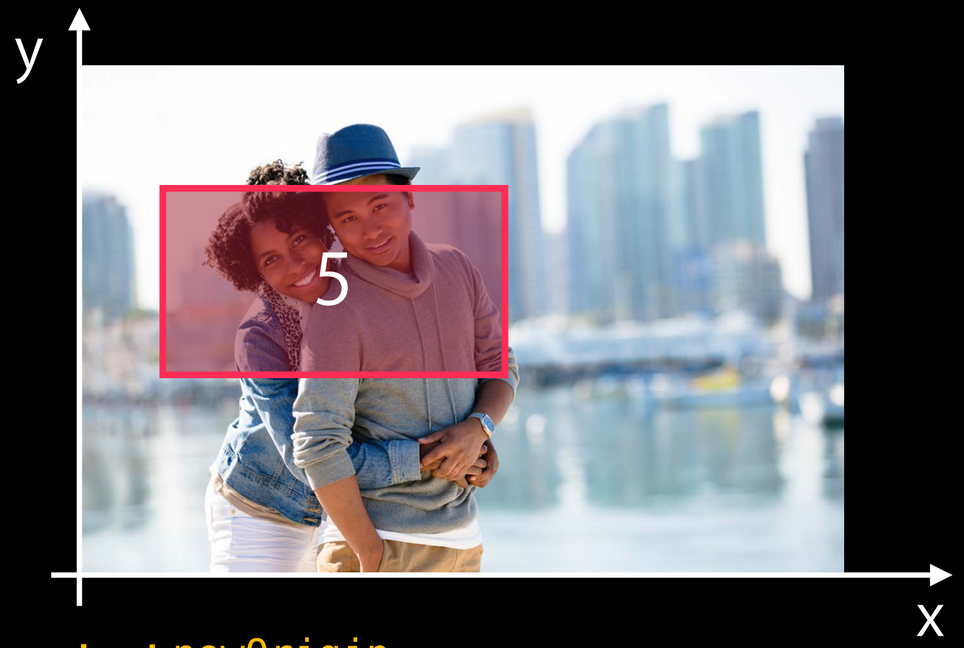


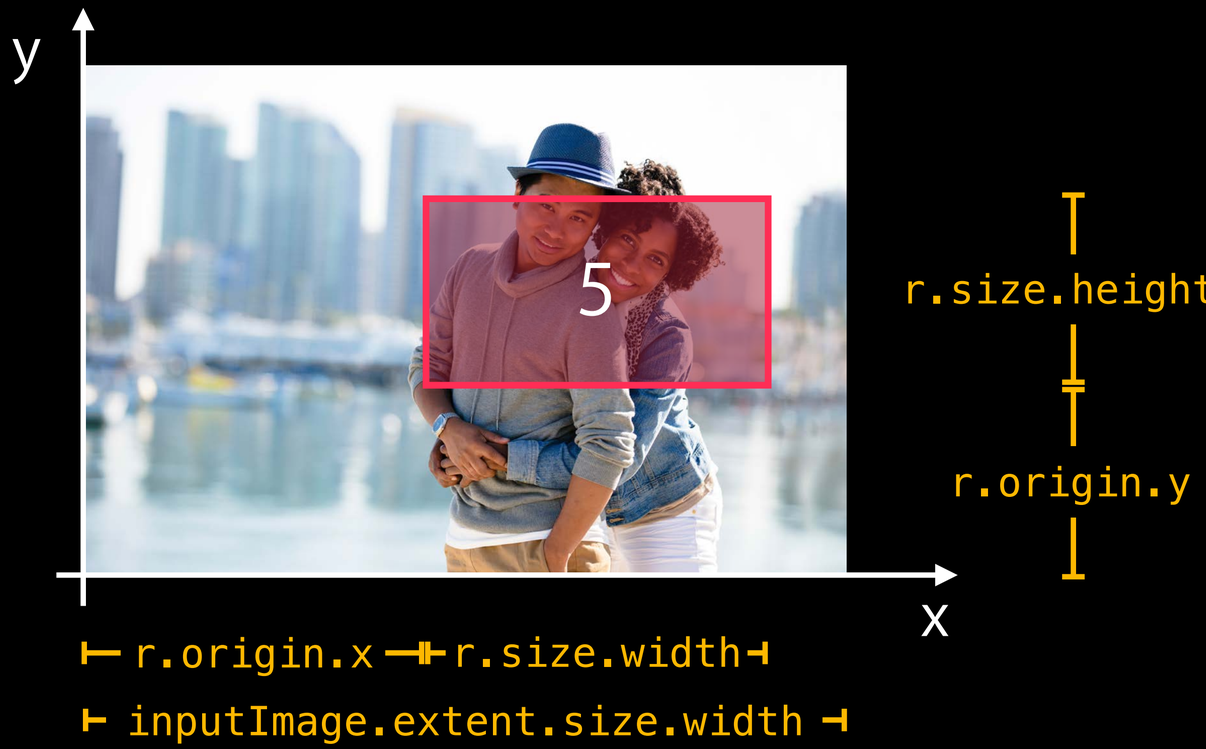



```
CGRectMake (newOrigin,r.origin.y,r.size.width,r.size.height);
newOrigin = inputImage.extent.size.width
```

Basic Principles of Warp Kernels

For output rect, what part of input rect do we need—ROI






```
CGRectMake (newOrigin,r.origin.y,r.size.width,r.size.height);
newOrigin = inputImage.extent.size.width - ( r.origin.x + r.size.width );
```

Invoking MirrorX Kernel

```
- (CIImage *) outputImage {
  if ( CGRectIsInfinite ( inputImage.extent ) ) return nil;

  CGPoint o          = inputImage.extent.origin;
  CGFloat inputWidth = inputImage.extent.size.width;

  CGAffineTransform to = CGAffineTransformMakeTranslation ( -o.x, -o.y );

  CIImage *result = [inputImage imageByApplyingTransform:to];

  result = [[self myKernel] applyWithExtent: … ]; // same as before

  CGAffineTransform from = CGAffineTransformMakeTranslation ( o.x, o.y );

  return [result imageByApplyingTransform:from];
}
```

Invoking MirrorX Kernel

```
- (CIImage *) outputImage {
  if ( CGRectIsInfinite ( inputImage.extent ) ) return nil;

  CGPoint o           = inputImage.extent.origin;
  CGFloat inputWidth = inputImage.extent.size.width;

  CGAffineTransform to = CGAffineTransformMakeTranslation ( -o.x, -o.y );

  CIImage *result = [inputImage imageByApplyingTransform:to];

  result = [[self myKernel] applyWithExtent: … ]; // same as before

  CGAffineTransform from = CGAffineTransformMakeTranslation ( o.x, o.y );

  return [result imageByApplyingTransform:from];
}
```

Invoking MirrorX Kernel

```
- (CIImage *) outputImage {
  if ( CGRectIsInfinite ( inputImage.extent ) ) return nil;

  CGPoint o           = inputImage.extent.origin;
  CGFloat inputWidth = inputImage.extent.size.width;

  CGAffineTransform to = CGAffineTransformMakeTranslation ( -o.x, -o.y );

  CIImage *result = [inputImage imageByApplyingTransform:to];

  result = [[self myKernel] applyWithExtent: … ]; // same as before

  CGAffineTransform from = CGAffineTransformMakeTranslation ( o.x, o.y );

  return [result imageByApplyingTransform:from];
}
```

Invoking MirrorX Kernel

```
- (CIImage *) outputImage {
  if ( CGRectIsInfinite ( inputImage.extent ) ) return nil;

  CGPoint o          = inputImage.extent.origin;
  CGFloat inputWidth = inputImage.extent.size.width;

  CGAffineTransform to = CGAffineTransformMakeTranslation ( -o.x, -o.y );

  CIImage *result = [inputImage imageByApplyingTransform:to];

  result = [[self myKernel] applyWithExtent: … ]; // same as before

  CGAffineTransform from = CGAffineTransformMakeTranslation ( o.x, o.y );

  return [result imageByApplyingTransform:from];
}
```

Invoking MirrorX Kernel

```
- (CIImage *) outputImage {
  if ( CGRectIsInfinite ( inputImage.extent ) ) return nil;

  CGPoint o          = inputImage.extent.origin;
  CGFloat inputWidth = inputImage.extent.size.width;

  CGAffineTransform to = CGAffineTransformMakeTranslation ( -o.x, -o.y );

  CIImage *result = [inputImage imageByApplyingTransform:to];

  result = [[self myKernel] applyWithExtent: … ]; // same as before

  CGAffineTransform from = CGAffineTransformMakeTranslation ( o.x, o.y );

  return [result imageByApplyingTransform:from];
}
```

Invoking MirrorX Kernel

```
- (CIImage *) outputImage {
  if ( CGRectIsInfinite ( inputImage.extent ) ) return nil;

  CGPoint o          = inputImage.extent.origin;
  CGFloat inputWidth = inputImage.extent.size.width;

  CGAffineTransform to = CGAffineTransformMakeTranslation ( -o.x, -o.y );

  CIImage *result = [inputImage imageByApplyingTransform:to];

  result = [[self myKernel] applyWithExtent: … ]; // same as before

  CGAffineTransform from = CGAffineTransformMakeTranslation ( o.x, o.y );

  return [result imageByApplyingTransform:from];
}
```

Invoking MirrorX Kernel

```
- (CIImage *) outputImage {
  if ( CGRectIsInfinite ( inputImage.extent ) ) return nil;

  CGPoint o          = inputImage.extent.origin;
  CGFloat inputWidth = inputImage.extent.size.width;

  CGAffineTransform to = CGAffineTransformMakeTranslation ( -o.x, -o.y );

  CIImage *result = [inputImage imageByApplyingTransform:to];

  result = [[self myKernel] applyWithExtent: … ]; // same as before

  CGAffineTransform from = CGAffineTransformMakeTranslation ( o.x, o.y );

  return [result imageByApplyingTransform:from];
}
```

Invoking MirrorX Kernel

```
- (CIImage *) outputImage {
  if ( CGRectIsInfinite ( inputImage.extent ) ) return nil;

  CGPoint o          = inputImage.extent.origin;
  CGFloat inputWidth = inputImage.extent.size.width;

  CGAffineTransform to = CGAffineTransformMakeTranslation ( -o.x, -o.y );

  CIImage *result = [inputImage imageByApplyingTransform:to];

  result = [[self myKernel] applyWithExtent: … ]; // same as before

  CGAffineTransform from = CGAffineTransformMakeTranslation ( o.x, o.y );

  return [result imageByApplyingTransform:from];
}
```

Invoking MirrorX Kernel

```
- (CIImage *) outputImage {
  if ( CGRectIsInfinite ( inputImage.extent ) ) return nil;

  CGPoint o          = inputImage.extent.origin;
  CGFloat inputWidth = inputImage.extent.size.width;

  CGAffineTransform to = CGAffineTransformMakeTranslation ( -o.x, -o.y );

  CIImage *result = [inputImage imageByApplyingTransform:to];

  result = [[self myKernel] applyWithExtent: … ]; // same as before

  CGAffineTransform from = CGAffineTransformMakeTranslation ( o.x, o.y );

  return [result imageByApplyingTransform:from];
}
```

More Complex Warp Kernel

Anamorphic stretch

More Complex Warp Kernel

Anamorphic stretch

1024

More Complex Warp Kernel

Anamorphic stretch

More Complex Warp Kernel

Anamorphic stretch

768

1024

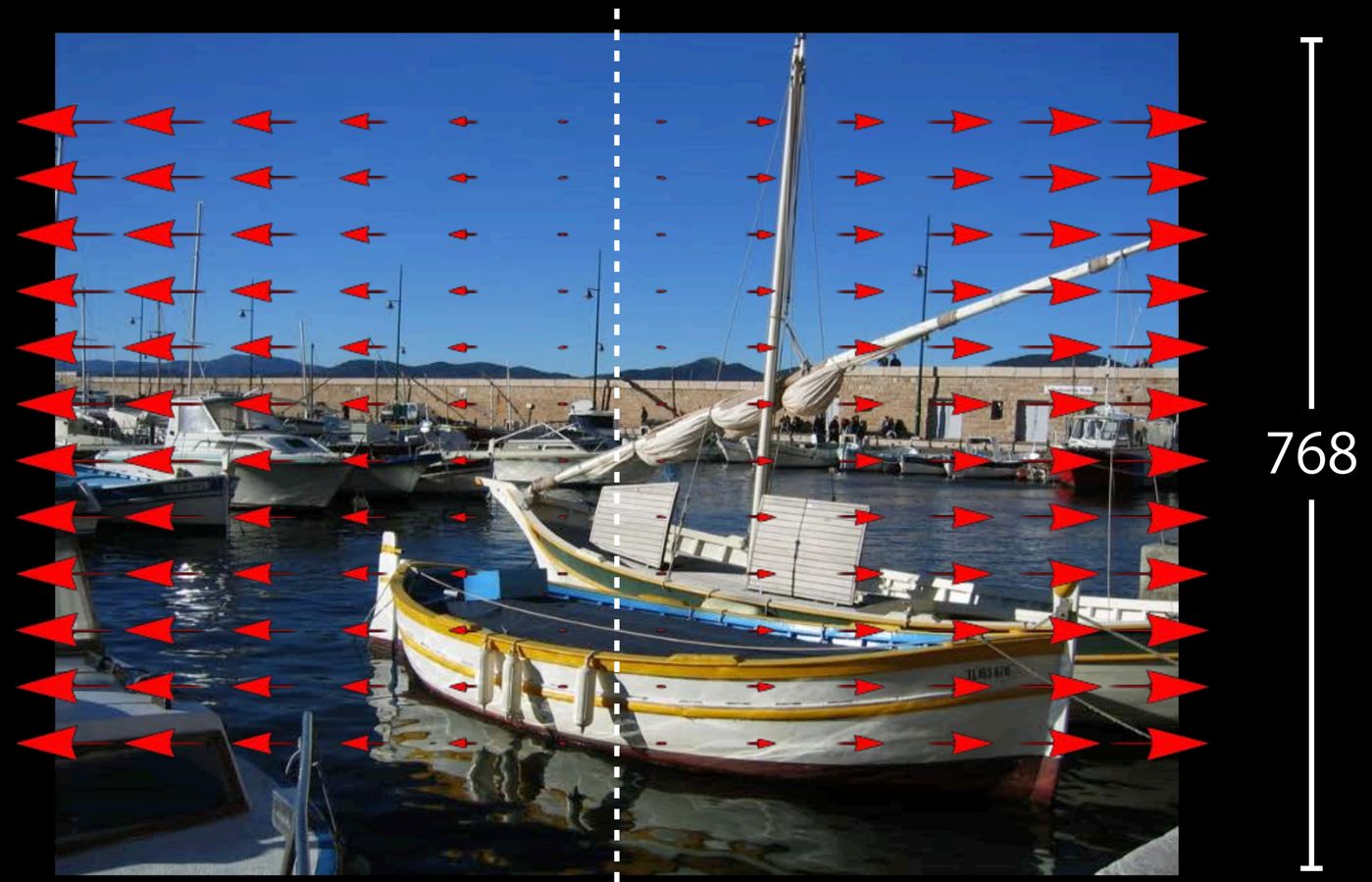
1280

More Complex Warp Kernel

Anamorphic stretch

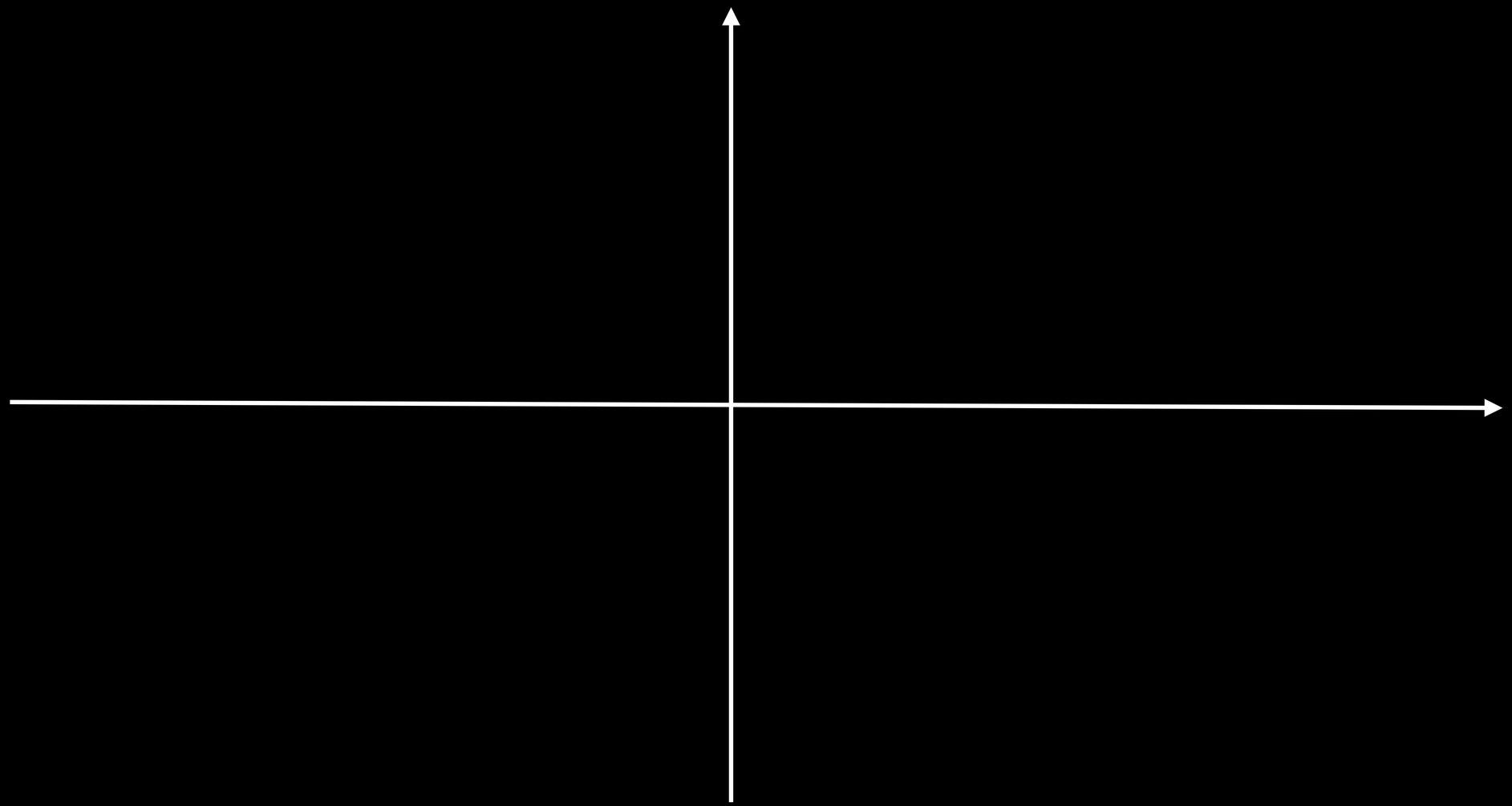
More Complex Warp Kernel

Anamorphic stretch



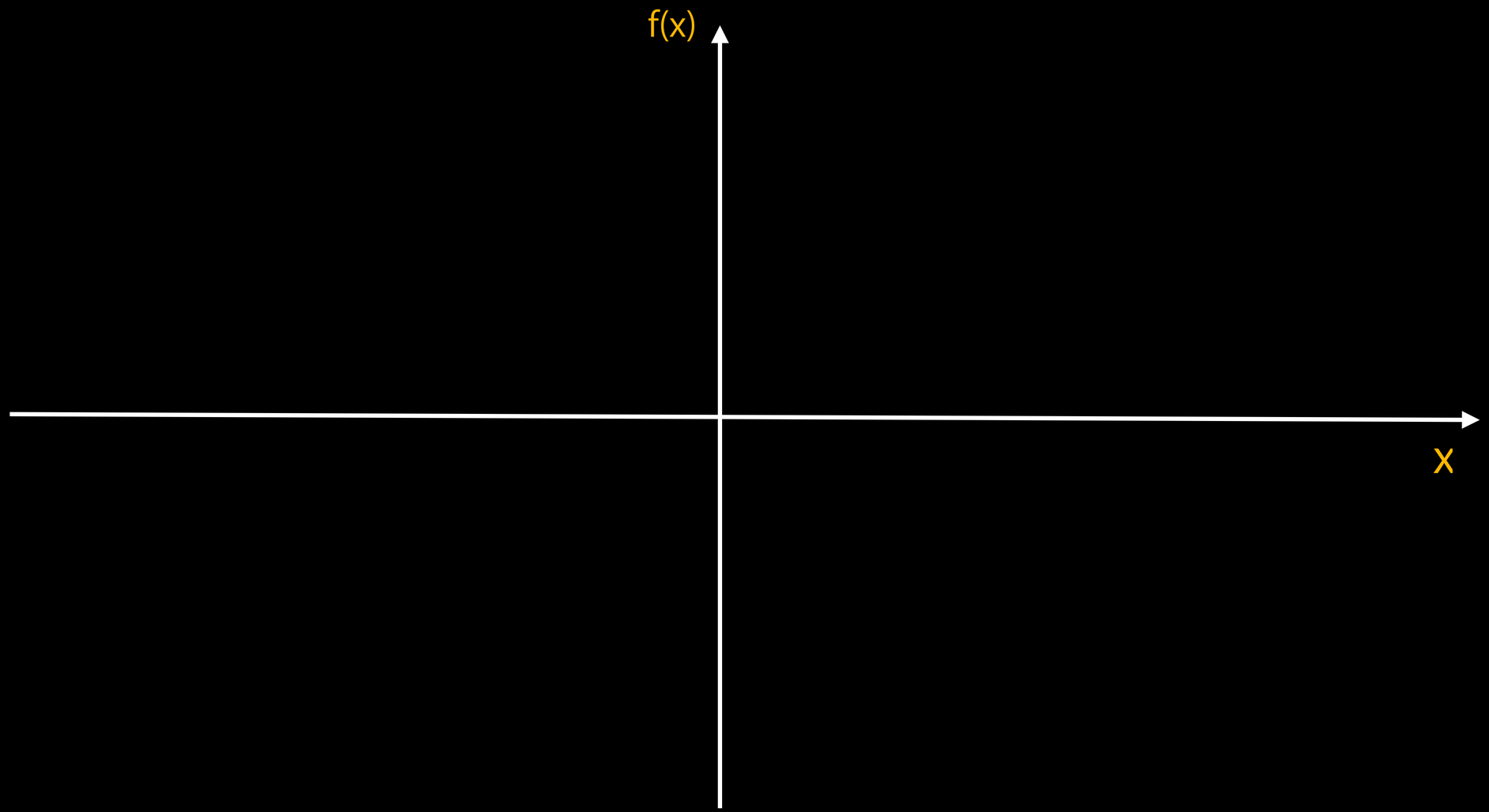
sourceToDest() and destToSource()

