Metal Shading Language for Core Image Kernels

Developer

Contents

Overview

The Metal Shading Language is a C++11-style programming language normally used for writing Metal performance shaders to run on the GPU. This guide shows how you can use the language to write Core Image kernels to add custom image processing routines to your Core Image pipeline. This document defines the Metal Shading Language features supported for `CIKernel`.

`CIKernel` Function Requirements

Denote a `CIKernel` function by enclosing it with an `extern` C qualifier. The name of the function can then be used to initialize a `CIKernel` with the `[CIKernel kernelWithName:fromMetalLibraryData:]` API.

Data Types

For a complete list of supported data types, see the Metal Shading Language Specification. The following additional data types are supported for `CIKernel` objects and declared in `coreimage namespace`.

Destination Types

Type	Description
`destination`	A kernel parameter type that allows access to the position of the pixel currently being computed. This parameter, which is required for `CIWarpKernel` and optional for `CIColorKernel` and `CIKernel`, must be the last parameter to a kernel function.
`group::destination`[1]	Same as a destination type, but allows write access to the 2 x 2 group of `float4` pixels currently being computed.
`group::destination_h`[1]	Same as a destination type, but allows write access to the 2 x 2 group of `half4` pixels currently being computed.

Sampler Types

Type	Description
`sample_t`	A sample value from a `CIImage` represented by a 4D 32-bit floating-point vector. Use as a parameter type only for representing a sample from an image. Otherwise behaves as a `float4`.
`sample_h`[1]	A sample value from a `CIImage` represented by a 4D 16-bit floating-point vector. Use as a parameter type only for representing a sample from an image. Otherwise behaves as a `half4`.
`sampler`	A sampler for a `CIImage` that returns 4D 32-bit floating-point precision samples.
`sampler_h`[1]	A sampler for a `CIImage` that returns 4D 16-bit floating-point precision samples.

[1] Available in iOS 12 and later and macOS 10.14 and later.

Functions

In addition to all intrinsic functions available in the Metal standard library, the following built-in functions are also available in `coreimage namespace`.

Relational Functions

Function	Returns
`vec<T,N> compare(vec<T,N> c, vec<T,N> a, vec<T,N> b)`	Elementwise `(c < 0) ? a : b`

Trigonometry Functions

Function	Returns
`float2 sincos(float)`	A vector containing the sine and cosine of an angle
`float2 cossin(float)`	A vector containing the cosine and sine of an angle

Color Functions

Function	Returns
`float4 premultiply(float4)` `half4 premultiply(half4)`	Multiplies red, green, and blue components of the parameter by its alpha component.
`float4 unpremultiply(float4)` `half4 unpremultiply(half4)`	If the alpha component of the parameter is greater than 0, divides the red, green, and blue components by alpha. If alpha is 0, this function returns the parameter.
`float3 srgb_to_linear(float3)` `half3 srgb_to_linear(half3)`	`(abs(s) < 0.04045) ? (s / 12.92) : sign(s) * pow(abs(s)*0.947867298578199 + 0.052132701421801, 2.4)`
`float3 linear_to_srgb(floa3)` `half3 linear_to_srgb(half3)`	`(abs(s) < 0.0031308) ? (s * 12.92) : sign(s) * pow(abs(s), 1.0/2.4) * 1.055 - 0.055)`
`float4 srgb_to_linear(float4)` `half4 srgb_to_linear(half4)`	`unpremultiply(s);srgb_to_linear(s.rgb);premultiply(s);`
`float4 linear_to_srgb(float4)` `half4 linear_to_srgb(half4)`	`unpremultiply(s);linear_to_srgb(s.rgb);premultiply(s);`

Destination Functions

coord

```
float2 coord()
```

Returns the position, in working space coordinates, of the pixel currently being computed. The destination space refers to the coordinate space of the image you're rendering.

write

```
void write(float4 v0, float4 v1, float4 v2, float4 v3)
void write(half4 v0, half4 v1, half4 v2, half4 v3)
```

Writes four-color values to the destination image for the current 2 x 2 group of pixels.

