Class

CIImageProcessorKernel

The abstract class you extend to create custom image processors that can integrate with Core Image workflows.

iOS 10.0+ | iPadOS 10.0+ | Mac Catalyst 13.1+ | macOS 10.12+ | tvOS 10.0+ | visionOS 1.0+

```
class CIImageProcessorKernel
```

Overview

Unlike the `CIKernel` class and its other subclasses that allow you to create new image-processing effects with the Core Image Kernel Language, the `CIImageProcessorKernel` class provides direct access to the underlying bitmap image data for a step in the Core Image processing pipeline. As such, you can create subclasses of this class to integrate other image-processing technologies—such as Metal compute shaders, Metal Performance Shaders, Accelerate vImage operations, or your own CPU-based image-processing routines—with a Core Image filter chain.

Your custom image processing operation is invoked by your subclassed image processor kernel's `process(with:arguments:output:)` method. The method can accept zero, one or more inputs: kernels that generate imagery (such as a noise or pattern generator) need no inputs, while kernels that composite source images together require multiple inputs. The `arguments` dictionary allows the caller to pass in additional parameter values (such as the radius of a blur) and the `output` contains the destination for your image processing code to write to.

The following code shows how you can subclass `CIImageProcessorKernel` to apply the Metal Performance Shader `MPSImageThresholdBinary` kernel to a `CIImage`:

```
class ThresholdImageProcessorKernel: CIImageProcessorKernel {
static let device = MTLCreateSystemDefaultDevice()
```

```
override class func process(with inputs: [CIImageProcessorInput]?, arguments: [Strir
    guard
        let device = device,
        let commandBuffer = output.metalCommandBuffer,
        let input = inputs?.first,
        let sourceTexture = input.metalTexture,
        let destinationTexture = output.metalTexture,
        let thresholdValue = arguments?["thresholdValue"] as? Float else  {
            return
        }

    let threshold = MPSImageThresholdBinary(
        device: device,
        thresholdValue: thresholdValue,
        maximumValue: 1.0,
        linearGrayColorTransform: nil)

    threshold.encode(
        commandBuffer: commandBuffer,
        sourceTexture: sourceTexture,
        destinationTexture: destinationTexture)
    }
}
```

To apply to kernel to an image, the calling side invokes the image processor's `apply(withExtent:inputs:arguments:)` method. The following code generates a new `CIImage` object named `result` which contains a thresholded version of the source image, `inputImage`.

```
let result = try? ThresholdImageProcessorKernel.apply(
    withExtent: inputImage.extent,
    inputs: [inputImage],
    arguments: ["thresholdValue": 0.25])
```

Important

Core Image will concatenate filters in a network into as fewer kernels as possible, avoiding the creation of intermediate buffers. However, it is unable to do this with image processor kernels. To get the best performance, you should only use `CIImageProcessorKernel` objects when your image processing algorithms can't be expressed as Core Image Kernel Language.

Subclassing Notes

The `CIImageProcessorKernel` class is abstract; to create a custom image processor, you define a subclass of this class.

You do not directly create instances of a custom `CIImageProcessorKernel` subclass. Image processors must not carry or use state specific to any single invocation of the processor, so all methods (and accessors for readonly properties) of an image processor kernel class are class methods.

Your subclass should override at least the `process(with:arguments:output:)` method to perform its image processing.

If your image processor needs to work with a larger or smaller region of interest in the input image than each corresponding region of the output image (for example, a blur filter, which samples several input pixels for each output pixel), you should also override the `roi(forInput:arguments:outputRect:)` method.

You can also override the `formatForInput(at:)` method and `outputFormat` property getter to customize the input and output pixel formats for your processor (for example, as part of a multi-step workflow where you extract a single channel from an RGBA image, apply an effect to that channel only, then recombine the channels).

Using a Custom Image Processor

To apply your custom image processor class to filter one or more images, call the `apply(withExtent:inputs:arguments:)` class method. (Do not override this method.)

Topics

Type Properties

`class var outputFormat: CIFormat`

Override this class property if you want your processor's output to be in a specific pixel format.

`class var outputIsOpaque: Bool`

Override this class property if your processor's output stores 1.0 into the alpha channel of all pixels within the output extent.

`class var synchronizeInputs: Bool`

Override this class property to return false if you want your processor to be given input objects that have not been synchronized for CPU access.

Type Methods

```
class func apply(withExtent: CGRect, inputs: [CIImage]?, arguments: [String : Any]?) throws -> CIImage
```

Call this method on your Core Image Processor Kernel subclass to create a new image of the specified extent.

```
class func formatForInput(at: Int32) -> CIFormat
```

Override this class method if you want your any of the inputs to be in a specific pixel format.

```
class func process(with: [any CIImageProcessorInput]?, arguments: [String : Any]?, output: any CIImageProcessorOutput) throws
```

Override this class method to implement your Core Image Processor Kernel subclass.

```
class func roi(forInput: Int32, arguments: [String : Any]?, outputRect: CGRect) -> CGRect
```

Override this class method to implement your processor's ROI callback.

```
class func roiTileArray(forInput: Int32, arguments: [String : Any]?, outputRect: CGRect) -> [CIVector]
```

Override this class method to implement your processor's tiled ROI callback.

```
class func apply(withExtents: [CIVector], inputs: [CIImage]?, arguments: [String : Any]?) throws -> [CIImage]
```

Call this method on your multiple-output Core Image Processor Kernel subclass to create an array of new image objects given the specified array of extents.

```
class func outputFormat(at: Int32, arguments: [String : Any]?) -> CIFormat
```

Override this class method if your processor has more than one output and you want your processor's output to be in a specific supported `CIPixelFormat`.

```
class func process(with: [any CIImageProcessorInput]?, arguments: [String : Any]?, outputs: [any CIImageProcessorOutput]) throws
```

Override this class method of your Core Image Processor Kernel subclass if it needs to produce multiple outputs.

Relationships

Inherits From

NSObject

Conforms To

CVarArg
CustomDebugStringConvertible
CustomStringConvertible
Equatable
Hashable
NSObjectProtocol

See Also

Custom Image Processors

`protocol` **CIImageProcessorInput**

A container of image data and information for use in a custom image processor.

`protocol` **CIImageProcessorOutput**

A container for writing image data and information produced by a custom image processor.