Invertible functions



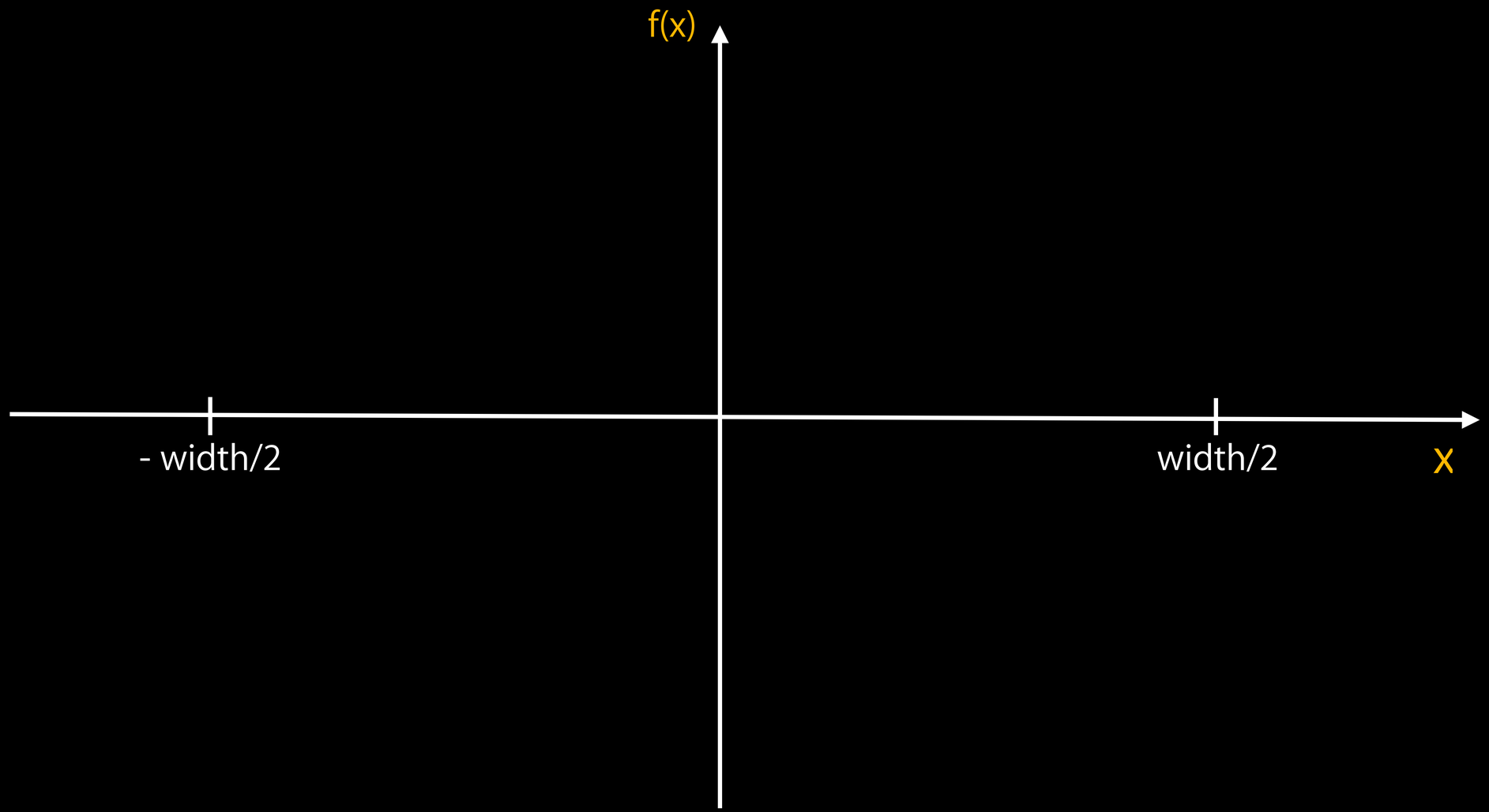
sourceToDest() and destToSource()

Invertible functions



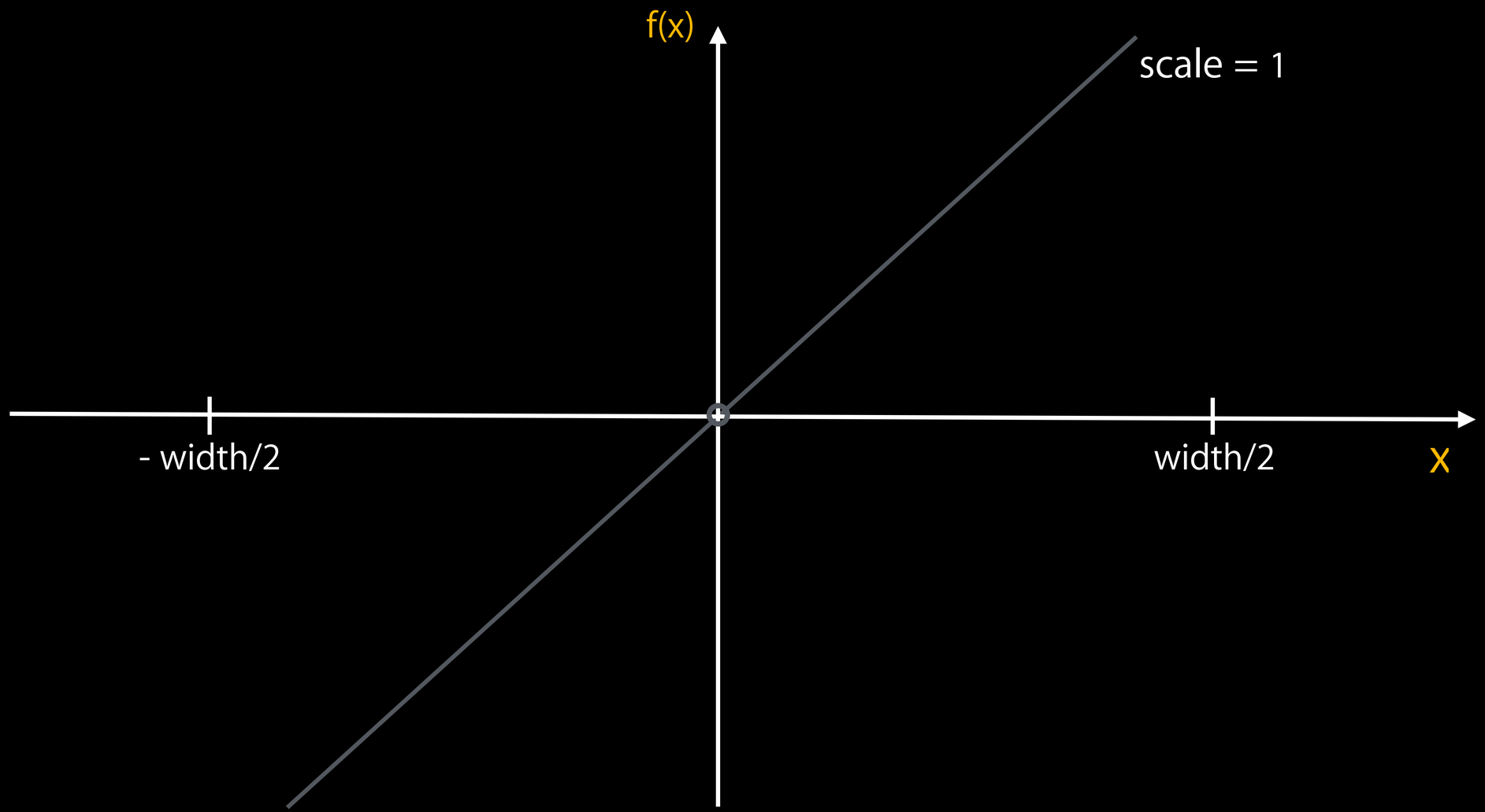
sourceToDest() and destToSource()

Invertible functions



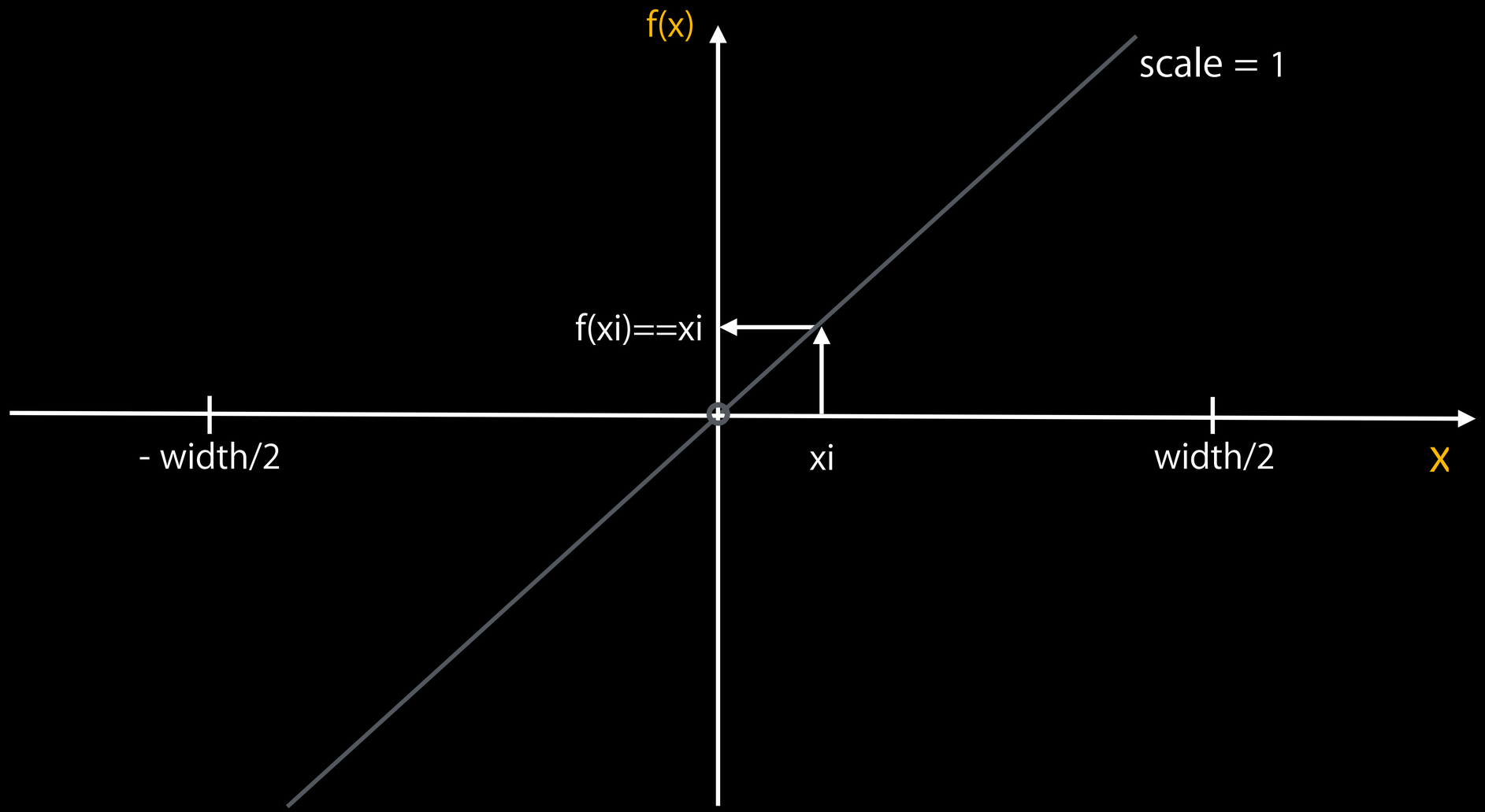
sourceToDest() and destToSource()

Invertible functions



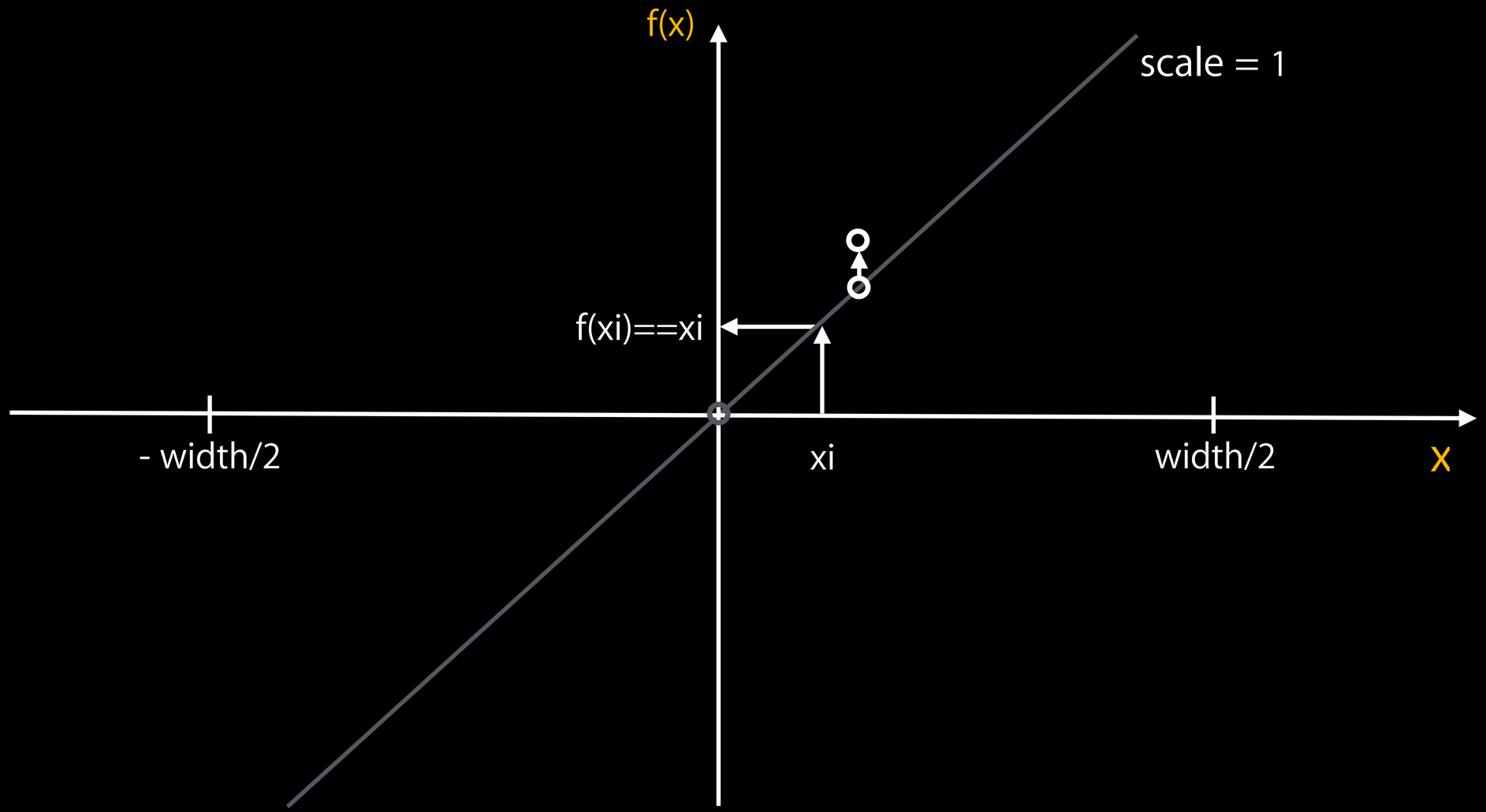
sourceToDest() and destToSource()

Invertible functions



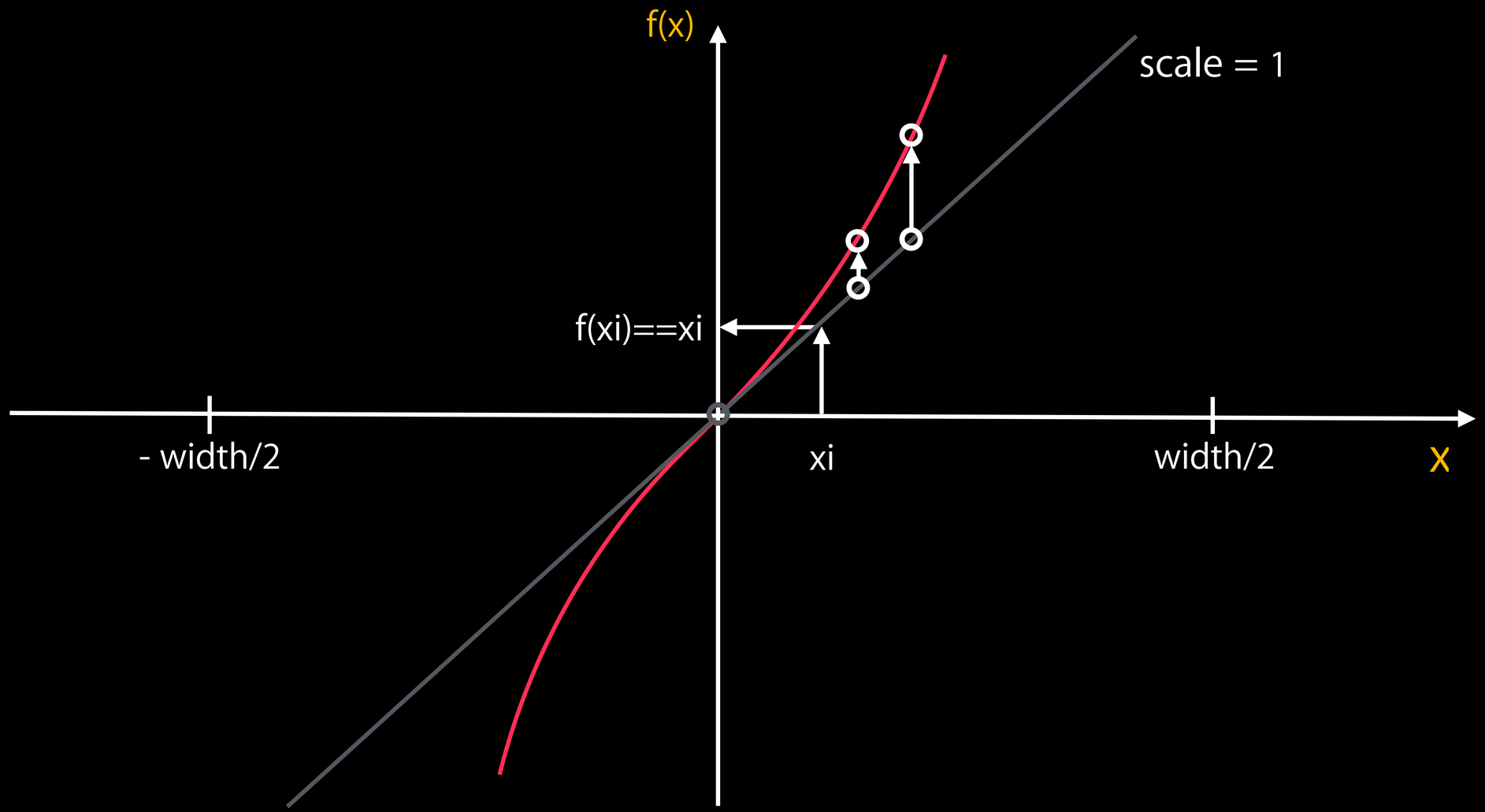
sourceToDest() and destToSource()

Invertible functions



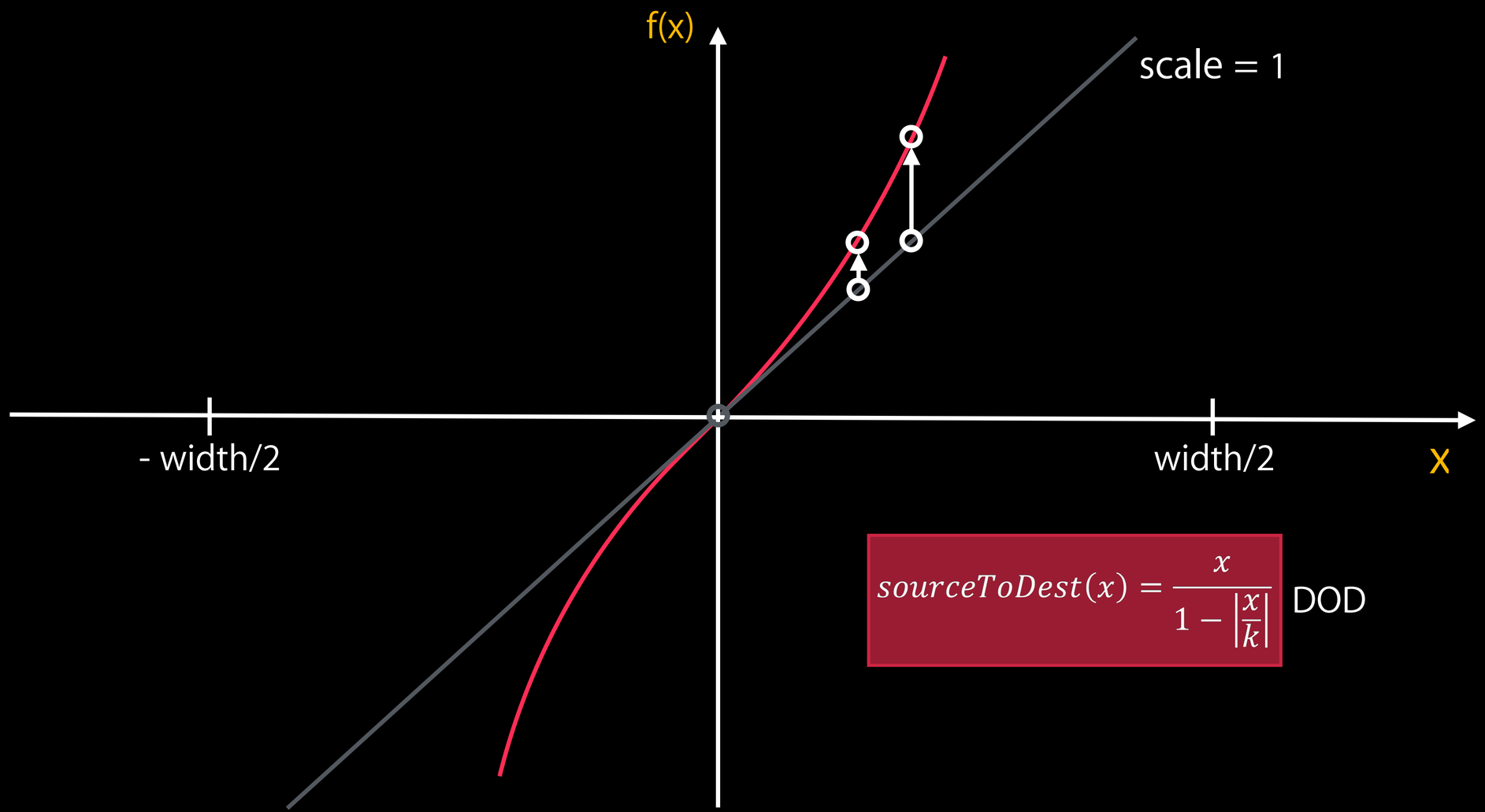
sourceToDest() and destToSource()

Invertible functions



sourceToDest() and destToSource()

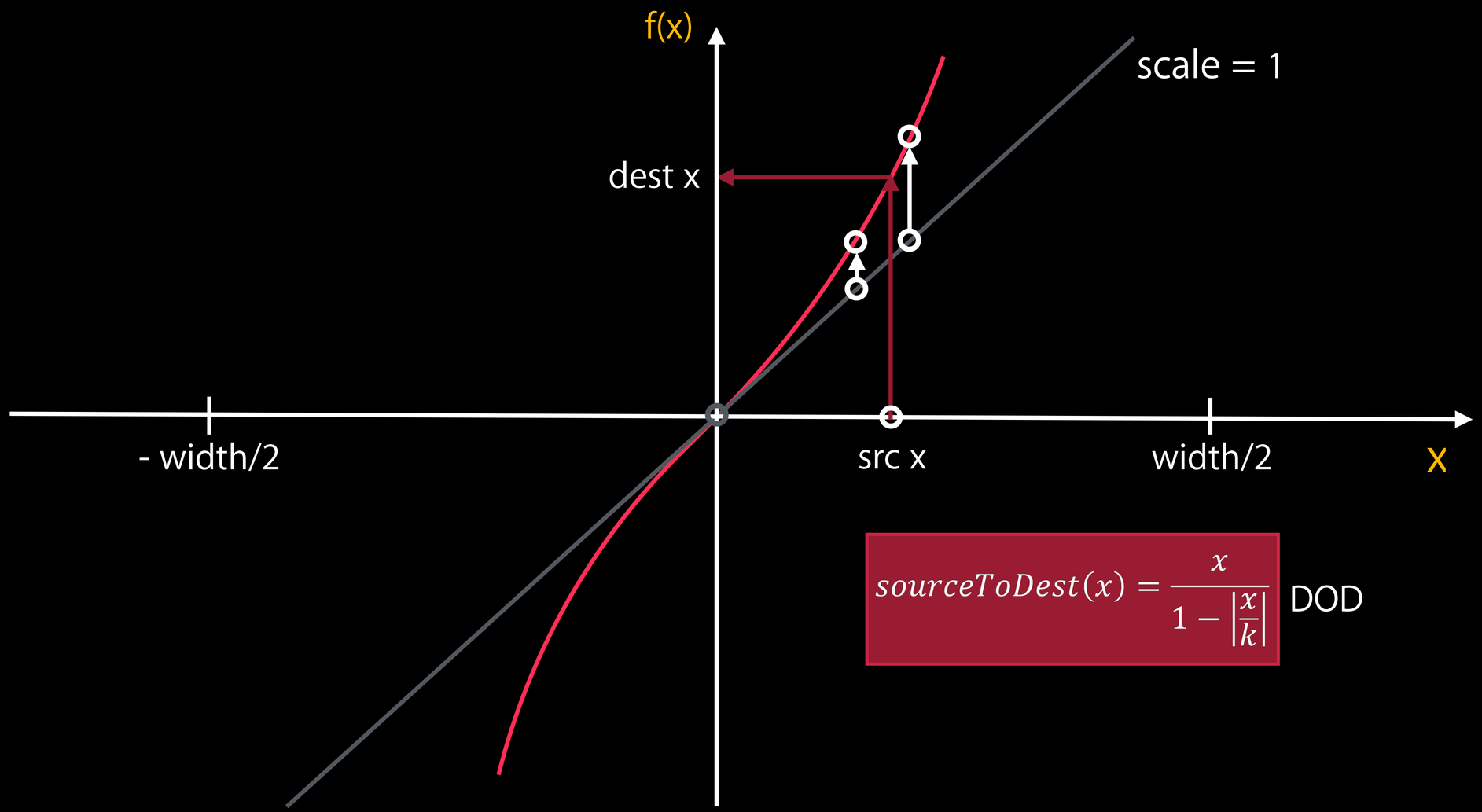
Invertible functions



$$sourceToDest(x) = \frac{x}{1 - \left|\frac{x}{k}\right|}$$ DOD

sourceToDest() and destToSource()