Sampling Functions

sample

```
float4 sample(float2 coord)
half4 sample(float2 coord)
```

Returns the pixel value produced from the sampler at the position `coord`, where `coord` is specified in the sampler's coordinate system.

transform

```
float2 transform(float2 coord)
```

Returns the position in the coordinate space of the sampler that's associated with the position defined in working space coordinates `coord`. Working space coordinates reflect any transformations that you've applied to the working space.

For example, if you're producing a pixel in the working space, and you need to retrieve the pixels that surround this pixel in the original image, you'd make calls similar to the following, where d is the location of the pixel you're producing in the working space, and `image` is the image source for the pixels.

```
src.transform(d + float2(-1.0,-1.0));
src.transform(d + float2(+1.0,-1.0));
src.transform(d + float2(-1.0,+1.0));
src.transform(d + float2(+1.0,+1.0));
```

coord

```
float2 coord()
```

Returns the position, in sampler space, of the sampler that's associated with the current output pixel after applying any transformation matrix associated with the sampler. The sample space refers to the coordinate space you're texturing from. If your source data is tiled, the sample coordinate will have an offset `(dx/dy)`. You can convert a destination location to the sampler location using the sampler's `transform` function, which is equivalent to `src.transform(dest.coord())`.

extent

```
float4 extent()
```

Returns the extent `(x, y, width, height)` of the sampler in world coordinates as a four-element vector. If the extent is infinite, the vector `(-INF,-INF,INF,INF)` is returned.

origin

```
float2 origin()
```

Returns the origin of the sampler extent; equivalent to `src.extent().xy`.

size

```
float2 size()
```

Returns the size of the sampler extent; equivalent to `src.extent().zw`.

gatherX

```
float4 gatherX(float2 coord)
half4 gatherX(float2 coord)
```

Returns four samples of the X-component to be used for bilinear interpolation when sampling at the position `coord`, where `coord` is specified in the sampler's coordinate system. The samples are positioned counterclockwise, starting with the sample to the lower left.

gatherY

```
float4 gatherY(float2 coord)
half4 gatherY(float2 coord)
```

Returns four samples of the Y-component to be used for bilinear interpolation when sampling at the position `coord`, where `coord` is specified in the sampler's coordinate system. The samples are positioned counterclockwise, starting with the sample to the lower left.

gatherZ

```
float4 gatherZ(float2 coord)
half4 gatherZ(float2 coord)
```

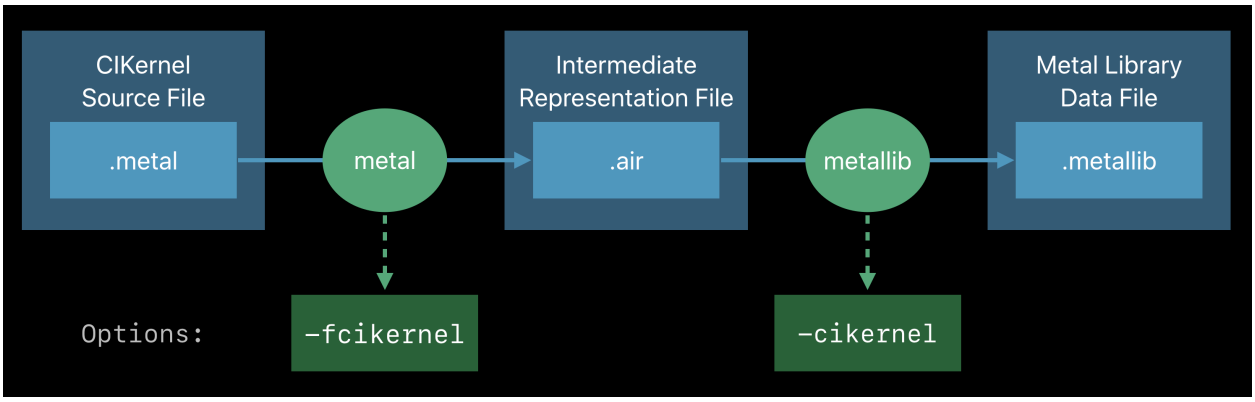
Returns four samples of the Z-component to be used for bilinear interpolation when sampling at the position `coord`, where `coord` is specified in the sampler's coordinate system. The samples are positioned counterclockwise, starting with the sample to the lower left.

gatherW

```
float4 gatherW(float2 coord)
half4 gatherW(float2 coord)
```

Returns four samples of the W-component to be used for bilinear interpolation when sampling at the position `coord`, where `coord` is specified in the sampler's coordinate system. The samples are positioned counterclockwise, starting with the sample to the lower left.