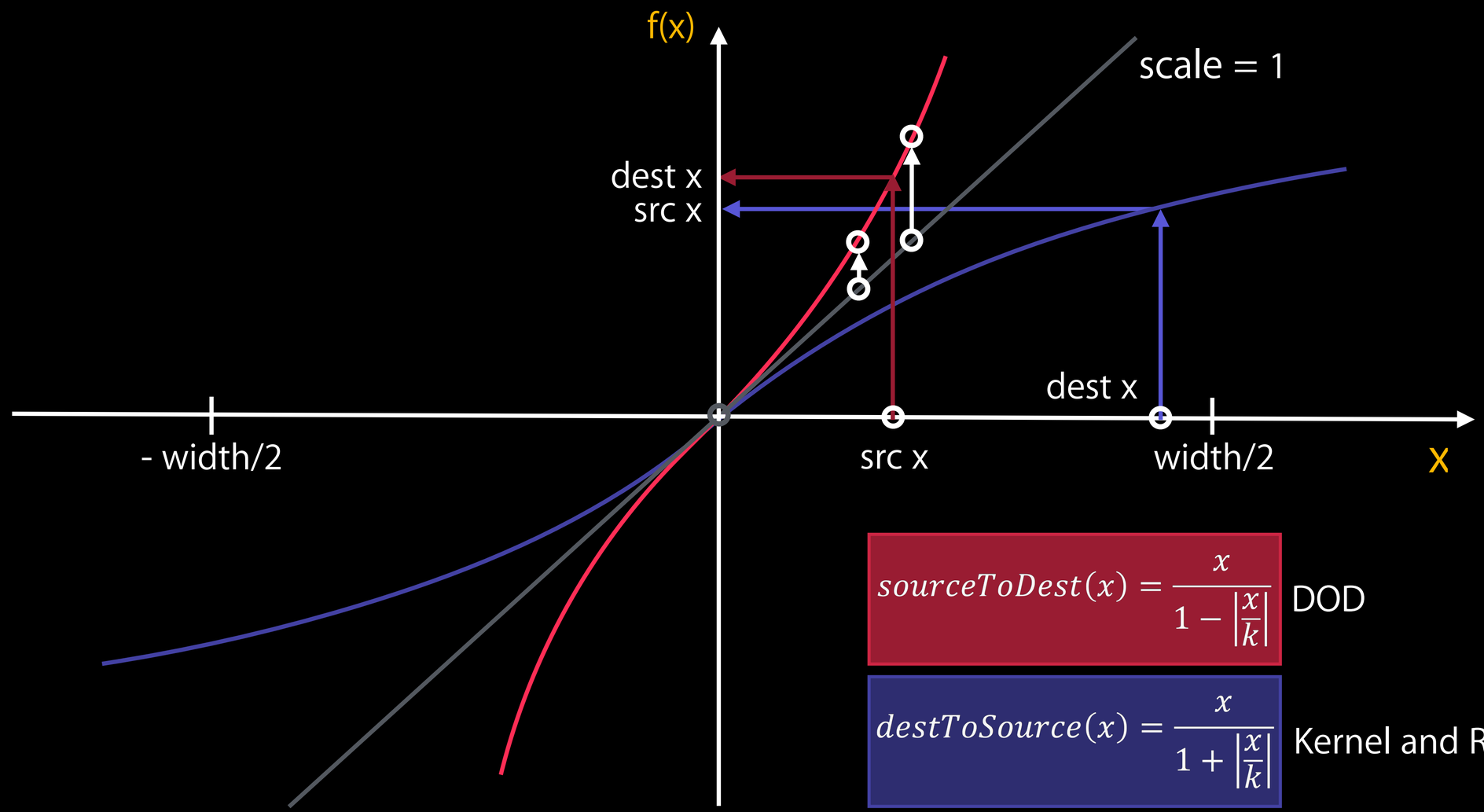
Invertible functions



$$sourceToDest(x) = \frac{x}{1 - \left|\frac{x}{k}\right|}$$ DOD

sourceToDest() and destToSource()

Invertible functions



$$sourceToDest(x) = \frac{x}{1 - \left|\frac{x}{k}\right|}$$ DOD

$$destToSource(x) = \frac{x}{1 + \left|\frac{x}{k}\right|}$$ Kernel and ROI

sourceToDest() and destToSource()

Invertible functions

```
scale = desiredWidth/inputWidth
k = inputWidth/(1.0 – (1.0/scale))
```

sourceToDest() and destToSource()

Invertible functions

```
scale = desiredWidth/inputWidth
k = inputWidth/(1.0 – (1.0/scale))

sourceToDest (1024) == 1280
destToSource (1280) == 1024
```

Kernel

```
kernel vec2 strech (float center, float k)
{
    vec2 p = destCoord();

    return p;
}
```

Kernel

```
kernel vec2 strech (float center, float k)
{
    vec2 p = destCoord();
    return p;
}
```

$$destToSource(x) = \frac{x}{1 + \left|\frac{x}{k}\right|}$$

Kernel

```
kernel vec2 strech (float center, float k)
{
    vec2 p = destCoord();
    p.x -= center;

    return p;
}
```

$$destToSource(x) = \frac{x}{1 + \left|\frac{x}{k}\right|}$$

Kernel

```
kernel vec2 strech (float center, float k)
{
    vec2 p = destCoord();
    p.x -= center;
    p.x  = p.x / (1.0 + abs(p.x / k));
    p.x += center;

    return p;
}
```

$$destToSource(x) = \frac{x}{1 + \left|\frac{x}{k}\right|}$$

ROI Pseudocode

r

ROI Pseudocode

ROI Pseudocode


```
leftP = r.origin.x
```

ROI Pseudocode


```
leftP = r.origin.x
```

ROI Pseudocode


```
leftP = r.origin.x
leftP’ = destToSource ( leftP )
```

ROI Pseudocode


```
leftP = r.origin.x
leftP’ = destToSource ( leftP )
```

```
rightP = r.origin.x + r.size.width
```

ROI Pseudocode


```
leftP = r.origin.x
leftP’ = destToSource ( leftP )
```

```
rightP = r.origin.x + r.size.width
```

ROI Pseudocode


```
leftP = r.origin.x
leftP’ = destToSource ( leftP )
```

```
rightP = r.origin.x + r.size.width
rightP’ = destToSource ( rightP )
```

ROI Pseudocode


```
leftP = r.origin.x
leftP’ = destToSource ( leftP )
```

```
rightP = r.origin.x + r.size.width
rightP’ = destToSource ( rightP )
```

```
newWidth = rightP’ – leftP’
r’ = CGRectMake ( leftP’,  r.origin.y, newWidth, r.size.height )
```

ROI—Region of Interest

Re-use kernel equation

```
double destToSource(double x, double center, double k)
{
    return x;
}
```

ROI—Region of Interest

Re-use kernel equation

```
double destToSource(double x, double center, double k)
{
    return x;
}
```

$$destToSource(x) = \frac{x}{1 + \left|\frac{x}{k}\right|}$$

ROI—Region of Interest

Re-use kernel equation

```
double destToSource(double x, double center, double k)
{
    x -= center;
    x = x / (1.0 + fabs(x / k));
    x += center;
    return x;
}
```

$$destToSource(x) = \frac{x}{1 + \left|\frac{x}{k}\right|}$$

ROI—Region of Interest

Re-use kernel equation

```
double destToSource(double x, double center, double k)
{
    x -= center;
    x = x / (1.0 + fabs(x / k));
    x += center;
    return x;
}

CGRect regionOf (CGRect r, float center, float k)
{
    double leftP  = destToSource (r.origin.x, center, k);
    double rightP = destToSource (r.origin.x + r.size.width, center, k);
    return CGRectMake (leftP, r.origin.y, rightP-leftP, r.size.height);
}
```

$$destToSource(x) = \frac{x}{1 + \left|\frac{x}{k}\right|}$$

DOD—Domain of Definition

```
double sourceToDest (double x, double center, double k)
{
    x -= center;
    x = x / (1.0 - fabs(x / k));
    x += center;
    return x;
}
```

$$sourceToDest(x) = \frac{x}{1 - \left|\frac{x}{k}\right|}$$

DOD—Domain of Definition

```
double sourceToDest (double x, double center, double k)
{
    x -= center;
    x = x / (1.0 - fabs(x / k));
    x += center;
    return x;
}

CGRect dodForInputRect (CGRect r, double center, double k)
{
    double leftP  = sourceToDest (r.origin.x, center, k);
    double rightP = sourceToDest (r.origin.x + r.size.width, center, k);
    return CGRectMake (leftP, r.origin.y, rightP-leftP, r.size.height);
}
```

$$sourceToDest(x) = \frac{x}{1 - \left|\frac{x}{k}\right|}$$

outputImage Method

```
- (CIImage *) outputImage {
    double inputWidth = inputImage.extent.size.width;

    double k = inputWidth / ( 1.0 - 1.0 / [inputScale floatValue] );
    double center = inputImage.extent.origin.x + inputWidth / 2.0;
    CGRect dod = dodForInputRect (inputImage.extent, center, k);

    CIImage * result =
      [[self myKernel] applyWithExtent: dod
                           roiCallback: ^(int index, CGRect rect) {
                              return regionOf(rect, center, k); }
                           inputImage: inputImage
                            arguments: @[@(center), @(k)]];
    return result;
}
```

outputImage Method

```
- (CIImage *) outputImage {
    double inputWidth = inputImage.extent.size.width;

    double k = inputWidth / ( 1.0 - 1.0 / [inputScale floatValue] );
    double center = inputImage.extent.origin.x + inputWidth / 2.0;
    CGRect dod = dodForInputRect (inputImage.extent, center, k);

    CIImage * result =
      [[self myKernel] applyWithExtent: dod
                           roiCallback: ^(int index, CGRect rect) {
                              return regionOf(rect, center, k); }
                           inputImage: inputImage
                            arguments: @[@(center), @(k)]];
    return result;
}
```

outputImage Method

```
- (CIImage *) outputImage {
    double inputWidth = inputImage.extent.size.width;

    double k = inputWidth / ( 1.0 - 1.0 / [inputScale floatValue] );
    double center = inputImage.extent.origin.x + inputWidth / 2.0;
    CGRect dod = dodForInputRect (inputImage.extent, center, k);

    CIImage * result =
      [[self myKernel] applyWithExtent: dod
                           roiCallback: ^(int index, CGRect rect) {
                              return regionOf(rect, center, k); }
                           inputImage: inputImage
                            arguments: @[@(center), @(k)]];
    return result;
}
```

outputImage Method

```
- (CIImage *) outputImage {
    double inputWidth = inputImage.extent.size.width;

    double k = inputWidth / ( 1.0 - 1.0 / [inputScale floatValue] );
    double center = inputImage.extent.origin.x + inputWidth / 2.0;
    CGRect dod = dodForInputRect (inputImage.extent, center, k);

    CIImage * result =
      [[self myKernel] applyWithExtent: dod
                           roiCallback: ^(int index, CGRect rect) {
                              return regionOf(rect, center, k); }
                           inputImage: inputImage
                            arguments: @[@(center), @(k)]];

    return result;
}
```

outputImage Method

```
- (CIImage *) outputImage {
    double inputWidth = inputImage.extent.size.width;

    double k = inputWidth / ( 1.0 - 1.0 / [inputScale floatValue] );
    double center = inputImage.extent.origin.x + inputWidth / 2.0;
    CGRect dod = dodForInputRect (inputImage.extent, center, k);

    CIImage * result =
      [[self myKernel] applyWithExtent: dod
                           roiCallback: ^(int index, CGRect rect) {
                              return regionOf(rect, center, k); }
                           inputImage: inputImage
                            arguments: @[@(center), @(k)]];
    return result;
}
```

customAttributes Method

```
+ (NSDictionary *)customAttributes {
  return @{  kCIAttributeFilterDisplayName : @“Anamorphic Stretch",
             kCIAttributeFilterCategories :
              @[kCICategoryDistortionEffect, kCICategoryVideo,
                 kCICategoryInterlaced, kCICategoryNonSquarePixels,
                 kCICategoryStillImage],
            @"inputScale" : @{
                    kCIAttributeSliderMin : @1.0,
                    kCIAttributeSliderMax : @2.0,
                    kCIAttributeDefault   : @1.0,
                    kCIAttributeIdentity  : @1.0,
                    kCIAttributeType      : kCIAttributeTypeScalar
            }
          };
}
```

customAttributes Method

```
+ (NSDictionary *)customAttributes {
  return @{  kCIAttributeFilterDisplayName : @“Anamorphic Stretch",
             kCIAttributeFilterCategories :
              @[kCICategoryDistortionEffect, kCICategoryVideo,
                 kCICategoryInterlaced, kCICategoryNonSquarePixels,
                 kCICategoryStillImage],
            @"inputScale" : @{
                    kCIAttributeSliderMin : @1.0,
                    kCIAttributeSliderMax : @2.0,
                    kCIAttributeDefault   : @1.0,
                    kCIAttributeIdentity  : @1.0,
                    kCIAttributeType      : kCIAttributeTypeScalar
            }
          };
}
```

customAttributes Method

```
+ (NSDictionary *)customAttributes {
  return @{  kCIAttributeFilterDisplayName : @“Anamorphic Stretch",
             kCIAttributeFilterCategories :
              @[kCICategoryDistortionEffect, kCICategoryVideo,
                 kCICategoryInterlaced, kCICategoryNonSquarePixels,
                 kCICategoryStillImage],
            @"inputScale" : @{
                    kCIAttributeSliderMin : @1.0,
                    kCIAttributeSliderMax : @2.0,
                    kCIAttributeDefault   : @1.0,
                    kCIAttributeIdentity  : @1.0,
                    kCIAttributeType      : kCIAttributeTypeScalar
            }
           };
}
```

customAttributes Method

```
+ (NSDictionary *)customAttributes {
  return @{  kCIAttributeFilterDisplayName : @“Anamorphic Stretch",
             kCIAttributeFilterCategories :
              @[kCICategoryDistortionEffect, kCICategoryVideo,
                 kCICategoryInterlaced, kCICategoryNonSquarePixels,
                 kCICategoryStillImage],
              @"inputScale" : @{
                    kCIAttributeSliderMin : @1.0,
                    kCIAttributeSliderMax : @2.0,
                    kCIAttributeDefault   : @1.0,
                    kCIAttributeIdentity  : @1.0,
                    kCIAttributeType      : kCIAttributeTypeScalar
                }
            };
}
```

CIKernel Wrap-Up

	# Input Images	Input Type	Access	Output Image	DOD	ROI
CIColorKernel	0 .. n	__sample	vec4	vec4	YES	NO
CIWarpKernel	1	destCoord()	vec2	vec2	YES	YES

CIKernel Wrap-Up

	# Input Images	Input Type	Access	Output Image	DOD	ROI
CIColorKernel	0 .. n	__sample	vec4	vec4	YES	NO
CIWarpKernel	1	destCoord()	vec2	vec2	YES	YES

CIKernel Wrap-Up

	# Input Images	Input Type	Access	Output Image	DOD	ROI
CIColorKernel	0 .. n	__sample	vec4	vec4	YES	NO
CIWarpKernel	1	destCoord()	vec2	vec2	YES	YES

CIKernel Wrap-Up

	# Input Images	Input Type	Access	Output Image	DOD	ROI
CIColorKernel	0 .. n	__sample	vec4	vec4	YES	NO
CIWarpKernel	1	destCoord()	vec2	vec2	YES	YES

CIKernel Wrap-Up

	# Input Images	Input Type	Access	Output Image	DOD	ROI
CIColorKernel	0 .. n	__sample	vec4	vec4	YES	NO
CIWarpKernel	1	destCoord()	vec2	vec2	YES	YES

CIKernel Wrap-Up

	# Input Images	Input Type	Access	Output Image	DOD	ROI
CIColorKernel	0 .. n	__sample	vec4	vec4	YES	NO
CIWarpKernel	1	destCoord()	vec2	vec2	YES	YES

CIKernel Wrap-Up

	# Input Images	Input Type	Access	Output Image	DOD	ROI
CIColorKernel	0 .. n	__sample	vec4	vec4	YES	NO
CIWarpKernel	1	destCoord()	vec2	vec2	YES	YES

CIKernel Wrap-Up

	# Input Images	Input Type	Access	Output Image	DOD	ROI
CIColorKernel	0 .. n	__sample	vec4	vec4	YES	NO
CIWarpKernel	1	destCoord()	vec2	vec2	YES	YES

CIKernel Wrap-Up

	# Input Images	Input Type	Access	Output Image	DOD	ROI
CIColorKernel	0 .. n	__sample	vec4	vec4	YES	NO
CIWarpKernel	1	destCoord()	vec2	vec2	YES	YES

CIKernel Wrap-Up

	# Input Images	Input Type	Access	Output Image	DOD	ROI
CIColorKernel	0 .. n	__sample	vec4	vec4	YES	NO
CIWarpKernel	1	destCoord()	vec2	vec2	YES	YES

CIKernel Wrap-Up

	# Input Images	Input Type	Access	Output Image	DOD	ROI
CIColorKernel	0 .. n	__sample	vec4	vec4	YES	NO
CIWarpKernel	1	destCoord()	vec2	vec2	YES	YES

CIKernel Wrap-Up

	# Input Images	Input Type	Access	Output Image	DOD	ROI
CIColorKernel	0 .. n	__sample	vec4	vec4	YES	NO
CIWarpKernel	1	destCoord()	vec2	vec2	YES	YES

CIKernel Wrap-Up

	# Input Images	Input Type	Access	Output Image	DOD	ROI
CIColorKernel	0 .. n	__sample	vec4	vec4	YES	NO
CIWarpKernel	1	destCoord()	vec2	vec2	YES	YES

CIKernel Wrap-Up

	# Input Images	Input Type	Access	Output Image	DOD	ROI
CIColorKernel	0 .. n	__sample	vec4	vec4	YES	NO
CIWarpKernel	1	destCoord()	vec2	vec2	YES	YES
CIKernel	0 .. n	sampler	sample(…)	vec4	YES	YES

General Kernels

CIKernel

Tony Chu

Different Types of Kernels

Color Kernels (CIColorKernel)

Warp Kernels (CIWarpKernel)

General Kernels (CIKernel)

Motivation

CIKernel

When would you need to write a general kernel?

Motivation

CIKernel

When would you need to write a general kernel?

- Kernel needs multiple samples of an image

Motivation

CIKernel

When would you need to write a general kernel?

- Kernel needs multiple samples of an image
  - e.g., any kind of blur or convolution filter

Motivation

CIKernel

When would you need to write a general kernel?

- Kernel needs multiple samples of an image
  - e.g., any kind of blur or convolution filter
- Kernel contains a dependent texture read

Motivation

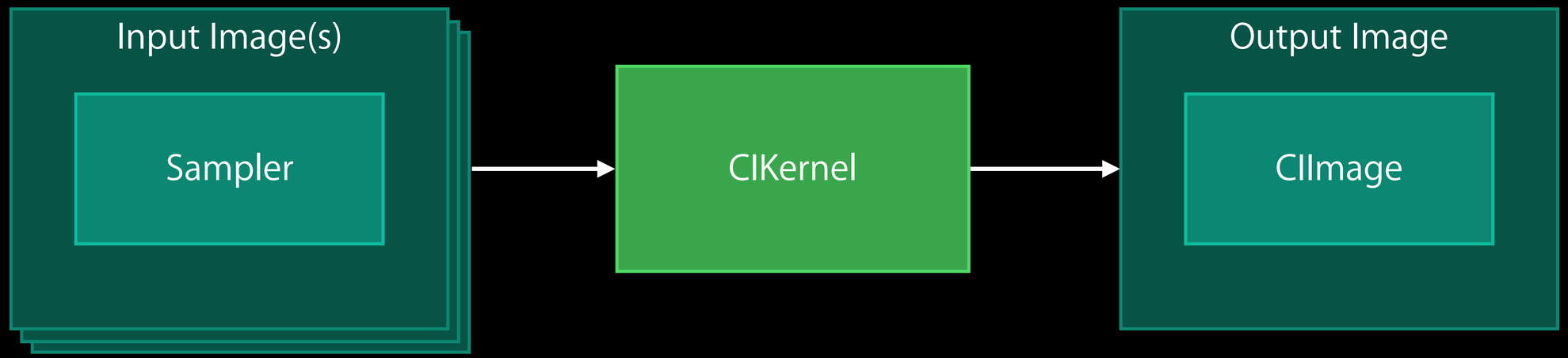
CIKernel

When would you need to write a general kernel?

- Kernel needs multiple samples of an image
  - e.g., any kind of blur or convolution filter
- Kernel contains a dependent texture read
  - e.g., sample from image A used to determine where to sample from image B

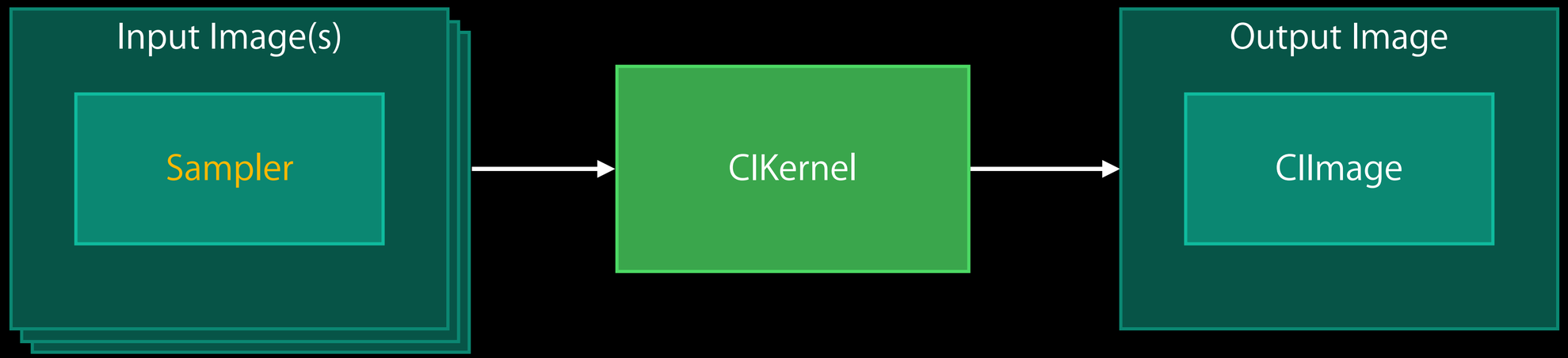
Basic Principles of General Kernels

CIKernel



Basic Principles of General Kernels

CIKernel



Basic Principles of General Kernels

```
kernel vec4 do_nothing (sampler image) {
    vec2 dc = destCoord();
    return sample (image, samplerTransform (image, dc));
}
```

Basic Principles of General Kernels

```
kernel vec4 do_nothing (sampler image) {
    vec2 dc = destCoord();
    return sample (image, samplerTransform (image, dc));
}
```

```
kernel vec4 do_nothing (sampler image) {
    return sample (image, samplerCoord (image));
}
```

Basic Principles of General Kernels

```
kernel vec4 do_nothing (sampler image) {
    vec2 dc = destCoord();
    return sample (image, samplerTransform (image, dc));
}
```

—

```
kernel vec4 do_nothing (sampler image) {
    return sample (image, samplerCoord (image));
}
```

Basic Principles of General Kernels

```
kernel vec4 do_nothing (sampler image) {
    vec2 dc = destCoord();
    return sample (image, samplerTransform (image, dc));
}
```

—

```
kernel vec4 do_nothing (sampler image) {
    return sample (image, samplerCoord (image));
}
```

Why use samplerTransform?

Basic Principles of General Kernels

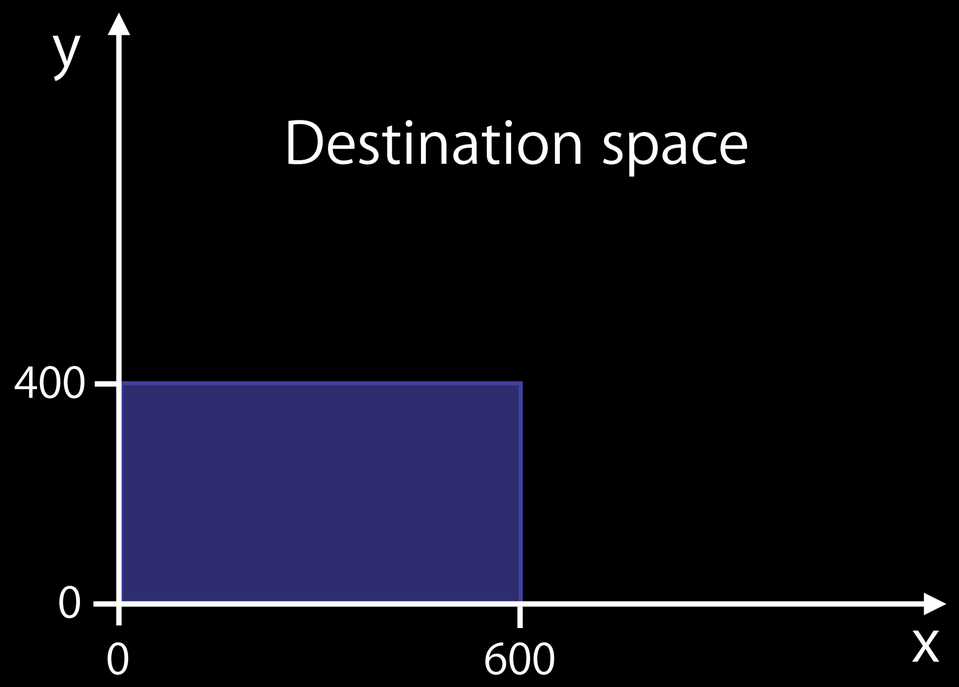
Destination space vs. Sampler space

```
kernel vec4 do_something (sampler image) {
    vec2 sc = samplerCoord (image);
    vec2 offset = vec2(0.0, 2.0);
    return sample (image, sc + offset);
}
```

Basic Principles of General Kernels

Destination space vs. Sampler space

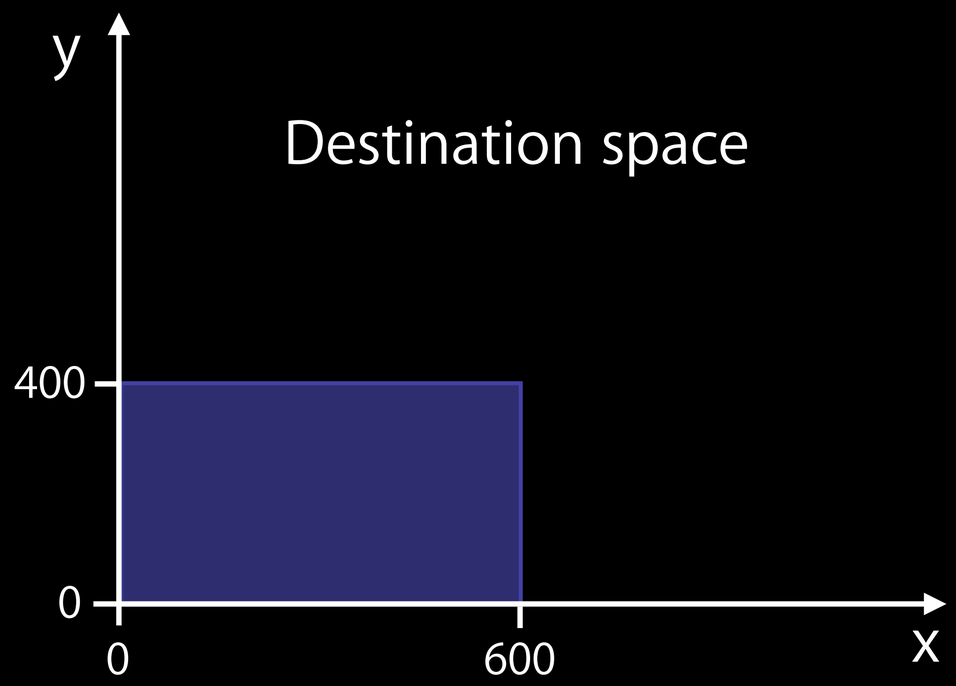
```
kernel vec4 do_something (sampler image) {
    vec2 sc = samplerCoord (image);
    vec2 offset = vec2(0.0, 2.0);
    return sample (image, sc + offset);
}
```

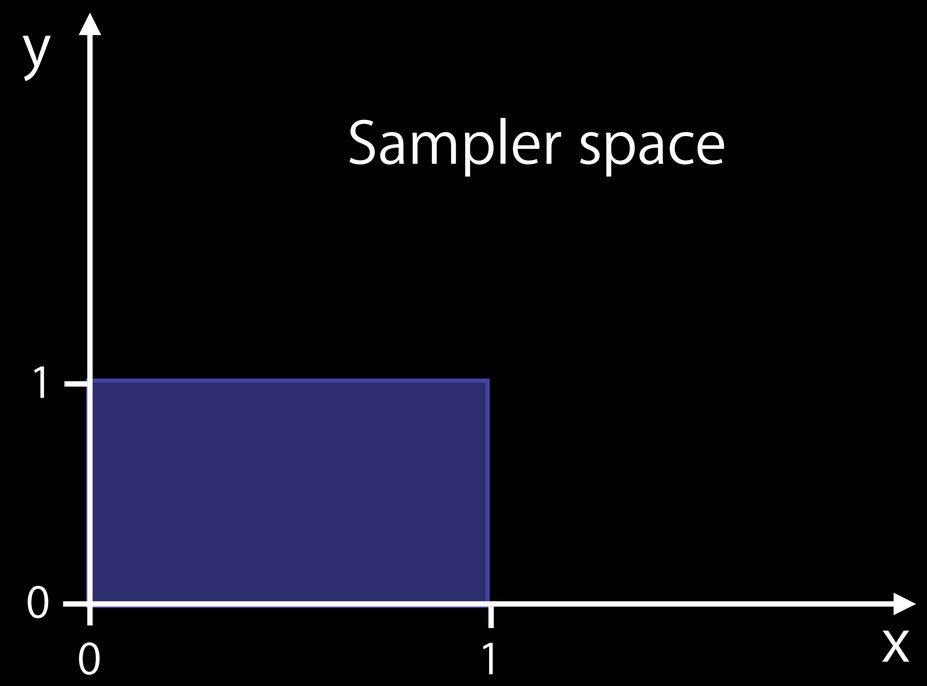


Basic Principles of General Kernels

Destination space vs. Sampler space

```
kernel vec4 do_something (sampler image) {
    vec2 sc = samplerCoord (image);
    vec2 offset = vec2(0.0, 2.0);
    return sample (image, sc + offset);
}
```

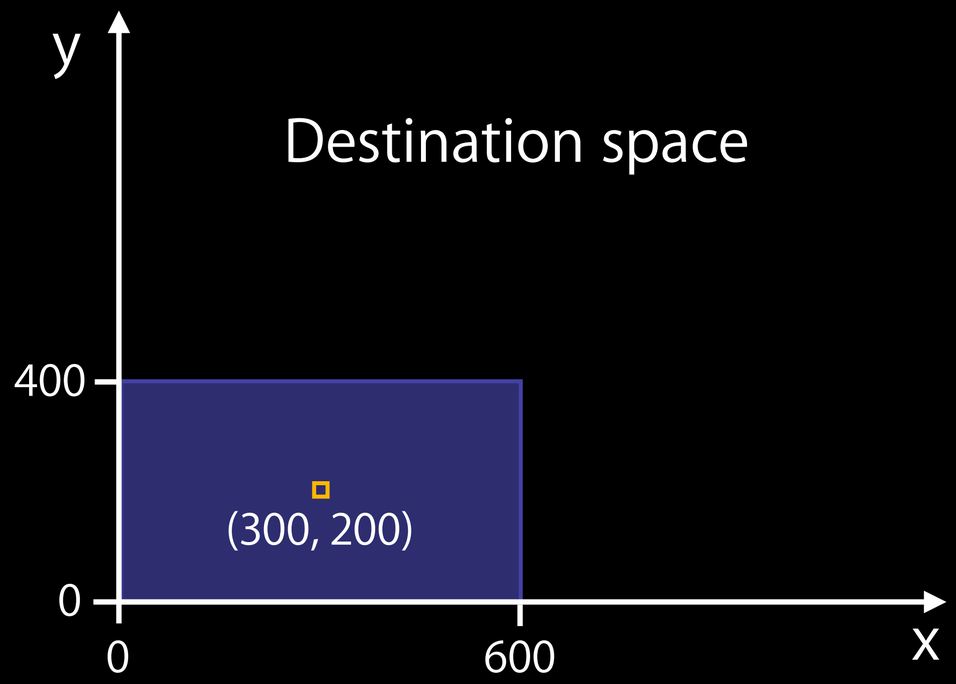


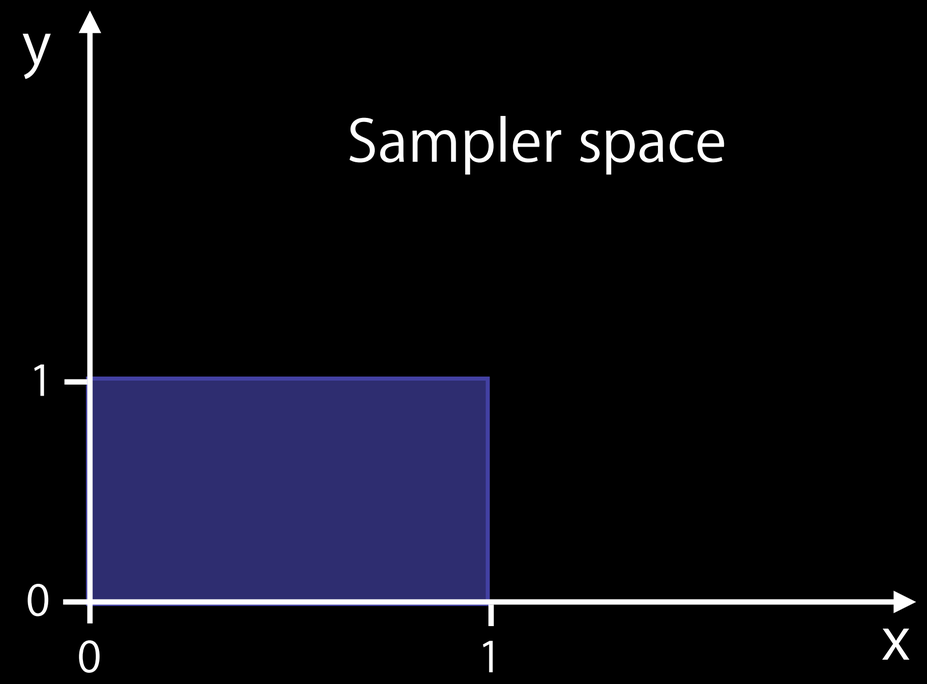


Basic Principles of General Kernels

Destination space vs. Sampler space

```
kernel vec4 do_something (sampler image) {
    vec2 sc = samplerCoord (image);
    vec2 offset = vec2(0.0, 2.0);
    return sample (image, sc + offset);
}
```

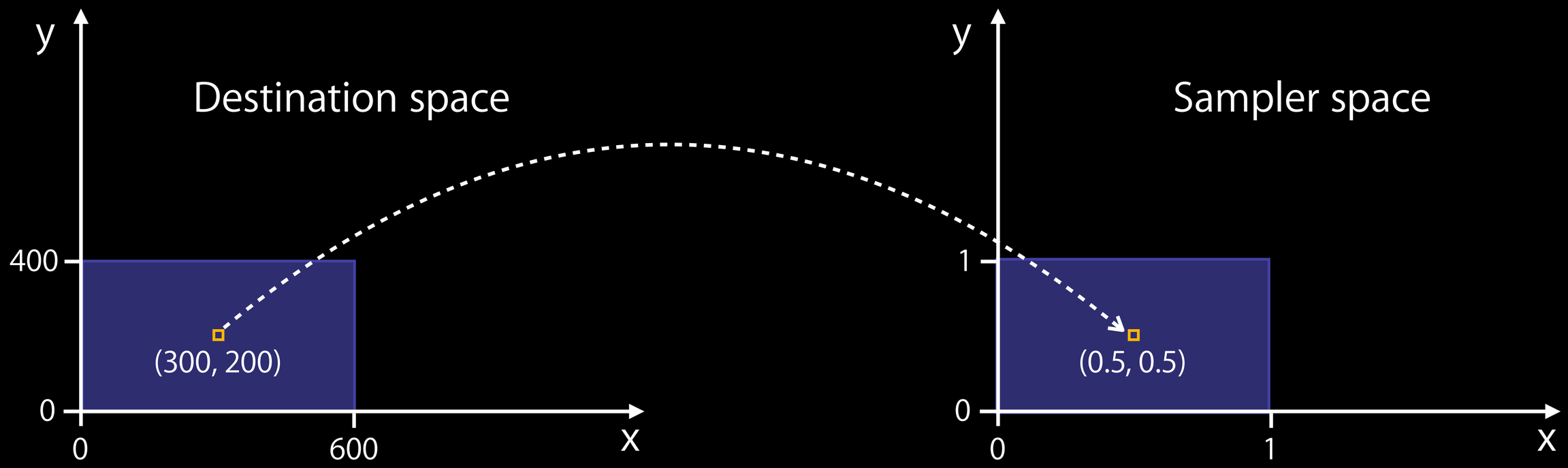




Basic Principles of General Kernels

Destination space vs. Sampler space

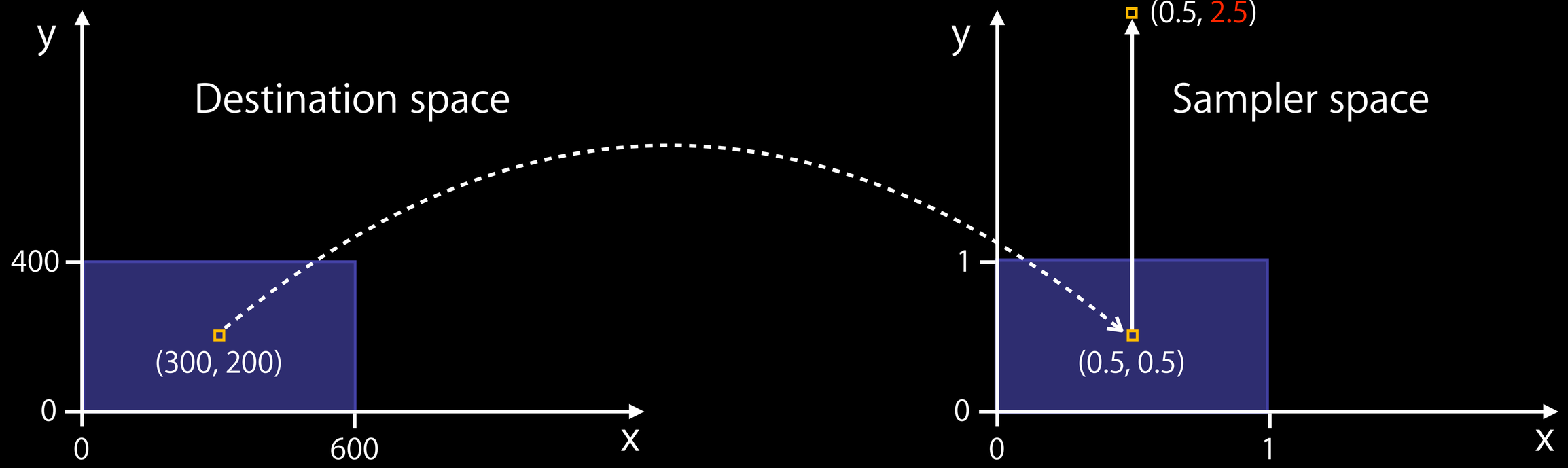
```
kernel vec4 do_something (sampler image) {
    vec2 sc = samplerCoord (image);
    vec2 offset = vec2(0.0, 2.0);
    return sample (image, sc + offset);
}
```



Basic Principles of General Kernels

Destination space vs. Sampler space

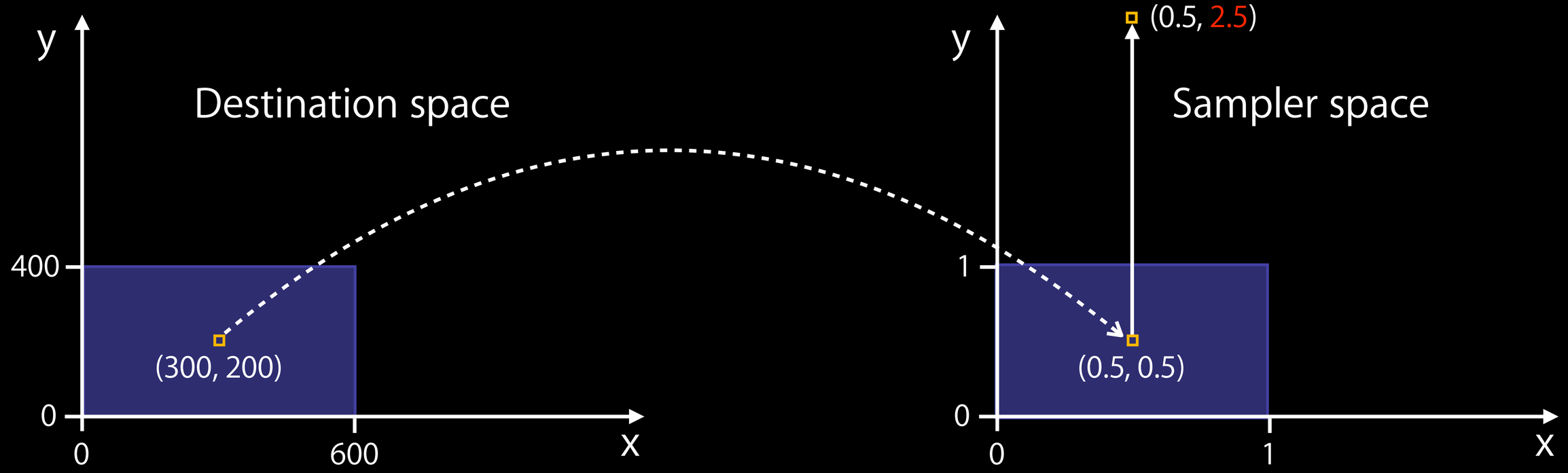
```
kernel vec4 do_something (sampler image) {
    vec2 sc = samplerCoord (image);
    vec2 offset = vec2(0.0, 2.0);
    return sample (image, sc + offset);
}
```



Basic Principles of General Kernels

Destination space vs. Sampler space

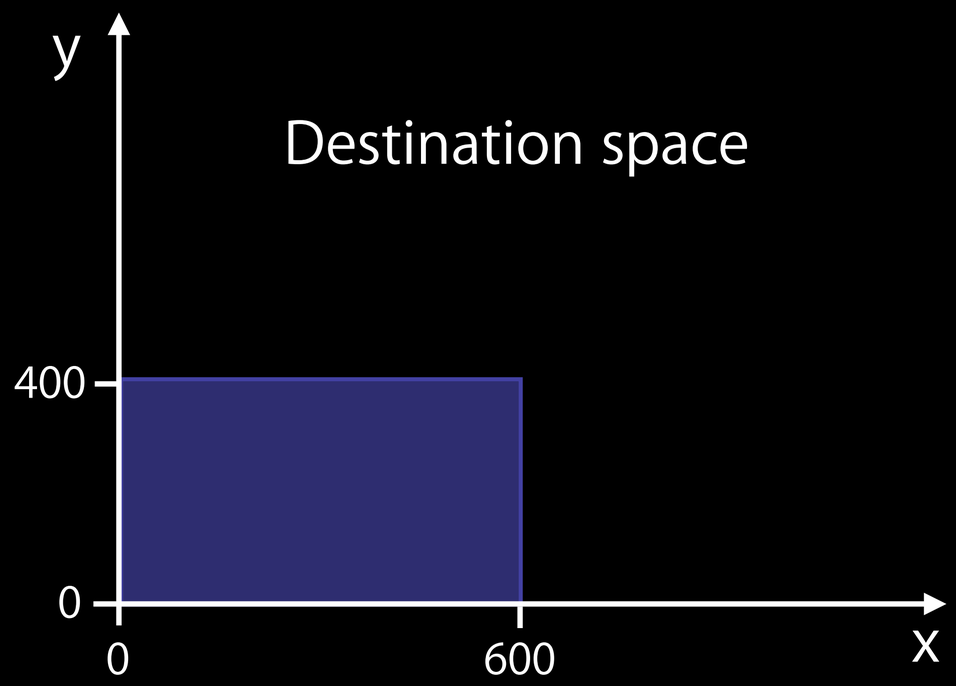
```
kernel vec4 do_something (sampler image) {
    vec2 sc = samplerCoord (image);
    vec2 offset = vec2(0.0, 2.0);
    return sample (image, sc + offset);
}
```

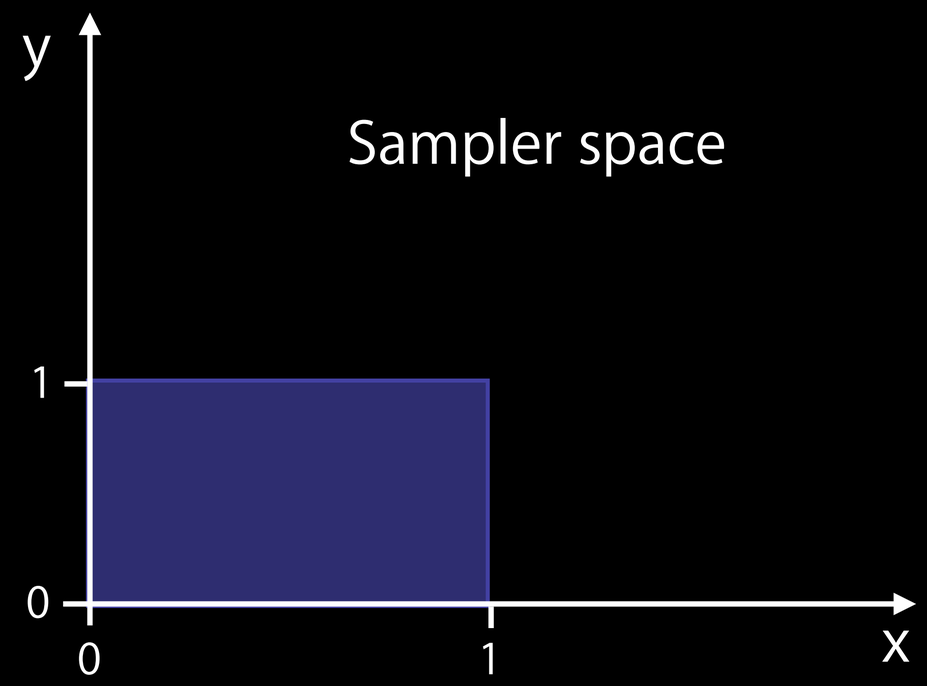


Basic Principles of General Kernels

Destination space vs. Sampler space

```
kernel vec4 do_something (sampler image) {
    vec2 dc = destCoord();
    vec2 offset = vec2(0.0, 2.0);
    return sample (image, samplerTransform (image, dc + offset));
}
```