Compiling and Linking



To compile a Metal shader with `CIKernel` objects, specify the `-fcikernel` option.

```
xcrun metal -fcikernel MyKernels.metal -o MyKernels.air
```

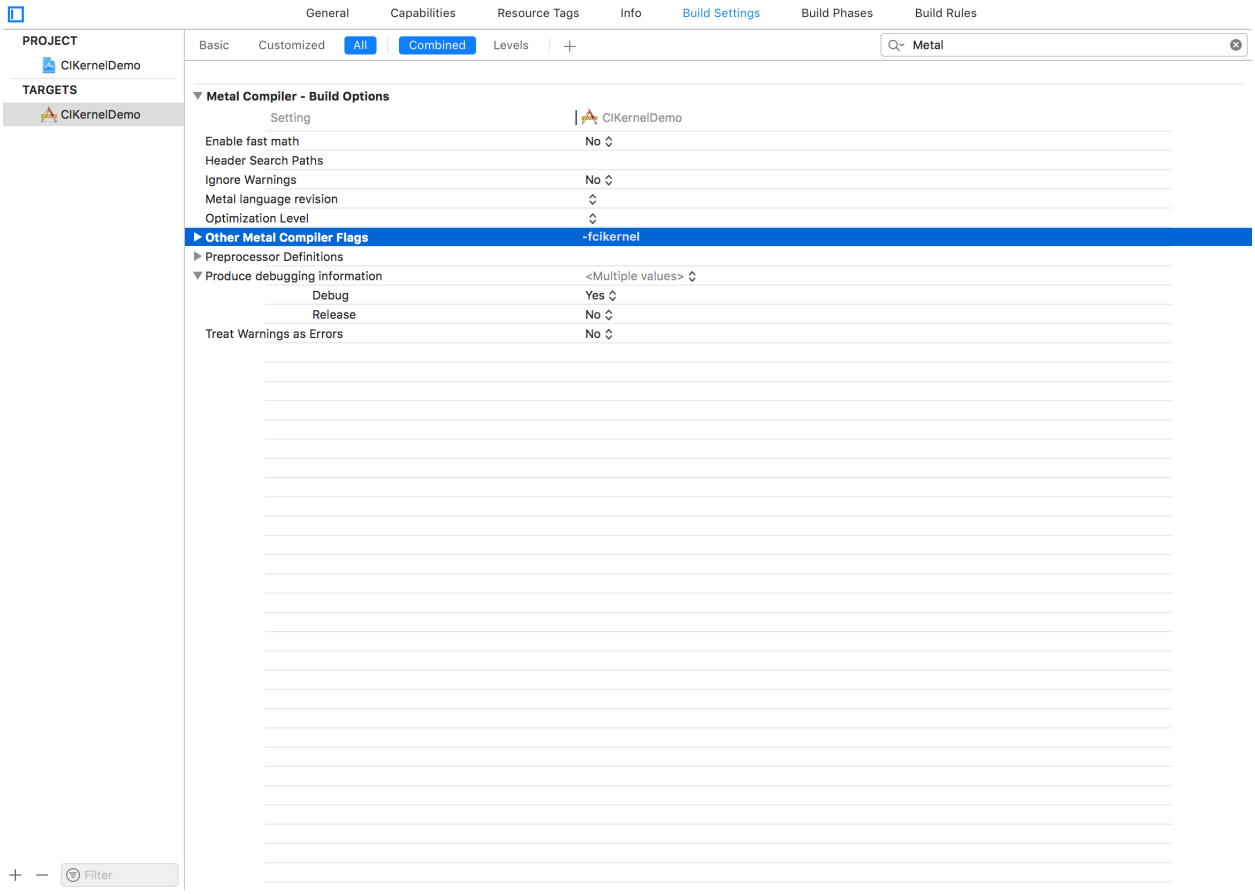
To link a Metal shader with `CIKernel` code, specify the `-cikernel` option.

```
xcrun metallib -cikernel MyKernels.air -o MyKernels.metallib
```