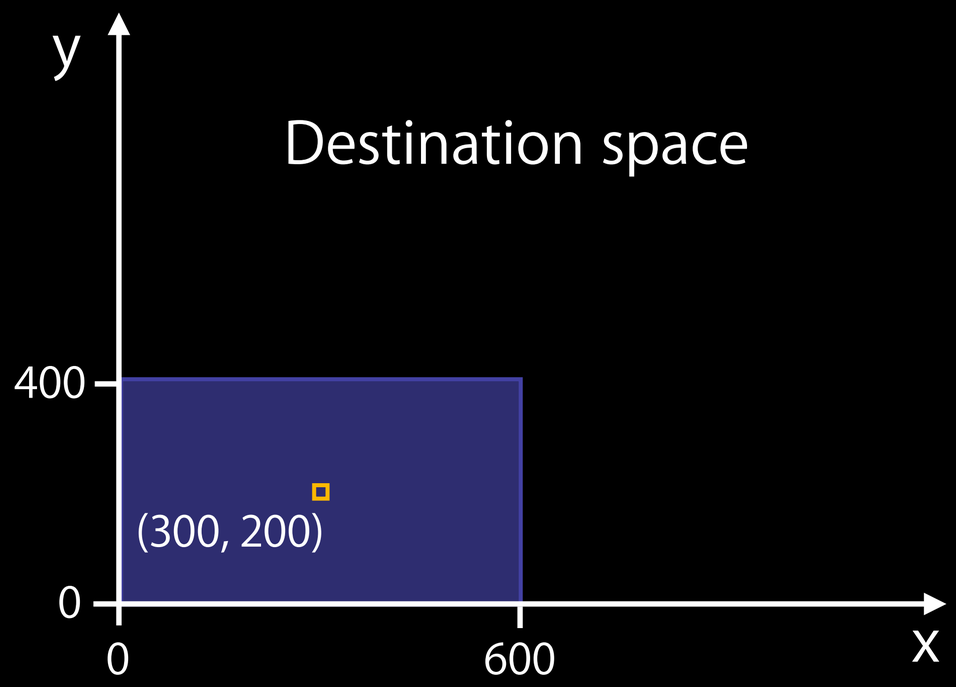


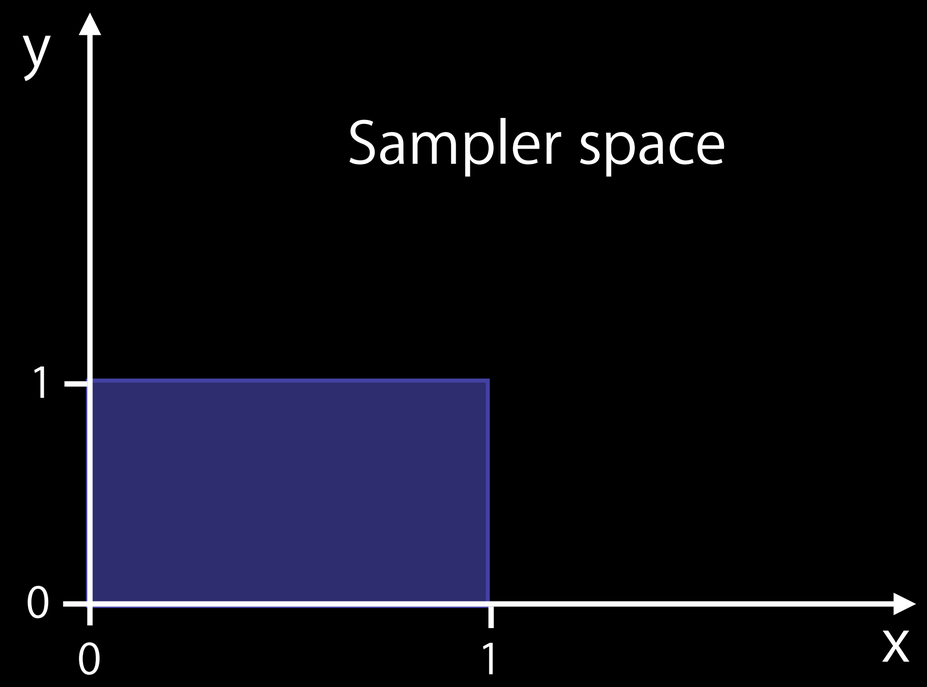


Basic Principles of General Kernels

Destination space vs. Sampler space

```
kernel vec4 do_something (sampler image) {
    vec2 dc = destCoord();
    vec2 offset = vec2(0.0, 2.0);
    return sample (image, samplerTransform (image, dc + offset));
}
```

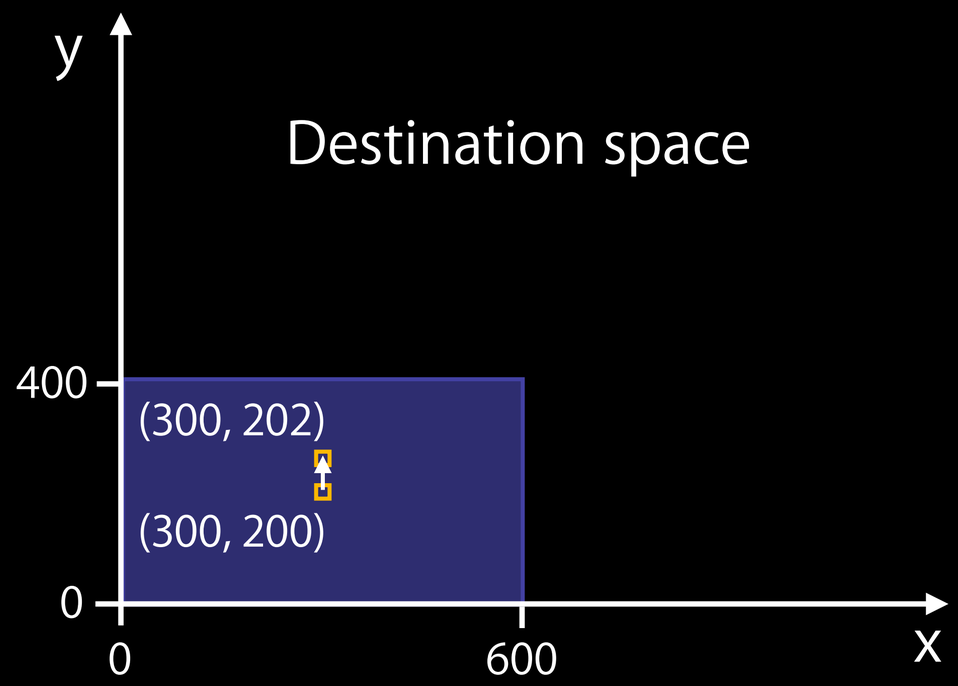


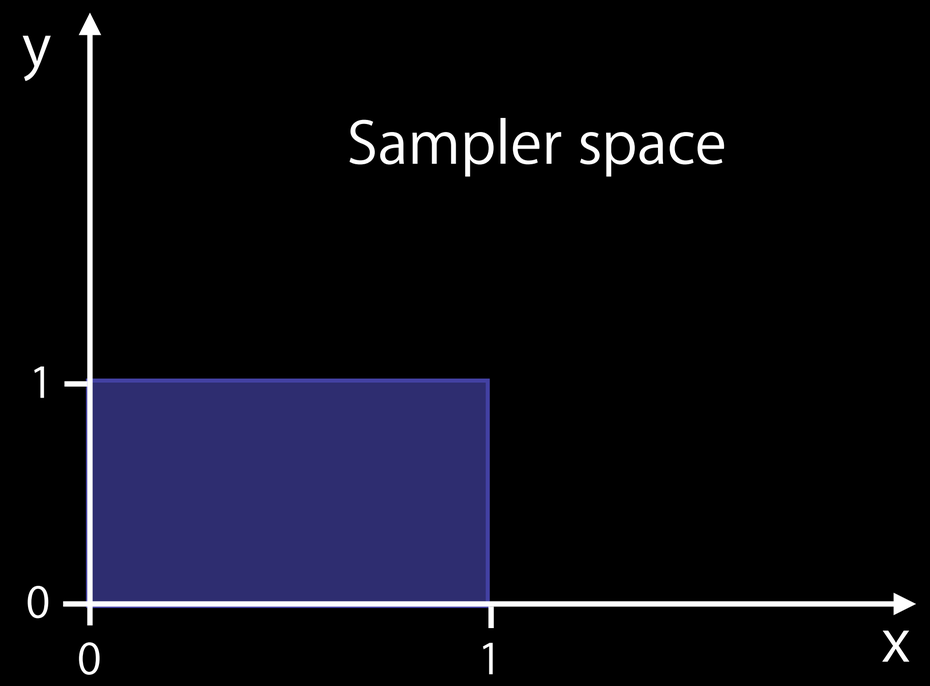


Basic Principles of General Kernels

Destination space vs. Sampler space

```
kernel vec4 do_something (sampler image) {
    vec2 dc = destCoord();
    vec2 offset = vec2(0.0, 2.0);
    return sample (image, samplerTransform (image, dc + offset));
}
```

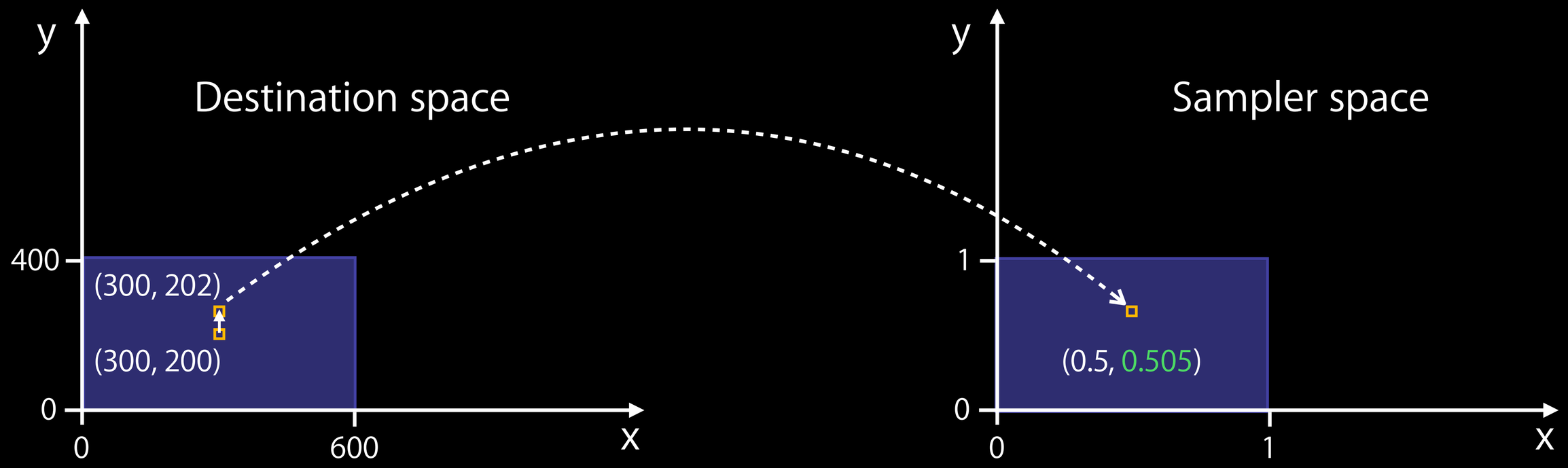




Basic Principles of General Kernels

Destination space vs. Sampler space

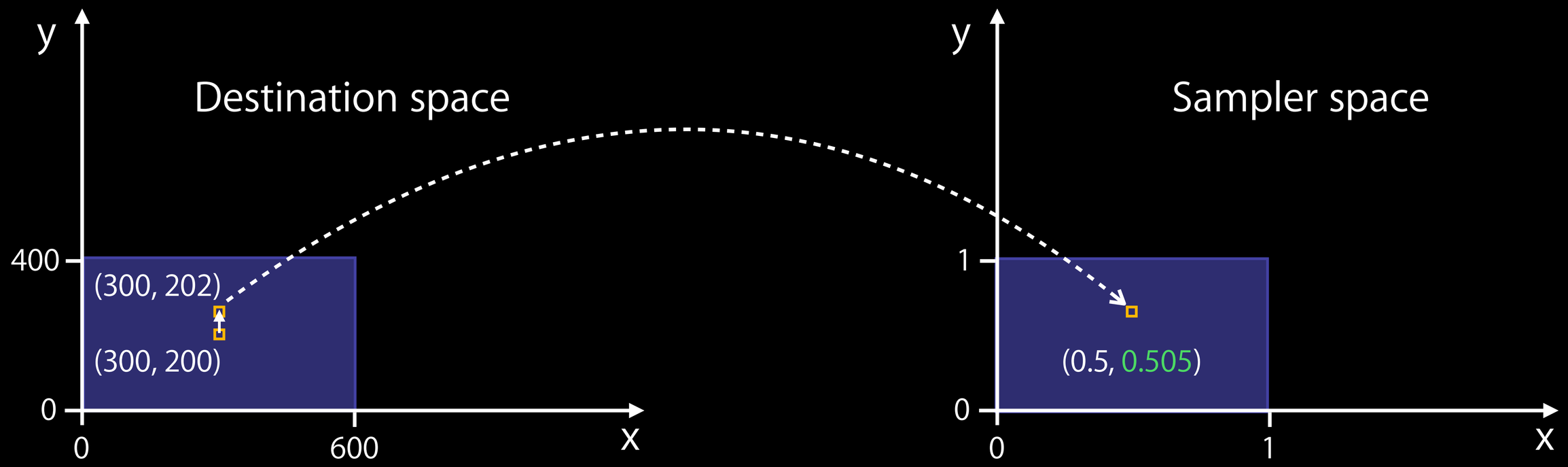
```
kernel vec4 do_something (sampler image) {
    vec2 dc = destCoord();
    vec2 offset = vec2(0.0, 2.0);
    return sample (image, samplerTransform (image, dc + offset));
}
```



Basic Principles of General Kernels

Destination space vs. Sampler space

```
kernel vec4 do_something (sampler image) {
    vec2 dc = destCoord();
    vec2 offset = vec2(0.0, 2.0);
    return sample (image, samplerTransform (image, dc + offset));
}
```



Examples

CIKernel

Motion Blur

Kernel requires multiple samples

Motion Blur

Kernel requires multiple samples

Motion Blur

Kernel requires multiple samples

Motion Blur

Kernel requires multiple samples

Motion Blur

CIKernel

```
kernel vec4 motionBlur (sampler image, vec2 velocity) {
    const int NUM_SAMPLES = 10; // per direction

    vec4 s = vec4(0.0);
    vec2 dc = destCoord(), offset = -velocity;

    for (int i=0; i < (NUM_SAMPLES * 2 + 1); i++) {
        s += sample (image, samplerTransform (image, dc + offset));
        offset += velocity / float(NUM_SAMPLES);
    }

    return s / float((NUM_SAMPLES * 2 + 1));
}
```

Motion Blur

CIKernel

```
kernel vec4 motionBlur (sampler image, vec2 velocity) {
    const int NUM_SAMPLES = 10; // per direction

    vec4 s = vec4(0.0);
    vec2 dc = destCoord(), offset = -velocity;

    for (int i=0; i < (NUM_SAMPLES * 2 + 1); i++) {
        s += sample (image, samplerTransform (image, dc + offset));
        offset += velocity / float(NUM_SAMPLES);
    }

    return s / float((NUM_SAMPLES * 2 + 1));
}
```

Motion Blur

CIKernel

```
kernel vec4 motionBlur (sampler image, vec2 velocity) {
    const int NUM_SAMPLES = 10; // per direction

    vec4 s = vec4(0.0);
    vec2 dc = destCoord(), offset = -velocity;

    for (int i=0; i < (NUM_SAMPLES * 2 + 1); i++) {
        s += sample (image, samplerTransform (image, dc + offset));
        offset += velocity / float(NUM_SAMPLES);
    }

    return s / float((NUM_SAMPLES * 2 + 1));
}
```

Motion Blur

CIKernel

```
kernel vec4 motionBlur (sampler image, vec2 velocity) {
    const int NUM_SAMPLES = 10; // per direction

    vec4 s = vec4(0.0);
    vec2 dc = destCoord(), offset = -velocity;

    for (int i=0; i < (NUM_SAMPLES * 2 + 1); i++) {
        s += sample (image, samplerTransform (image, dc + offset));
        offset += velocity / float(NUM_SAMPLES);
    }

    return s / float((NUM_SAMPLES * 2 + 1));
}
```

Motion Blur

CIKernel

```
kernel vec4 motionBlur (sampler image, vec2 velocity) {
    const int NUM_SAMPLES = 10; // per direction

    vec4 s = vec4(0.0);
    vec2 dc = destCoord(), offset = -velocity;

    for (int i=0; i < (NUM_SAMPLES * 2 + 1); i++) {
        s += sample (image, samplerTransform (image, dc + offset));
        offset += velocity / float(NUM_SAMPLES);
    }

    return s / float((NUM_SAMPLES * 2 + 1));
}
```

Motion Blur

CIKernel

```
kernel vec4 motionBlur (sampler image, vec2 velocity) {
    const int NUM_SAMPLES = 10; // per direction

    vec4 s = vec4(0.0);
    vec2 dc = destCoord(), offset = -velocity;

    for (int i=0; i < (NUM_SAMPLES * 2 + 1); i++) {
        s += sample (image, samplerTransform (image, dc + offset));
        offset += velocity / float(NUM_SAMPLES);
    }

    return s / float((NUM_SAMPLES * 2 + 1));
}
```

Motion Blur

CIFilter subclass

```
- (CIKernel *)myKernel {
    static CIKernel *kernel = [CIKernel kernelWithString:source];
    return kernel;
}

- (CIImage *)outputImage {
    float r = inputRadius.floatValue, a = inputAngle.floatValue;
    CIVector *velocity = [CIVector vectorWithX:r*cos(a) Y:r*sin(a)];

    return [[self myKernel] applyWithExtent:DOD
                                roiCallback:^(int index, CGRect rect) {
                                              return regionOf(rect,velocity); }
                                  arguments: @[inputImage, velocity] ];
}
```

Motion Blur

CIFilter subclass

```
- (CIKernel *)myKernel {
    static CIKernel *kernel = [CIKernel kernelWithString:source];
    return kernel;
}

- (CIImage *)outputImage {
    float r = inputRadius.floatValue, a = inputAngle.floatValue;
    CIVector *velocity = [CIVector vectorWithX:r*cos(a) Y:r*sin(a)];

    return [[self myKernel] applyWithExtent:DOD
                                roiCallback:^(int index, CGRect rect) {
                                                return regionOf(rect,velocity); }
                                  arguments: @[inputImage, velocity] ];
}
```

Motion Blur

CIFilter subclass

```
kernel vec4 motionBlur (sampler image, vec2 velocity) {
     const int NUM_SAMPLES = 10; // per direction
     vec4 s = vec4(0.0);
     vec2 dc = destCoord(), offset = -velocity;
     for (int i=0; i < (NUM_SAMPLES * 2 + 1); i++) {
          s += sample (image, samplerTransform (image, dc + offset));
          offset += velocity / float(NUM_SAMPLES);
     }
     return s / float((NUM_SAMPLES * 2 + 1));
}
```

```
- (CIKernel *)myKernel {
    static CIKernel *kernel = [CIKernel kernelWithString:source];
    return kernel;
}

- (CIImage *)outputImage {
    float r = inputRadius.floatValue, a = inputAngle.floatValue;
    CIVector *velocity = [CIVector vectorWithX:r*cos(a) Y:r*sin(a)];

    return [[self myKernel] applyWithExtent:DOD
                                roiCallback:^(int index, CGRect rect) {
                                              return regionOf(rect,velocity); }
                                  arguments: @[inputImage, velocity] ];
}
```

Motion Blur

CIFilter subclass

```
- (CIKernel *)myKernel {
    static CIKernel *kernel = [CIKernel kernelWithString:source];
    return kernel;
}

- (CIImage *)outputImage {
    float r = inputRadius.floatValue, a = inputAngle.floatValue;
    CIVector *velocity = [CIVector vectorWithX:r*cos(a) Y:r*sin(a)];

    return [[self myKernel] applyWithExtent:DOD
                                roiCallback:^(int index, CGRect rect) {
                                                return regionOf(rect,velocity); }
                                  arguments: @[inputImage, velocity] ];
}
```

Motion Blur

CIFilter subclass

```
- (CIKernel *)myKernel {
    static CIKernel *kernel = [CIKernel kernelWithString:source];
    return kernel;
}

- (CIImage *)outputImage {
    float r = inputRadius.floatValue, a = inputAngle.floatValue;
    CIVector *velocity = [CIVector vectorWithX:r*cos(a) Y:r*sin(a)];

    return [[self myKernel] applyWithExtent:DOD
                                roiCallback:^(int index, CGRect rect) {
                                                return regionOf(rect,velocity); }
                                  arguments: @[inputImage, velocity] ];
}
```

Motion Blur

CIFilter subclass

```
- (CIKernel *)myKernel {
    static CIKernel *kernel = [CIKernel kernelWithString:source];
    return kernel;
}

- (CIImage *)outputImage {
    float r = inputRadius.floatValue, a = inputAngle.floatValue;
    CIVector *velocity = [CIVector vectorWithX:r*cos(a) Y:r*sin(a)];

    return [[self myKernel] applyWithExtent:DOD
                                roiCallback:^(int index, CGRect rect) {
                                                return regionOf(rect,velocity); }
                                  arguments: @[inputImage, velocity] ];
}
```

Motion Blur

DOD

Motion Blur

DOD

Motion Blur

DOD

Motion Blur

DOD

Motion Blur

DOD

Motion Blur

DOD


```
DOD = CGRectInset (inputImage.extent, -abs(velocity.X), -abs(velocity.Y));
```

Motion Blur

ROI

Motion Blur

ROI

Motion Blur

ROI

Motion Blur

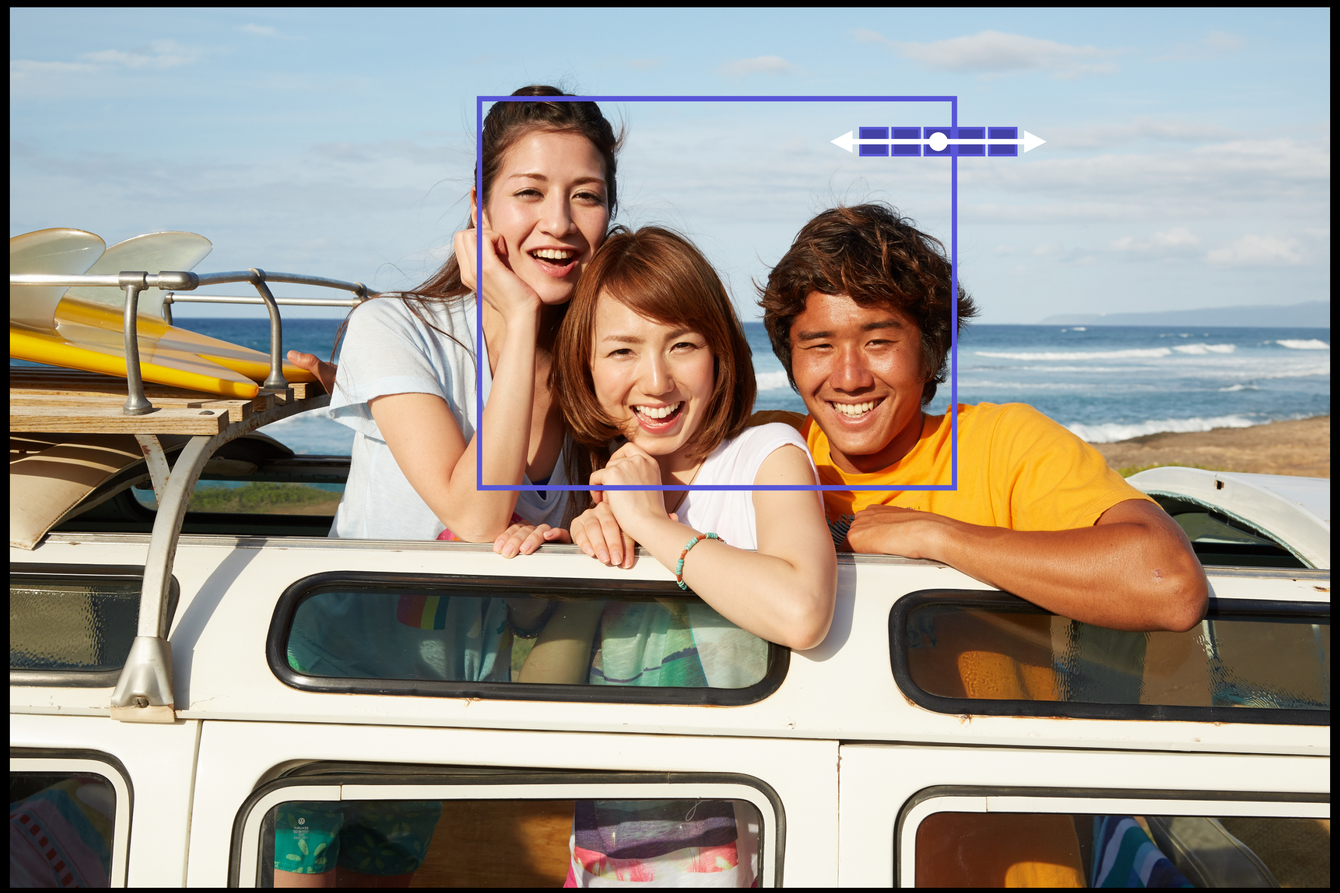
ROI

Motion Blur

ROI

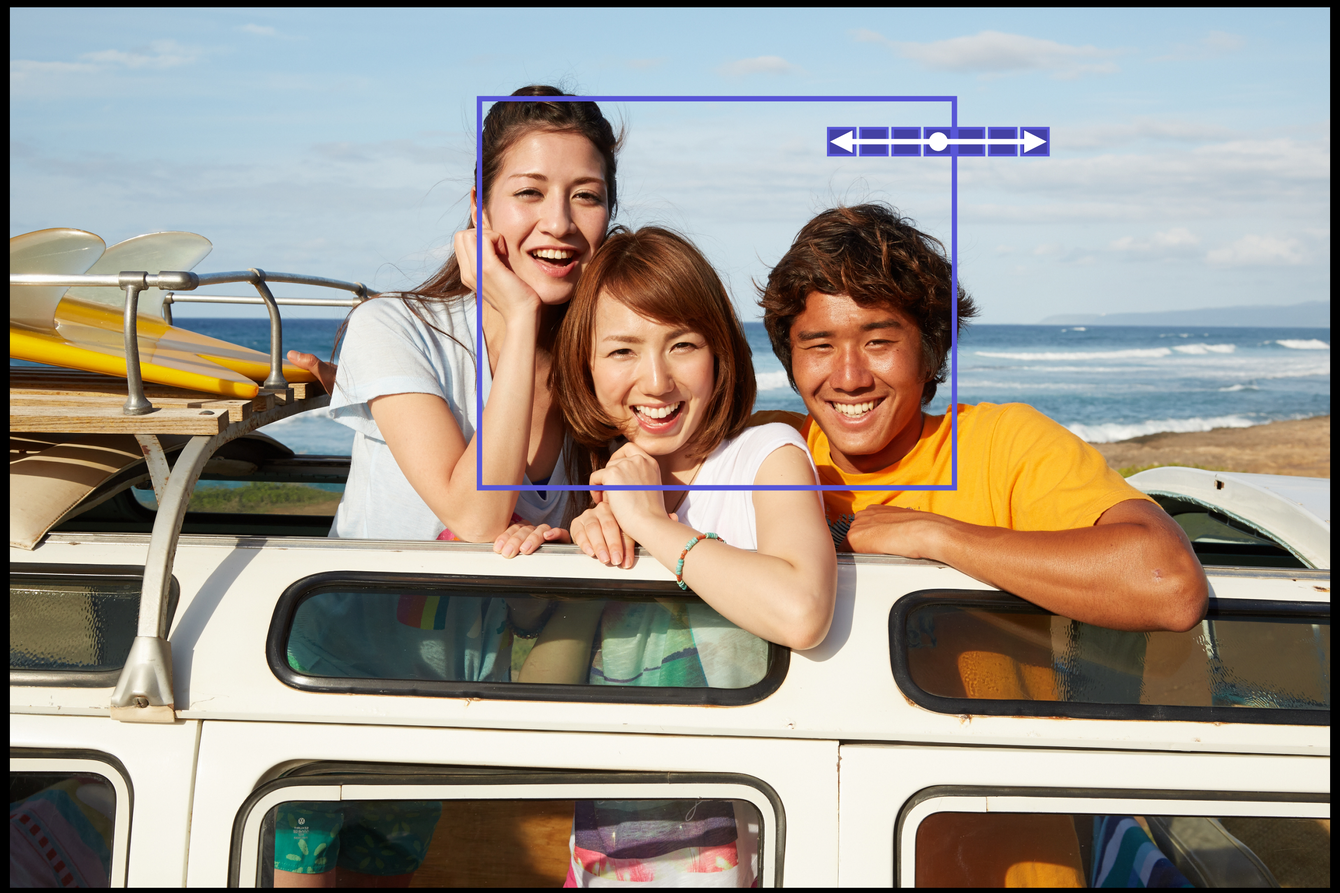
Motion Blur

ROI



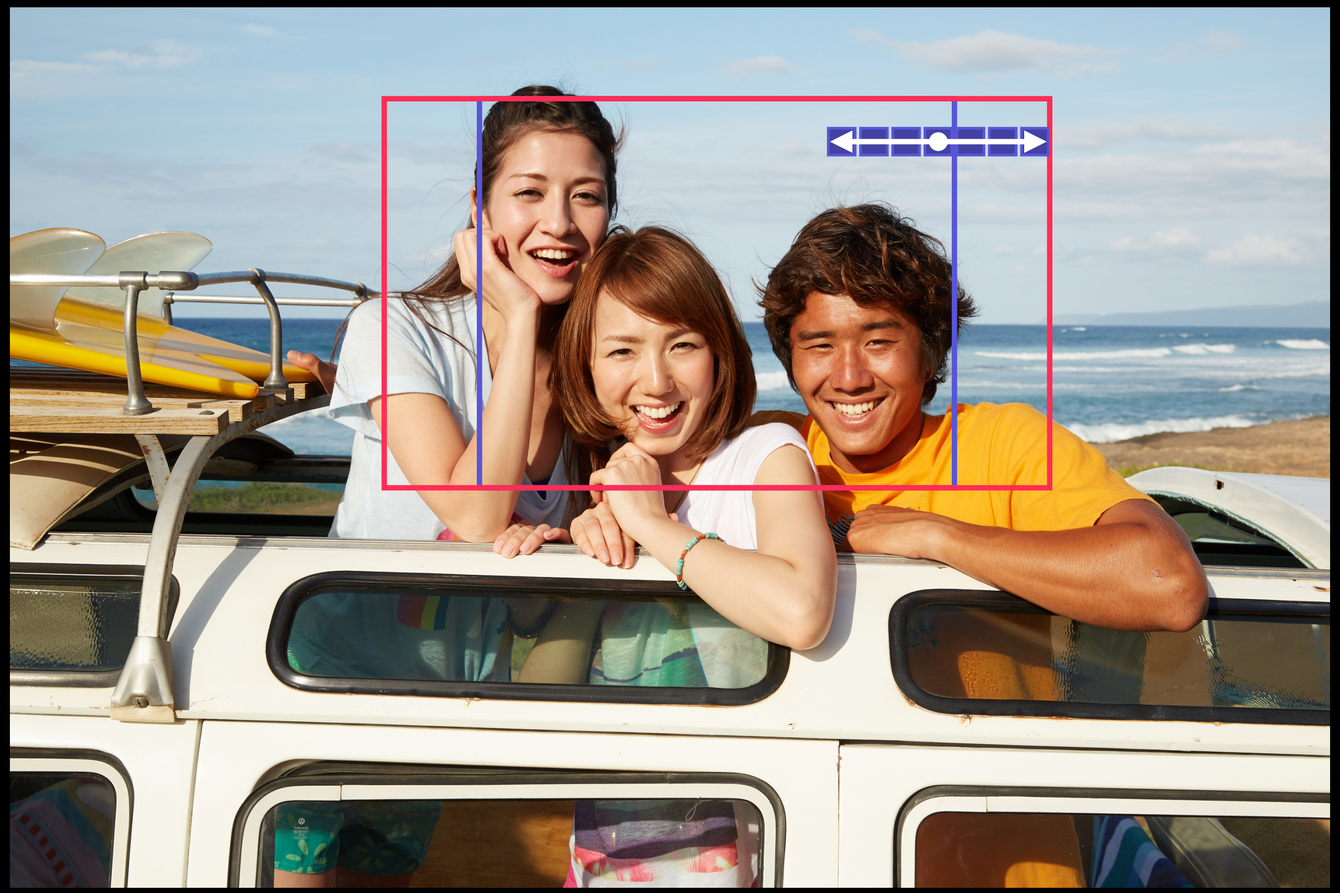
Motion Blur

ROI



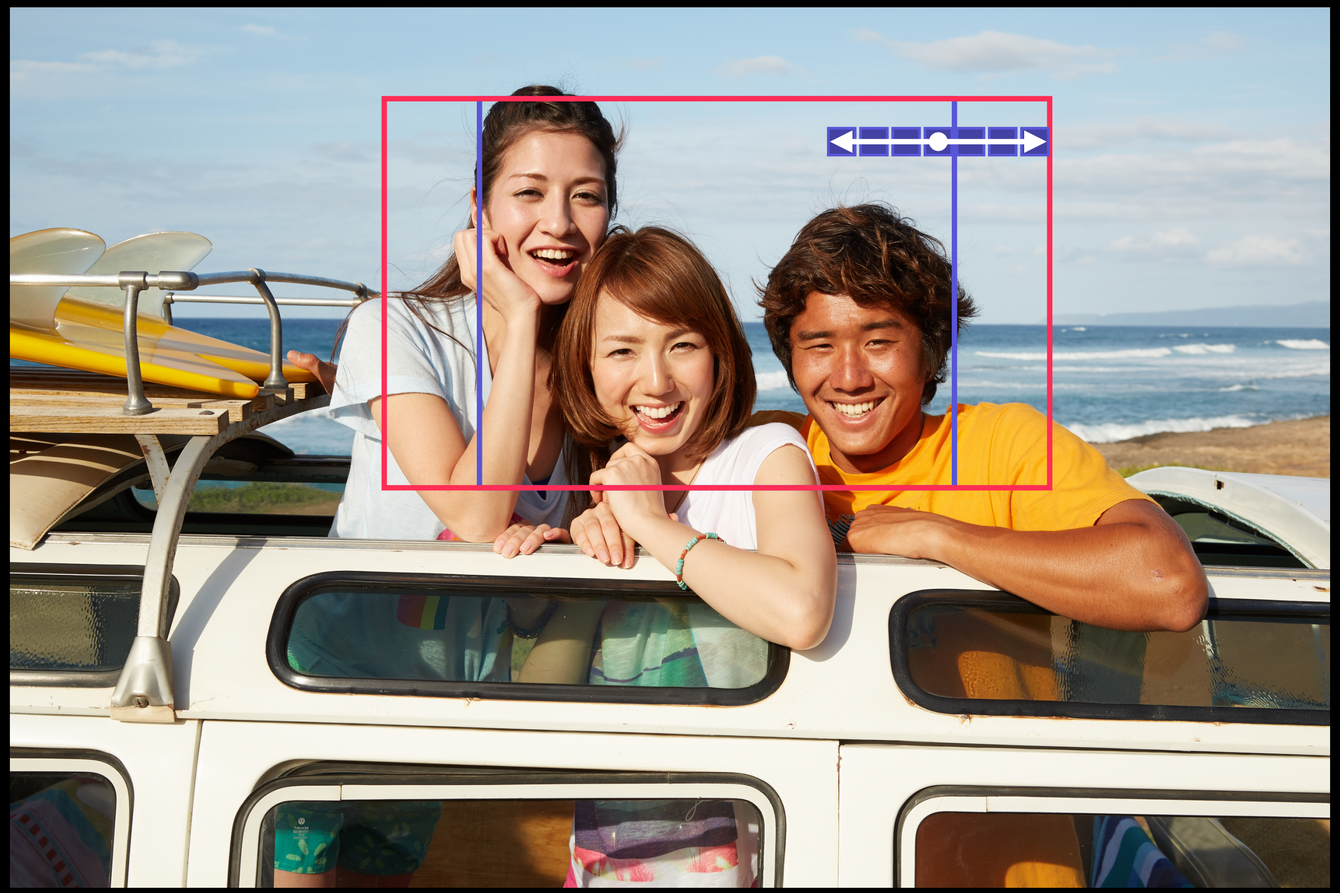
Motion Blur

ROI



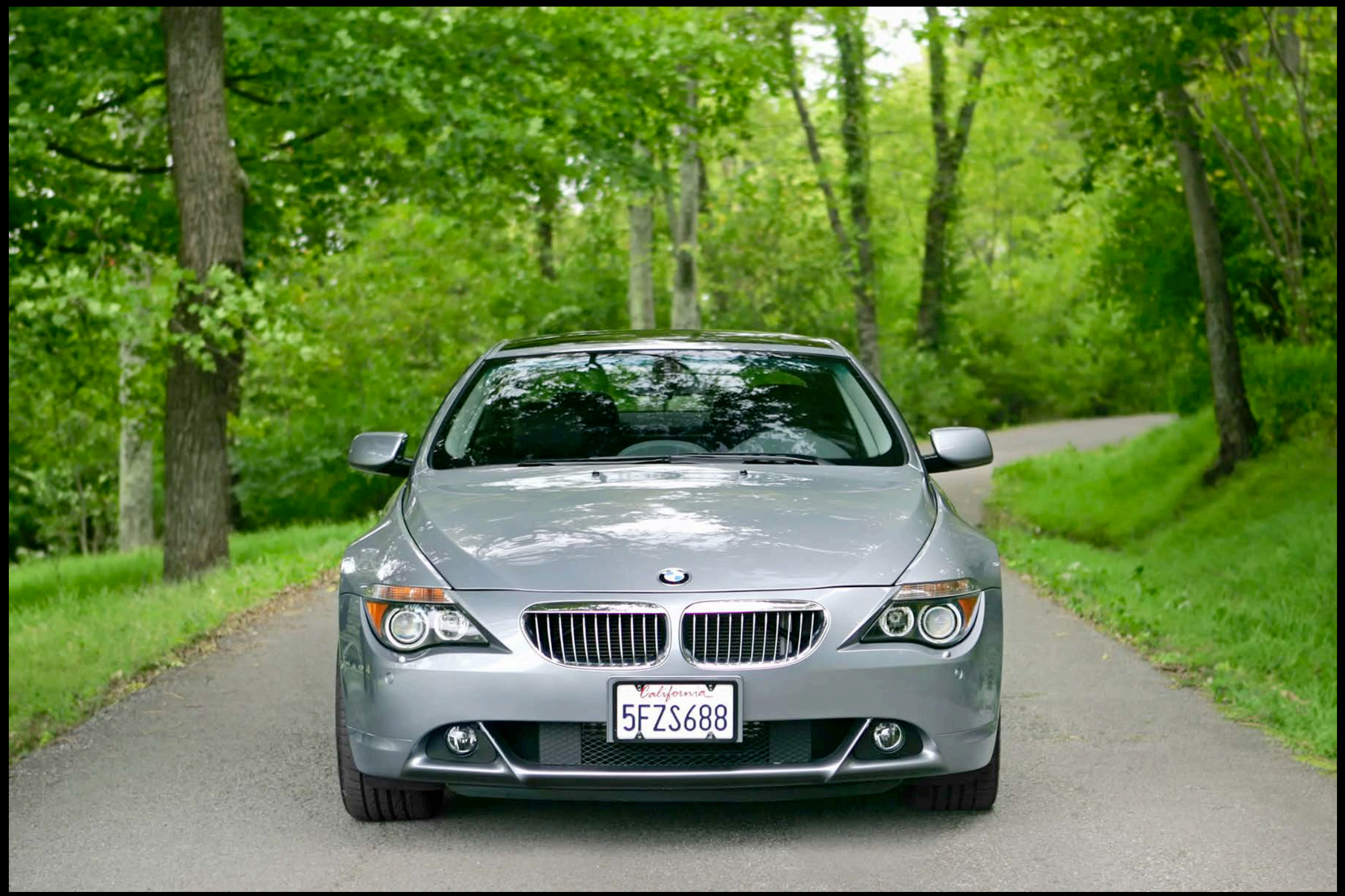
Motion Blur

ROI

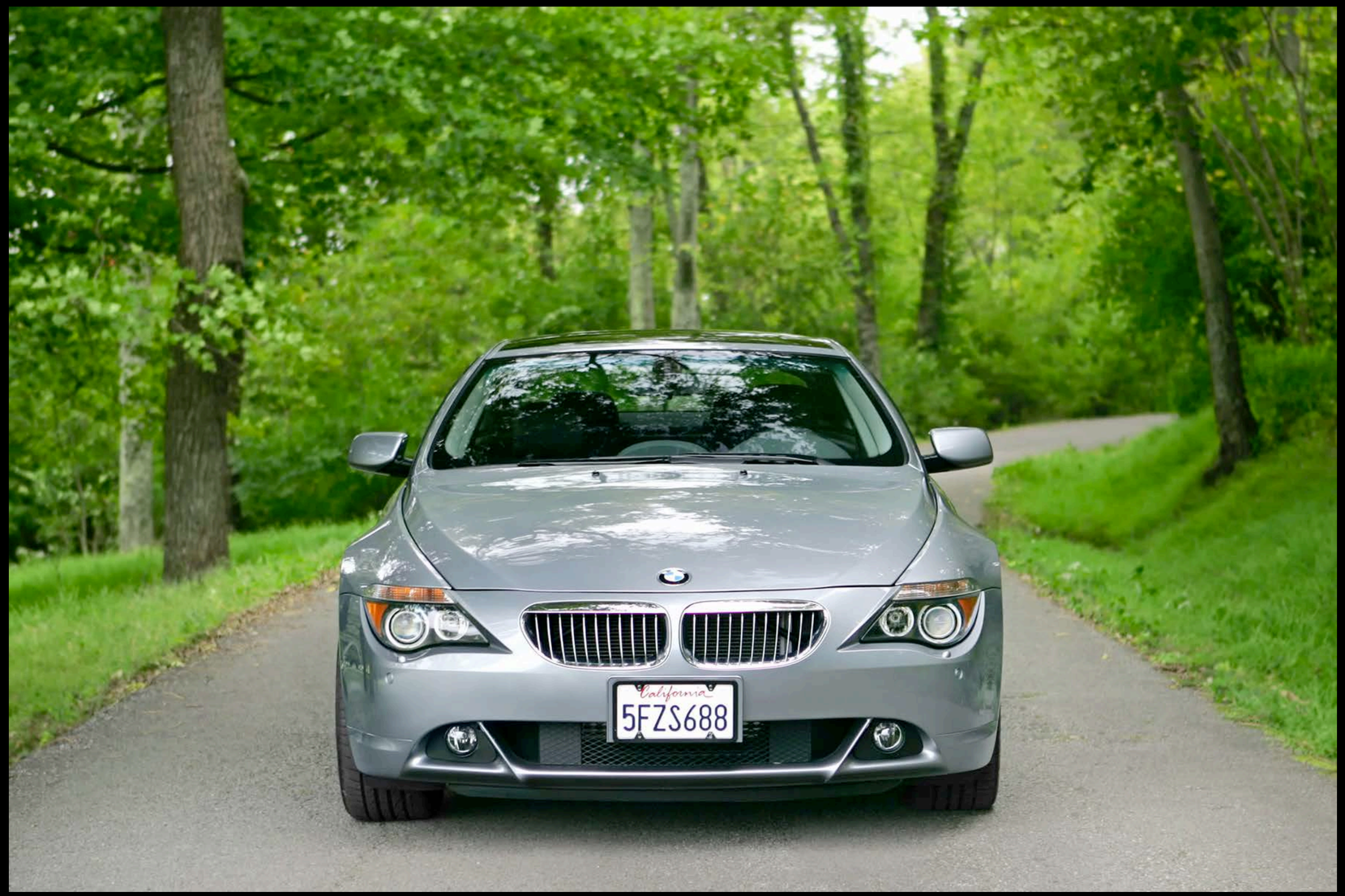


```
CGRect regionOf (CGRect rect, CIVector *velocity) {
    return CGRectInset (rect, -abs(velocity.X), -abs(velocity.Y));
}
```