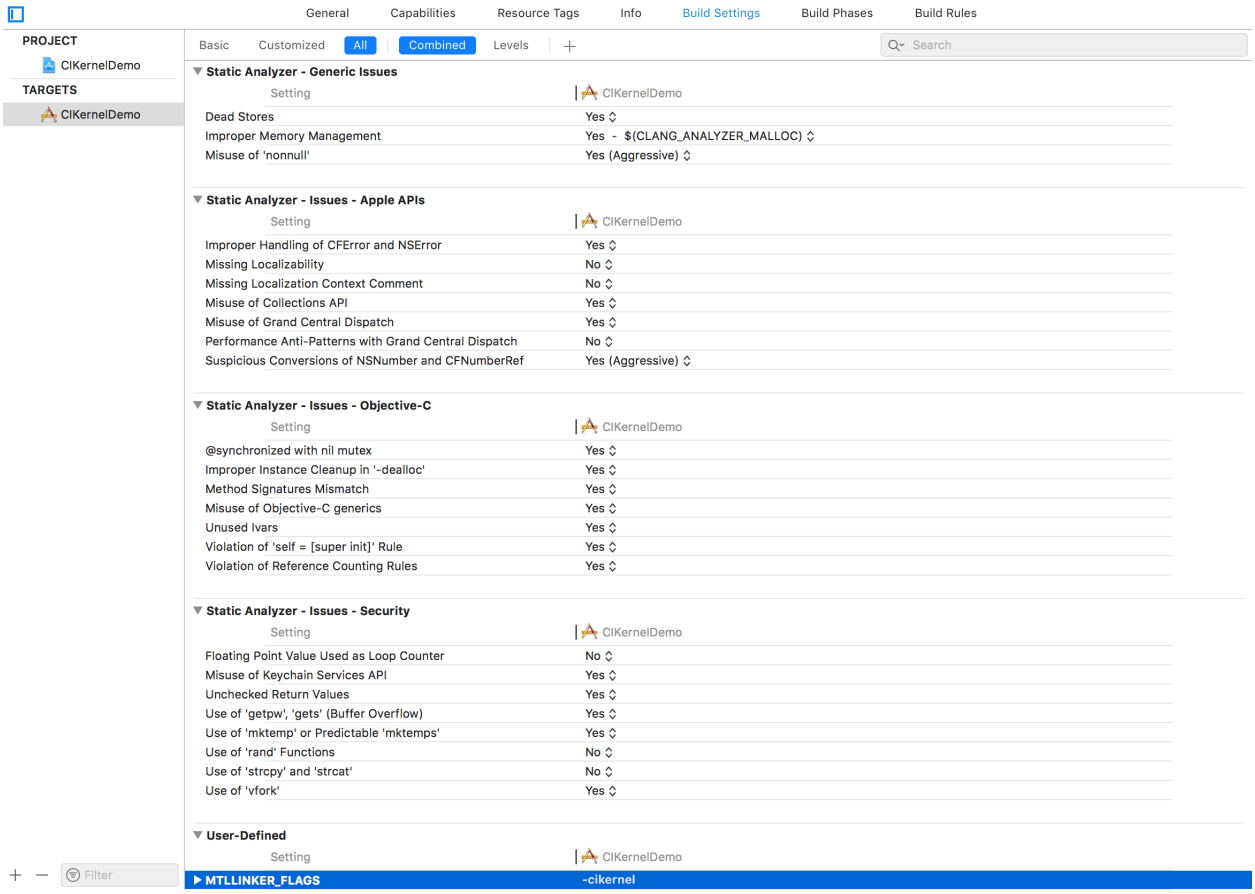
You can either integrate these steps into your project build configuration manually or specify them in your project's build settings within Xcode.

Xcode Integration

To specify the compiler option, add `-fcikernel` to `Other Metal Compiler Flags` within the Metal Compiler - Build Options group in Build Settings.



To specify the linker option, add a new user-defined setting named `MTLLINKER_FLAGS` in Build Settings and specify `-cikernel` for it.



Copyright and Notices

Apple Inc.
One Apple Park Way
Cupertino, CA 95014
USA
408-996-1010