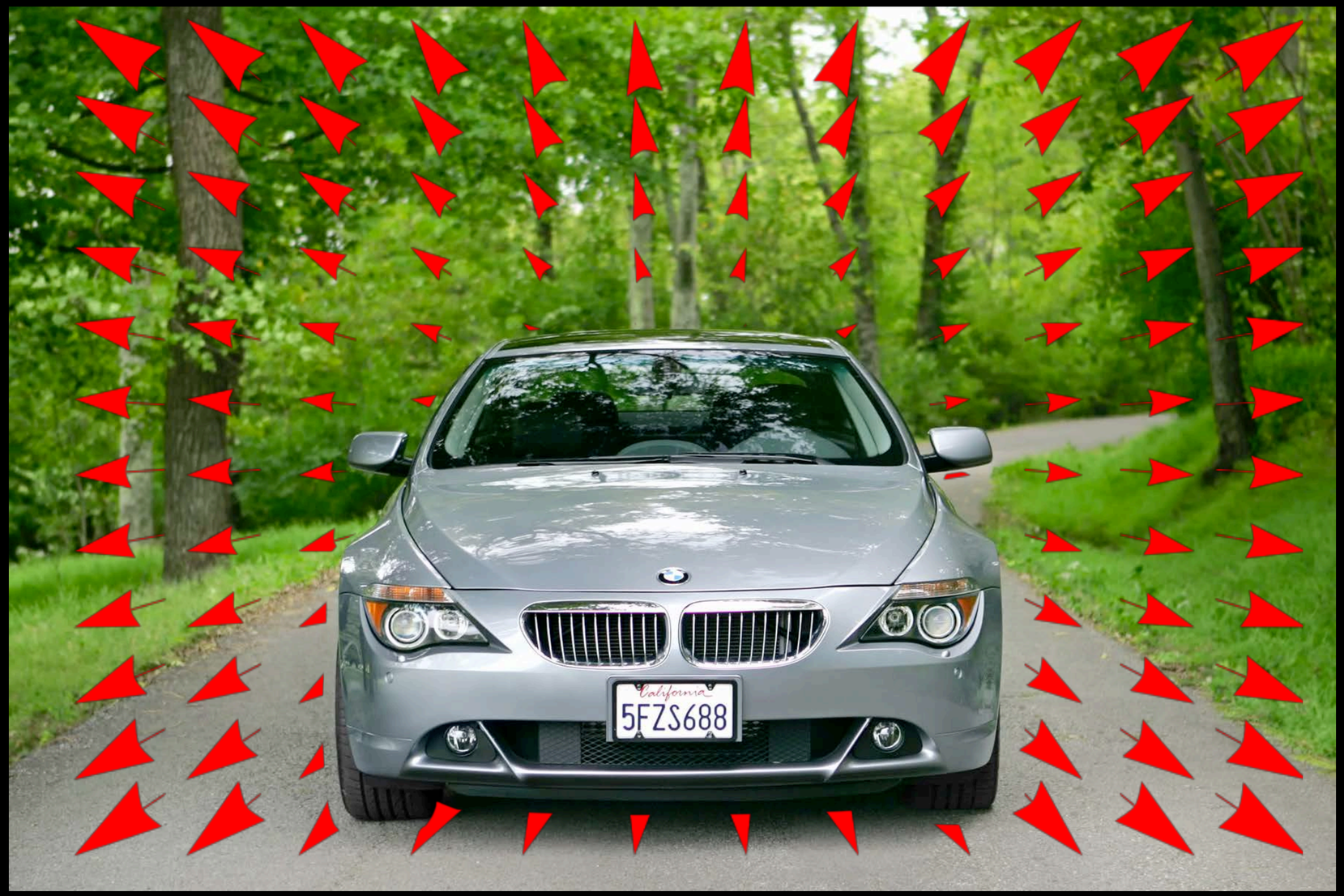
Motion Blur 2.0



Motion Blur 2.0



Motion Blur 2.0



Motion Blur with Masked Vector Field

Kernel contains dependent texture read

Motion Blur with Masked Vector Field

Kernel contains dependent texture read

Motion Blur with Masked Vector Field

Kernel contains dependent texture read

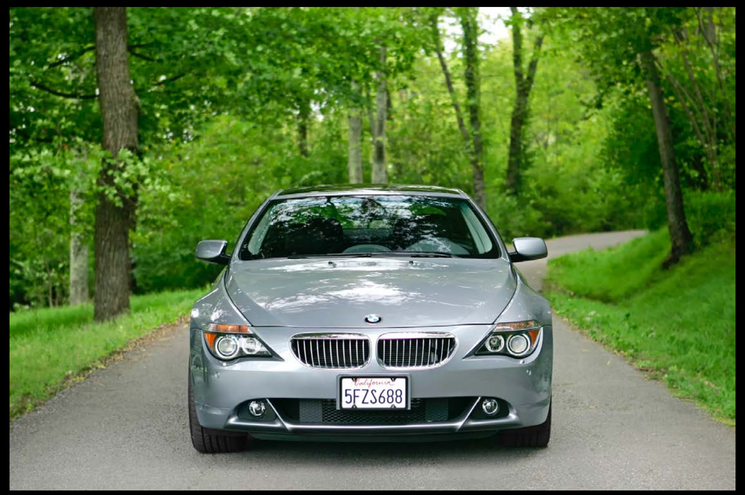
Motion Blur with Masked Vector Field

Kernel contains dependent texture read

CIKernel

Motion Blur with Masked Vector Field

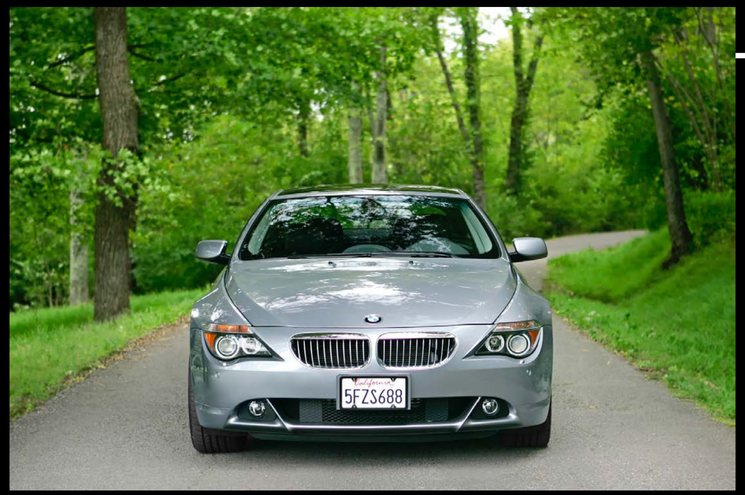
Kernel contains dependent texture read



CIKernel

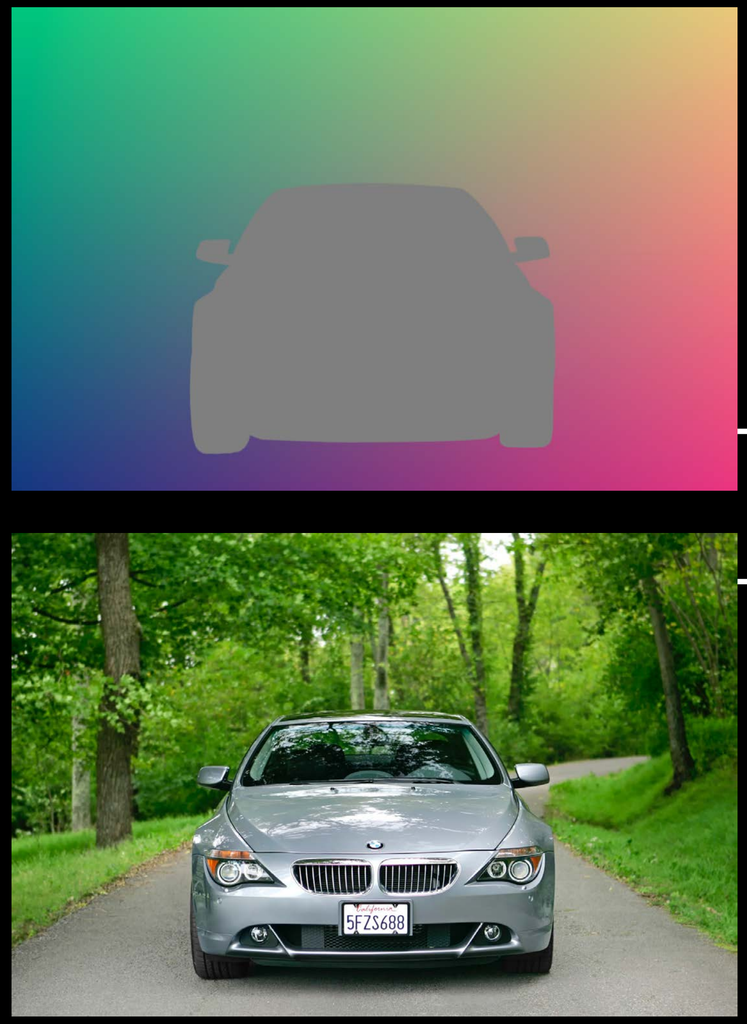
Motion Blur with Masked Vector Field

Kernel contains dependent texture read



Motion Blur with Masked Vector Field

Kernel contains dependent texture read



CIKernel

Motion Blur with Masked Vector Field

CIKernel

```
kernel vec4 motionBlur (sampler image, vec2 velocity) {
    const int NUM_SAMPLES = 10; // per direction

    vec4 s = vec4(0.0);
    vec2 dc = destCoord(), offset = -velocity;

    for (int i=0; i < (NUM_SAMPLES * 2 + 1); i++) {
        s += sample (image, samplerTransform (image, dc + offset));
        offset += velocity / float(NUM_SAMPLES);
    }

    return s / float((NUM_SAMPLES * 2 + 1));
}
```

Motion Blur with Masked Vector Field

CIKernel

```
kernel vec4 motionBlur (sampler image, vec2 velocity) {
    const int NUM_SAMPLES = 10; // per direction

    vec4 s = vec4(0.0);
    vec2 dc = destCoord(), offset = -velocity;

    for (int i=0; i < (NUM_SAMPLES * 2 + 1); i++) {
        s += sample (image, samplerTransform (image, dc + offset));
        offset += velocity / float(NUM_SAMPLES);
    }

    return s / float((NUM_SAMPLES * 2 + 1));
}
```

Motion Blur with Masked Vector Field

CIKernel

```
kernel vec4 motionBlur (sampler image, vec2 velocity) {
    const int NUM_SAMPLES = 10; // per direction

    vec4 s = vec4(0.0);
    vec2 dc = destCoord(), offset = -velocity;

    for (int i=0; i < (NUM_SAMPLES * 2 + 1); i++) {
        s += sample (image, samplerTransform (image, dc + offset));
        offset += velocity / float(NUM_SAMPLES);
    }
    return s / float(NUM_SAMPLES);
}

kernel vec4 motionBlurWithMask (sampler image, sampler mask, float radius) {
    // read per-pixel directional vector from mask image first
    vec2 dir = sample(mask, samplerCoord(mask)).rg;
    dir = dir * 2.0 - 1.0; // de-normalize vector from [0,1] to [-1,+1]

    vec2 velocity = dir * radius;

    return _motionBlur (image, velocity);
}
```

Motion Blur with Masked Vector Field

CIKernel

```
vec4        _motionBlur (sampler image, vec2 velocity) {
    // apply regular motion blur effect
}

kernel vec4 motionBlurWithMask (sampler image, sampler mask, float radius) {
    // read per-pixel directional vector from mask image first
    vec2 dir = sample (mask, samplerCoord(mask)).rg;
    dir = dir * 2.0 - 1.0; // de-normalize vector from [0,1] to [-1,+1]

    vec2 velocity = dir * radius;

    return _motionBlur (image, velocity);
}
```

Motion Blur with Masked Vector Field

CIKernel

```
vec4       _motionBlur (sampler image, vec2 velocity) {
    // apply regular motion blur effect
}

kernel vec4 motionBlurWithMask (sampler image, sampler mask, float radius) {
    // read per-pixel directional vector from mask image first
    vec2 dir = sample (mask, samplerCoord(mask)).rg;
    dir = dir * 2.0 - 1.0; // de-normalize vector from [0,1] to [-1,+1]

    vec2 velocity = dir * radius;

    return _motionBlur (image, velocity);
}
```

Motion Blur with Masked Vector Field

CIKernel

```
vec4       _motionBlur (sampler image, vec2 velocity) {
    // apply regular motion blur effect
}

kernel vec4 motionBlurWithMask (sampler image, sampler mask, float radius) {
    // read per-pixel directional vector from mask image first
    vec2 dir = sample (mask, samplerCoord(mask)).rg;
    dir = dir * 2.0 - 1.0; // de-normalize vector from [0,1] to [-1,+1]

    vec2 velocity = dir * radius;

    return _motionBlur (image, velocity);
}
```

Motion Blur with Masked Vector Field

CIKernel

```
vec4       _motionBlur (sampler image, vec2 velocity) {
    // apply regular motion blur effect
}

kernel vec4 motionBlurWithMask (sampler image, sampler mask, float radius) {
    // read per-pixel directional vector from mask image first
    vec2 dir = sample (mask, samplerCoord(mask)).rg;
    dir = dir * 2.0 - 1.0; // de-normalize vector from [0,1] to [-1,+1]

    vec2 velocity = dir * radius;

    return _motionBlur (image, velocity);
}
```

Motion Blur with Masked Vector Field

CIFilter subclass

```
- (CIImage *)outputImage {
    float r = inputRadius.floatValue;
    CGRect DOD = CGRectInset (inputImage.extent, -r, -r);

    return [[self myKernel]
                applyWithExtent:DOD
                    roiCallback:^(int index, CGRect rect) {
                                return regionOf (index, rect, r); }
                      arguments: @[inputImage, maskImage, inputRadius] ];
}
```

Motion Blur with Masked Vector Field

CIFilter subclass

```
- (CIImage *)outputImage {
    float r = inputRadius.floatValue;
    CGRect DOD = CGRectInset (inputImage.extent, -r, -r);

    return [[self myKernel]
                applyWithExtent:DOD
                    roiCallback:^(int index, CGRect rect) {
                                return regionOf (index, rect, r); }
                      arguments: @[inputImage, maskImage, inputRadius] ];
}
```

Motion Blur with Masked Vector Field

CIFilter subclass

```
- (CIImage *)outputImage {
    float r = inputRadius.floatValue;
    CGRect DOD = CGRectInset (inputImage.extent, -r, -r);

    return [[self myKernel]
                applyWithExtent:DOD
                    roiCallback:^(int index, CGRect rect) {
                                return regionOf (index, rect, r); }
                      arguments: @[inputImage, maskImage, inputRadius] ];
}
```

Motion Blur with Masked Vector Field

ROI for multiple images

```
CGRect regionOf (int index, CGRect rect, float radius) {
    if (index == 0) {
        return CGRectInset (rect, -radius, -radius); // ROI for input image
    }
    else if (index == 1) {
        return rect; // ROI for mask image
    }

    return CGRectNull; // should not reach here
}
```

Motion Blur with Masked Vector Field

ROI for multiple images

```
CGRect regionOf (int index, CGRect rect, float radius) {
    if (index == 0) {
        return CGRectInset (rect, -radius, -radius); // ROI for input image
    }
    else if (index == 1) {
        return rect; // ROI for mask image
    }

    return CGRectNull; // should not reach here
}
```

Motion Blur with Masked Vector Field

ROI for multiple images

```
CGRect regionOf (int index, CGRect rect, float radius) {
    if (index == 0) {
        return CGRectInset (rect, -radius, -radius); // ROI for input image
    }
    else if (index == 1) {
        return rect; // ROI for mask image
    }

    return CGRectNull; // should not reach here
}
```

Portability of General Kernels

OS X and iOS

Desktop-class kernel type with the same language syntax and semantics as OS X

Portability of General Kernels

OS X and iOS

Desktop-class kernel type with the same language syntax and semantics as OS X

New built-in filters on iOS ported from OS X using general kernels

Portability of General Kernels

OS X and iOS

Desktop-class kernel type with the same language syntax and semantics as OS X

New built-in filters on iOS ported from OS X using general kernels

Glass Distortion

Portability of General Kernels

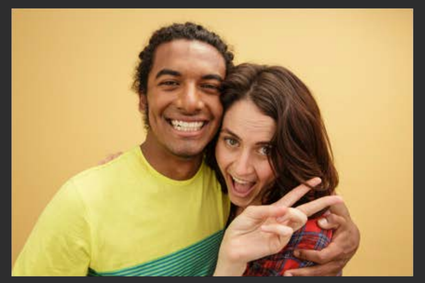
OS X and iOS

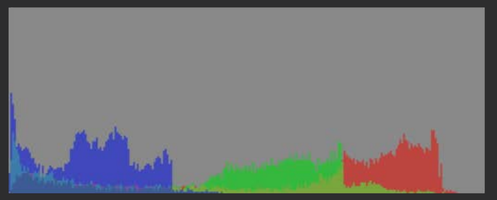
Desktop-class kernel type with the same language syntax and semantics as OS X

New built-in filters on iOS ported from OS X using general kernels

Glass Distortion

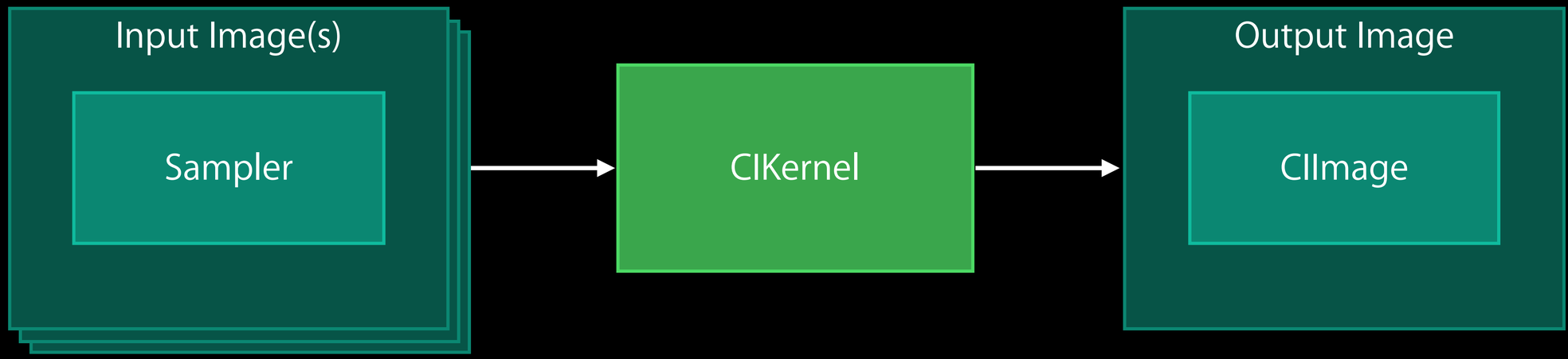
Histogram Display





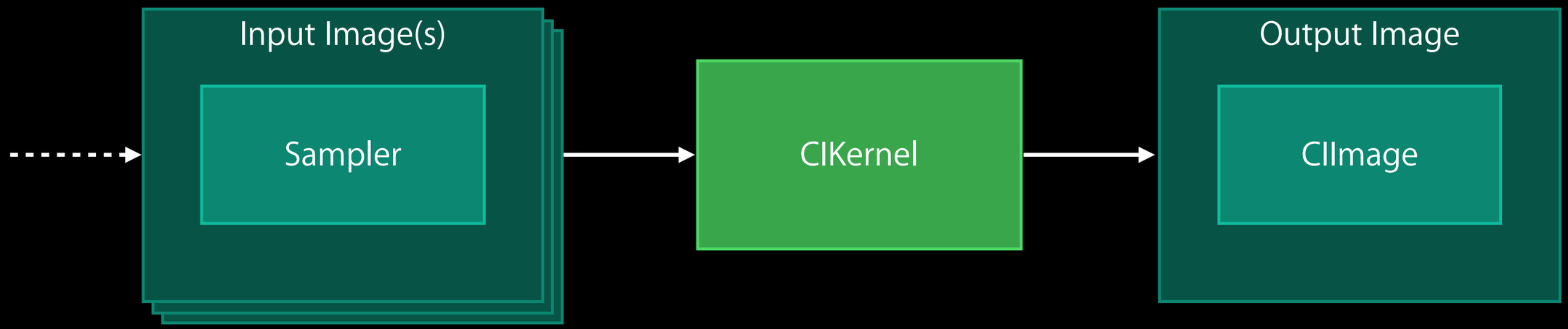
Performance Considerations

Multiple render passes



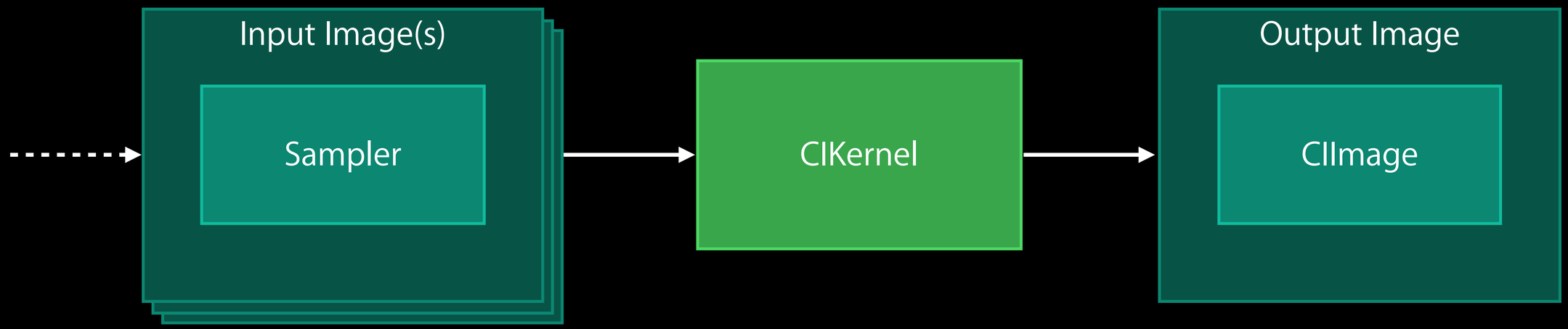
Performance Considerations

Multiple render passes



Performance Considerations

Multiple render passes



Each input image to a CIKernel adds an extra render pass

Intermediate Buffers

kCIContextWorkingFormat

NEW

	8-bit `kCIFormatRGBA8`	16-bit `kCIFormatRGBAh`
Null Working Space (Color Management Off)		
Default Working Space (Linear Rec. 709)	Needs Linear to sRGB	Needs 2X Memory

Intermediate Buffers

kCIContextWorkingFormat

NEW

	8-bit `kCIFormatRGBA8`	16-bit `kCIFormatRGBAh`
Null Working Space (Color Management Off)	✓	✗
Default Working Space (Linear Rec. 709)	Needs Linear to sRGB	Needs 2X Memory

Intermediate Buffers

kCIContextWorkingFormat

NEW

	8-bit `kCIFormatRGBA8`	16-bit `kCIFormatRGBAh`
Null Working Space (Color Management Off)	✓	✗
Default Working Space (Linear Rec. 709)	✓ Needs Linear to sRGB	Needs 2X Memory

Intermediate Buffers

kCIContextWorkingFormat

NEW

	8-bit `kCIFormatRGBA8`	16-bit `kCIFormatRGBAh`
Null Working Space (Color Management Off)	✓	✗
Default Working Space (Linear Rec. 709)	✓ Needs Linear to sRGB	✓ Needs 2X Memory

Square Kaleidoscope

CIKernel?

Square Kaleidoscope

CIKernel?

Square Kaleidoscope

CIKernel?

?

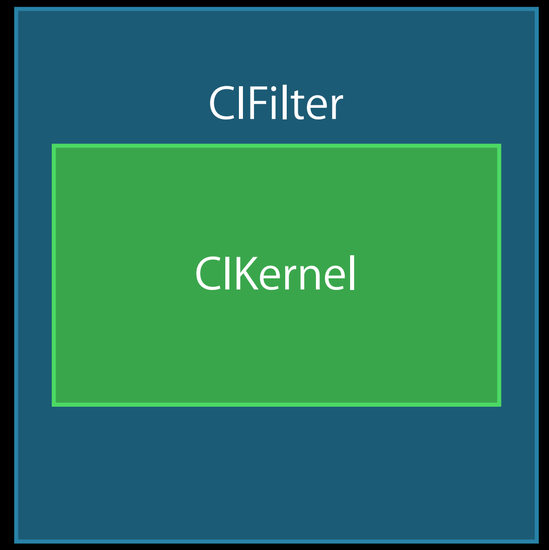
CIFilter

CIKernel

Square Kaleidoscope

CIKernel?

?



Square Kaleidoscope

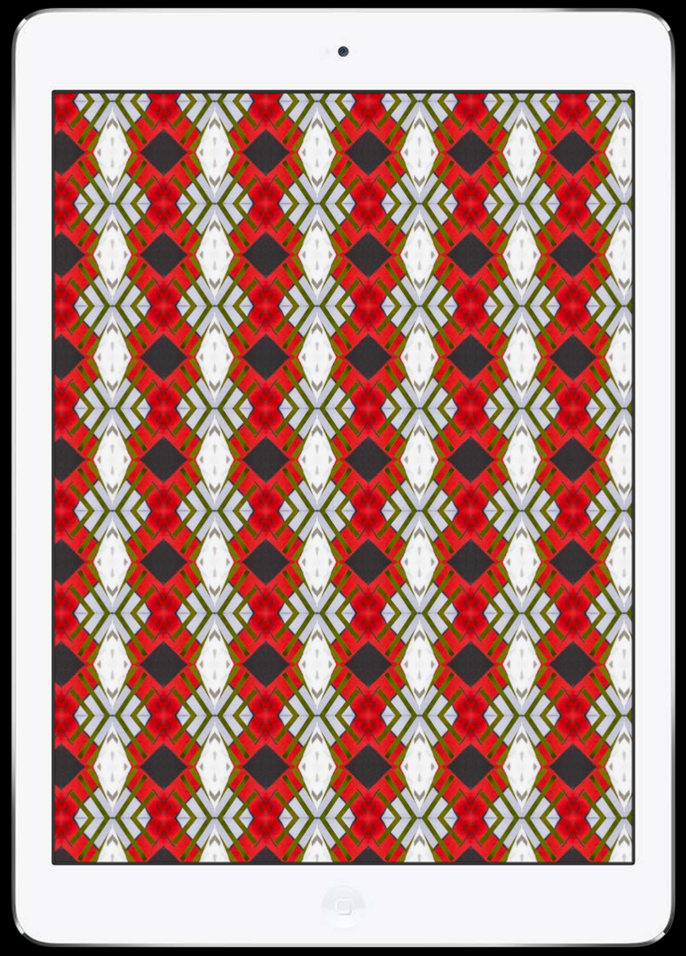
Better solution

?

CIFilter

Square Kaleidoscope

Better solution



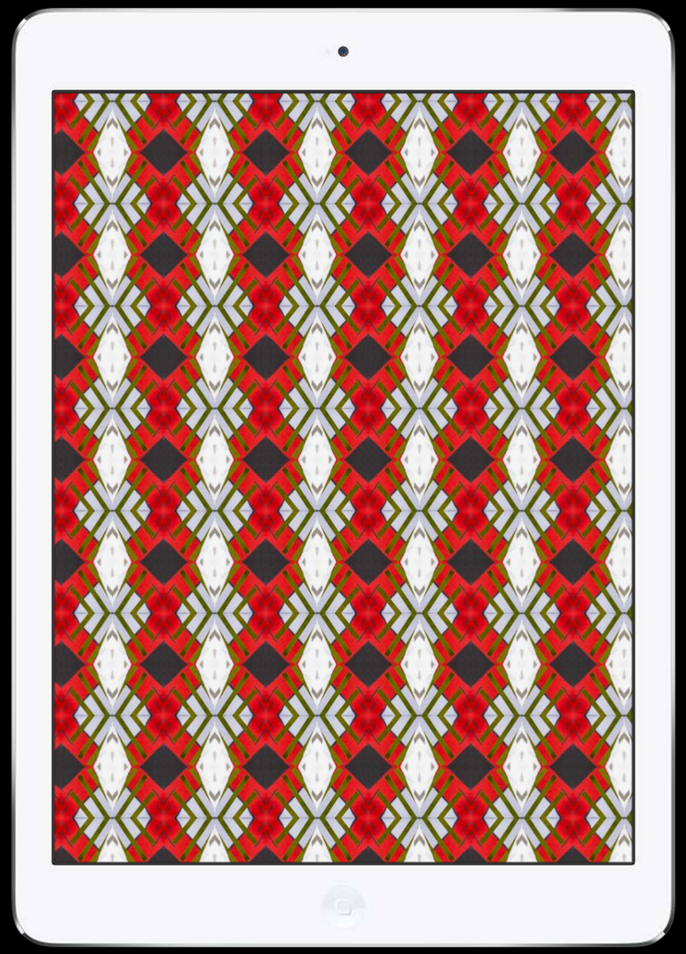
?

=

CIFilter

Square Kaleidoscope

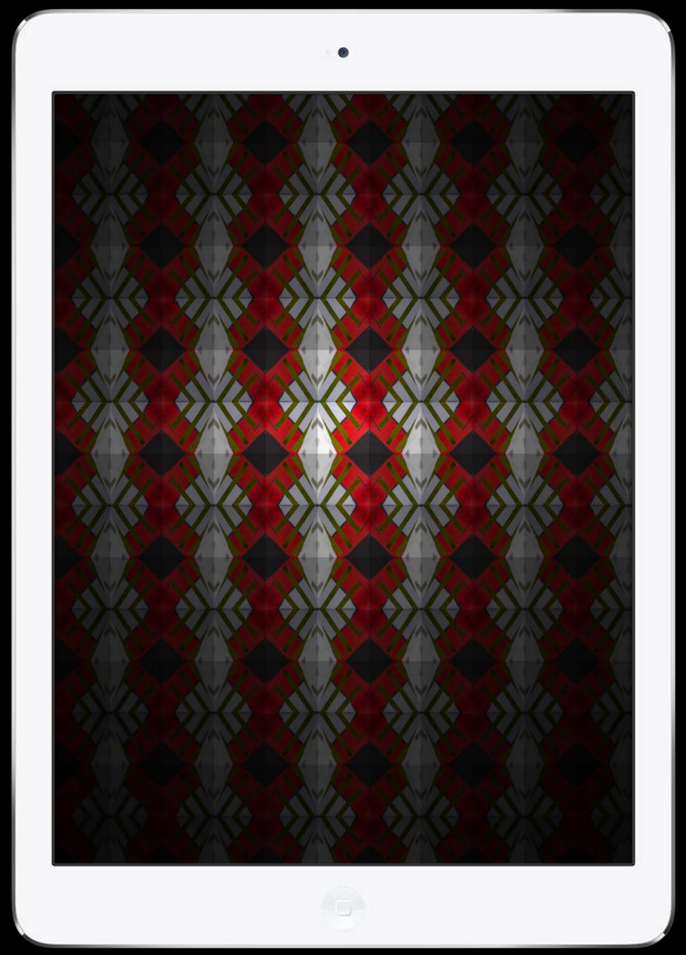
Better solution



?

Square Kaleidoscope

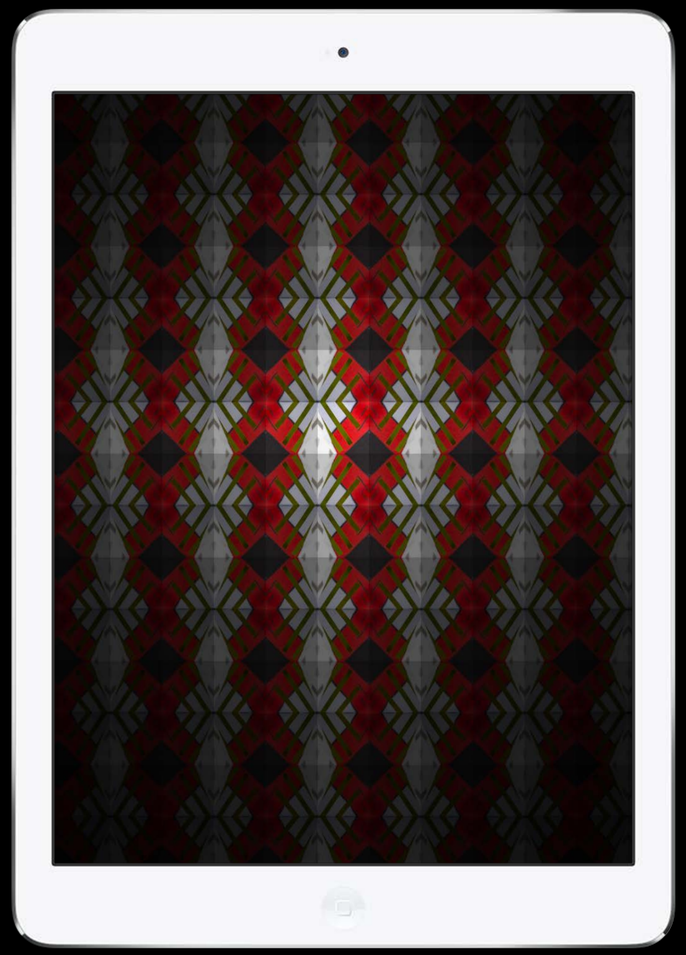
Better solution



?

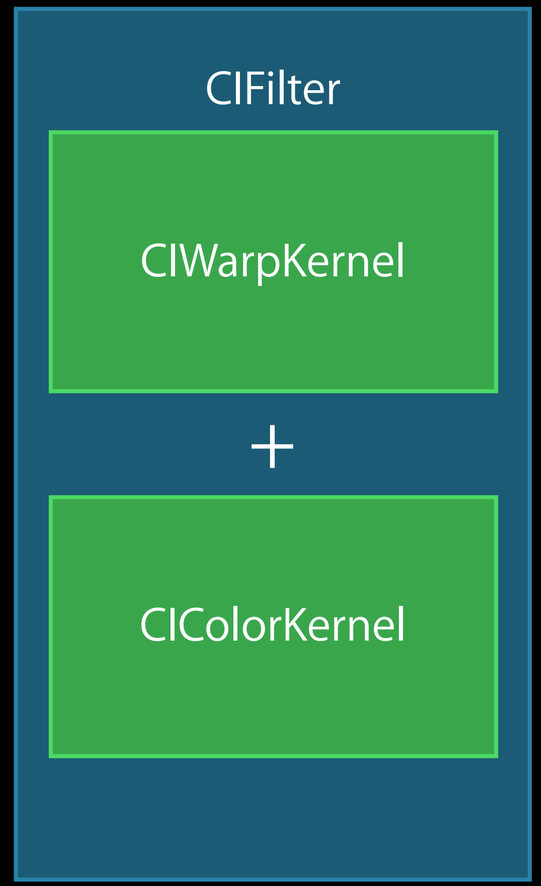
Square Kaleidoscope

Better solution



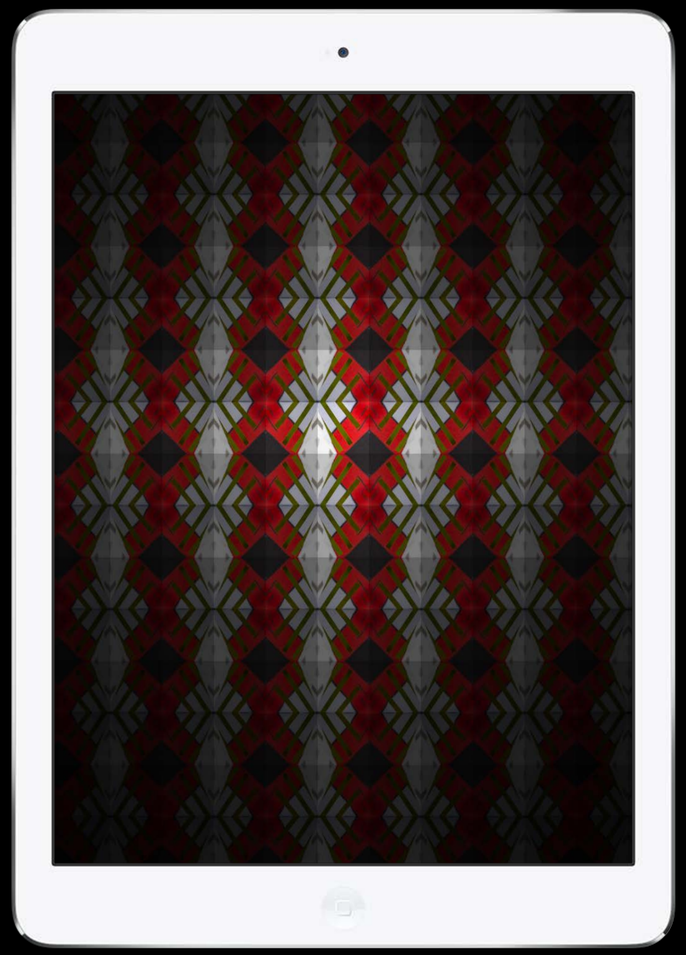
?

=

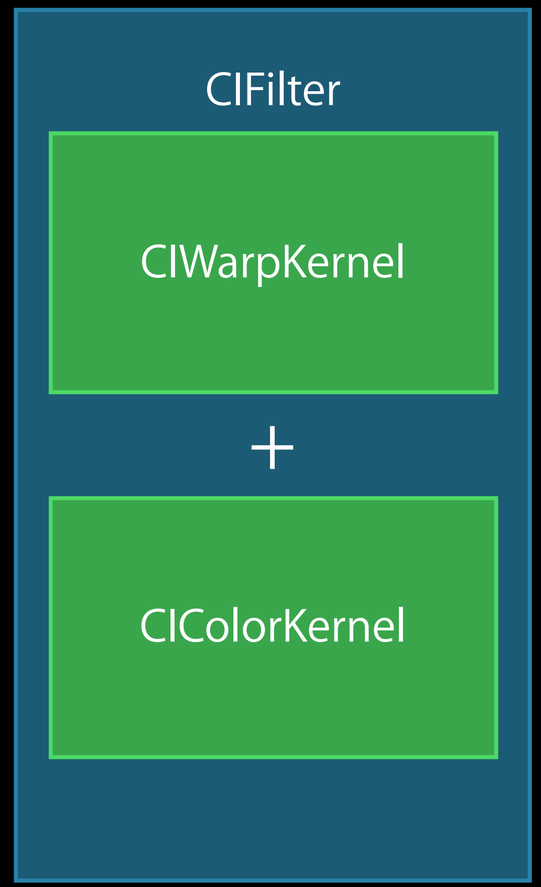


Square Kaleidoscope

Better solution



?



Square Kaleidoscope

CIWarpKernel

```
kernel vec2 squareKaleidoscopeWarp (vec2 center, float size) {
    // offset by center and transform to unit square
    vec2 p = (destCoord() - center) / size;

    // calculate number of mirror reflections to get to this point
    float z = abs(floor(p.x)) + abs(floor(p.y));

    // reflect this point into unit square space
    p = 1.0 - abs(mod(p, 2.0) - 1.0);

    // transform back to destination space
    return (p * size) + center;
}
```

Square Kaleidoscope

CIColorKernel

```
kernel vec4 squareKaleidoscopeColor (__sample s, vec2 center, float size) {
    // offset by center and transform to unit square
    vec2 p = (destCoord() - center) / size;

    // calculate number of mirror reflections to get to this point
    float z = abs(floor(p.x)) + abs(floor(p.y));

    // exponential decay
    float k = pow(0.8, z);

    // multiply color sample with decay
    return vec4(s.rgb * k, s.a);
}
```

Square Kaleidoscope

CIWarpKernel and CIColorKernel

```
CIImage *warpedImage = [warpKernel applyWithExtent:CGRectInfinite
    roiCallback:^(int index, CGRect rect) { return coreRect; }
     inputImage:inputImage
      arguments: @[center, size] ];
```

Square Kaleidoscope

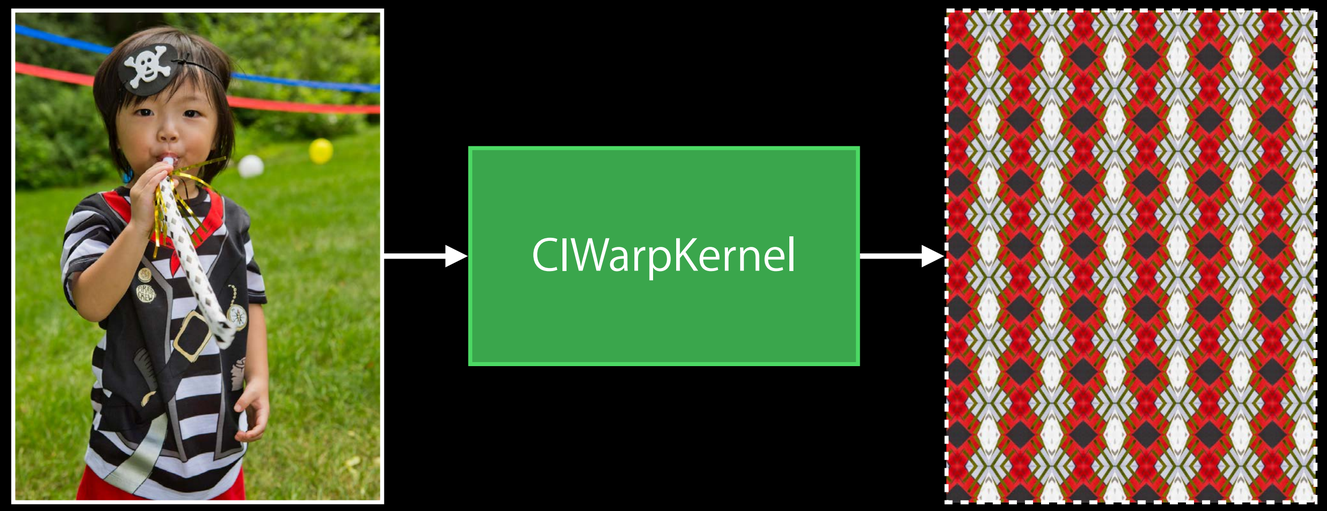
CIWarpKernel and CIColorKernel

CIWarpKernel

```
CIImage *warpedImage = [warpKernel applyWithExtent:CGRectInfinite
    roiCallback:^(int index, CGRect rect) { return coreRect; }
     inputImage:inputImage
      arguments: @[center, size] ];
```

Square Kaleidoscope

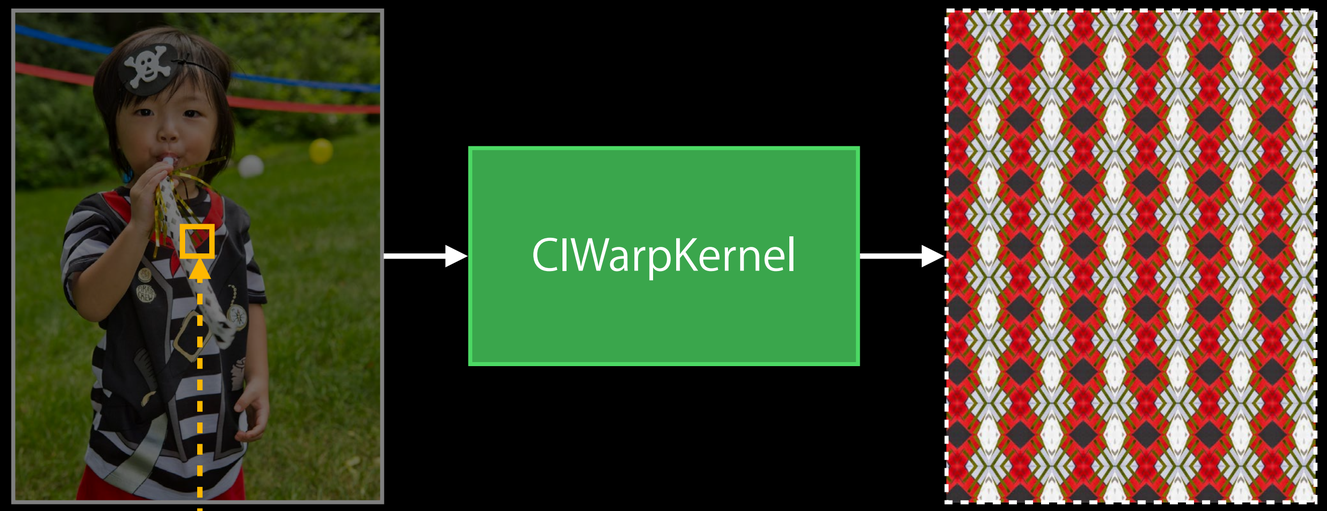
CIWarpKernel and CIColorKernel




```
CIImage *warpedImage = [warpKernel applyWithExtent:CGRectInfinite
    roiCallback:^(int index, CGRect rect) { return coreRect; }
     inputImage:inputImage
      arguments: @[center, size] ];
```

Square Kaleidoscope

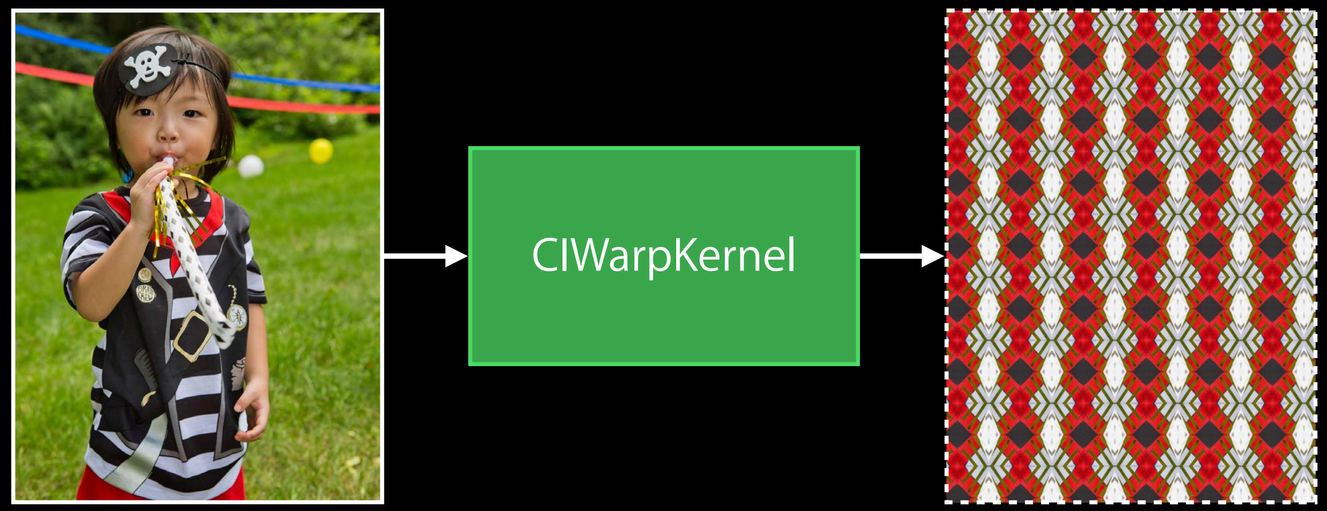
CIWarpKernel and CIColorKernel




```
CIImage *warpedImage = [warpKernel applyWithExtent:CGRectInfinite
    roiCallback:^(int index, CGRect rect) { return coreRect; }
     inputImage:inputImage
      arguments: @[center, size] ];
```

Square Kaleidoscope

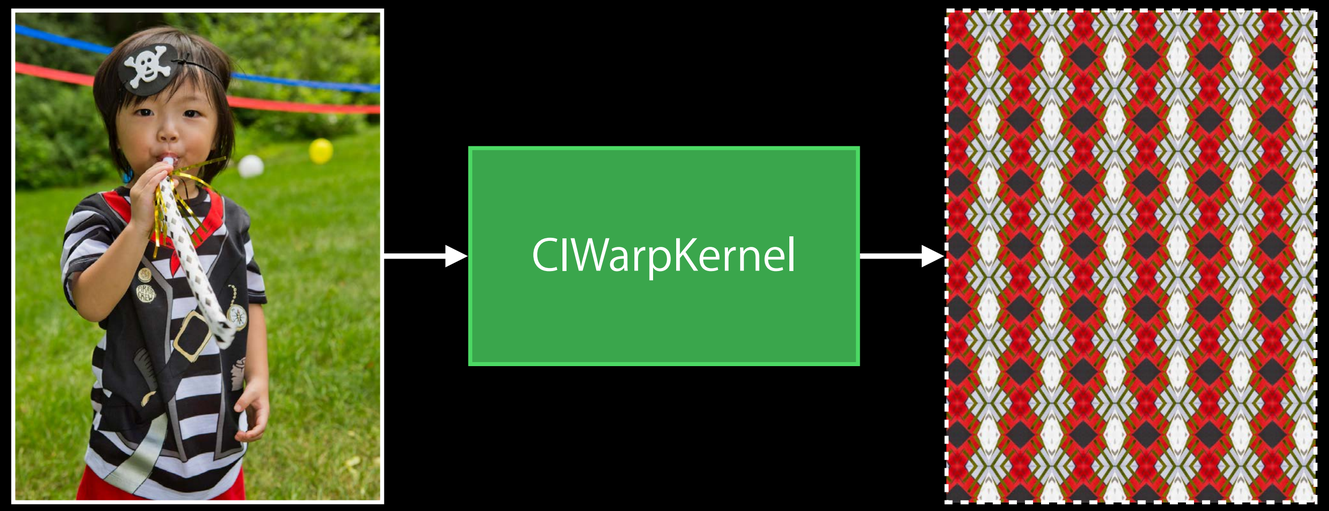
CIWarpKernel and CIColorKernel




```
CIImage *warpedImage = [warpKernel applyWithExtent:CGRectInfinite
    roiCallback:^(int index, CGRect rect) { return coreRect; }
     inputImage:inputImage
      arguments: @[center, size] ];
```

Square Kaleidoscope

CIWarpKernel and CIColorKernel

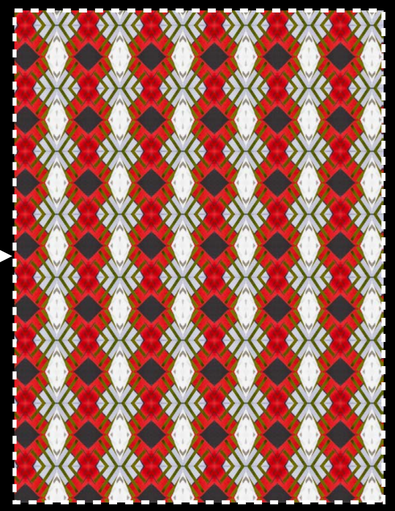



```
CIImage *warpedImage = [warpKernel applyWithExtent:CGRectInfinite
    roiCallback:^(int index, CGRect rect) { return coreRect; }
     inputImage:inputImage
      arguments: @[center, size] ];

CIImage *outputImage = [colorKernel applyWithExtent:warpedImage.extent
      arguments: @[warpedImage, center, size] ];
```

Square Kaleidoscope

CIWarpKernel and CIColorKernel

CIWarpKernel

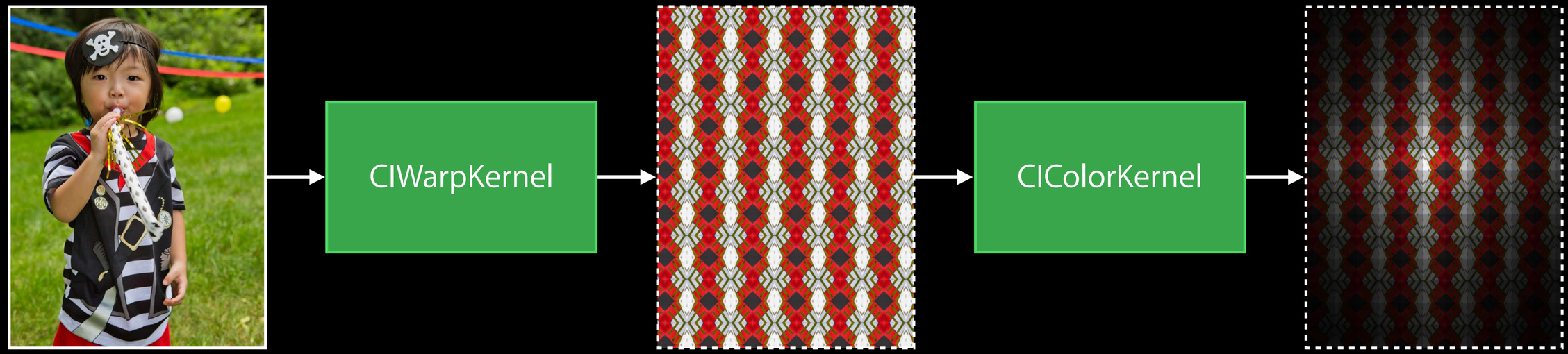


CIColorKernel

```
CIImage *warpedImage = [warpKernel applyWithExtent:CGRectInfinite
    roiCallback:^(int index, CGRect rect) { return coreRect; }
     inputImage:inputImage
      arguments: @[center, size] ];

CIImage *outputImage = [colorKernel applyWithExtent:warpedImage.extent
      arguments: @[warpedImage, center, size] ];
```

Square Kaleidoscope

CIWarpKernel and CIColorKernel




```
CIImage *warpedImage = [warpKernel applyWithExtent:CGRectInfinite
    roiCallback:^(int index, CGRect rect) { return coreRect; }
     inputImage:inputImage
      arguments: @[center, size] ];

CIImage *outputImage = [colorKernel applyWithExtent:warpedImage.extent
      arguments: @[warpedImage, center, size] ];
```

Recommendations

Recommendations

Write a general kernel only when needed

Recommendations

Write a general kernel only when needed

Write a general kernel initially for rapid prototyping, then try replacing with some combination of warp and color kernels

Platform Differences

Platform Differences

	iOS	OS X
Renderers	GPU	CPU and GPU
Control flow in language	Yes	No
Kernel classes	CIKernel CIColorKernel CIWarpKernel	CIKernel
CISampler	No	Yes
DOD	CGRect	CIFilterShape
ROI	Block Pointer	Selector
[CIFilter setDefaults]	Automatic	Explicit
[CIFilter customAttributes]	Class Method	Instance Method

What You've Learned Today

What You've Learned Today

How to write Color, Warp, and general kernels

What You've Learned Today

How to write Color, Warp, and general kernels

DOD and ROI

What You've Learned Today

How to write Color, Warp, and general kernels

DOD and ROI

- ROI function is critical for large image support to work correctly and optimally

What You've Learned Today

How to write Color, Warp, and general kernels

DOD and ROI

- ROI function is critical for large image support to work correctly and optimally

Platform differences

More Information

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

Developer Technical Support
http://developer.apple.com/contact

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

Related Sessions

• Introducing the Photos Frameworks	Nob Hill	Thursday 10:15AM
• Advances in Core Image	Pacific Heights	Thursday 2:00PM

Labs

• Core Image Lab	Media Lab B	Thursday 4:30PM

WWDC14