Core Image Kernel Language Reference

Graphics & Animation: 2D Drawing

2008-06-09

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

Contents

Introduction

The Core Image kernel language defines functions, data types, and keywords that you can use to specify image processing operations for custom Core Image filters that you write. You can also use a subset of the OpenGL Shading Language (glslang).

This document defines the symbols in the Core Image kernel language and lists the symbols in the OpenGL Shading Language that are unsupported in Core Image filters.

Any developer who wants to use the Core Image API to write custom image processing filters should read this document. Before reading this document you should be familiar with the documents listed in the See Also section.

Organization of This Document

"Core Image Kernel Language" (page 7) defines the functions, data types, and keywords available for writing image processing routines and lists symbols in the OpenGL Shading Language that are not supported.

See Also

- *Core Image Reference Collection* defines the classes used to define and access image processing filters.
- *Core Image Programming Guide* describes how to write custom image processing filters and package them as image units. It also provides several examples of kernel routines.
- OpenGL Shading Language, available from the OpenGL website, provides a reference to glslang.

Introduction

See Also

Core Image Kernel Language

The following sections list the symbols provided by the Core Image Kernel Language:

- "Data Types" (page 10)
- "Keywords" (page 11)
- "Functions" (page 7)

You can use these symbols together with any of the OpenGL Shading Language routines that Core Image supports. See "Unsupported Items" (page 11) for those you can't use.

Functions

This section describes the following functions:

- "compare" (page 8)
- "cos_" (page 8)
- "cossin" (page 8)
- "cossin_" (page 8)
- "destCoord" (page 8)
- "premultiply" (page 8)
- "sample" (page 8)
- "samplerCoord" (page 9)
- "samplerExtent" (page 9)
- "samplerOrigin" (page 9)
- "samplerSize" (page 9)
- "samplerTransform" (page 9)
- "sin_" (page 9)
- "sincos" (page 9)
- "sincos_" (page 10)
- "tan_" (page 10)
- "unpremultiply" (page 10)

compare

`genType compare (genType x, genType y, genType z)`

For each component, returns `x < 0 ? y : z`. Note that `genType` is a placeholder for an arbitrary vector type.

cos_

`genType cos_ (genType x)`

Similar to `cos (x)` except that `x` must be in the [`-pi`, `pi`] range. Note that `genType` is a placeholder for an arbitrary vector type.

cossin

`vec2 cossin (float x)`

Returns `vec2 (cos (x), sin (x))`.

cossin_

`vec2 cossin_ (float x)`

Returns `vec2 (cos (x), sin (x))`. This function expects `x` to be in the [`-pi`, `pi`] range.

destCoord

`varying vec2 destCoord ()`

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 are rendering.

premultiply

`vec4 premultiply (vec4 color)`

Multiplies the red, green, and blue components of the `color` parameter by its alpha component.

sample

`vec4 sample (uniform sampler src, vec2 point)`

Returns the pixel value produced from sampler `src` at the position `point`, where `point` is specified in sampler space.

samplerCoord

```
varying vec2 samplerCoord (uniform sampler src)
```

Returns the position, in sampler space, of the sampler `src` that is associated with the current output pixel (that is, after any transformation matrix associated with `src` is applied). The sample space refers to the coordinate space of that you are texturing from.

Note that 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 `samplerTransform` function.

samplerExtent

```
uniform vec4 samplerExtent (uniform sampler src)
```

Returns the extent of the sampler in world coordinates, as a four-element vector [x, y, width, height].

samplerOrigin

```
uniform vec2 samplerOrigin (uniform sampler src)
```

Equivalent to `samplerExtent (src).xy`.

samplerSize

```
uniform vec2 samplerSize (uniform sampler src)
```

Equivalent to `samplerExtent (src).zw`.

samplerTransform

```
vec2 samplerTransform (uniform sampler src, vec2 point)
```

Returns the position in the coordinate space of `src` that is associated with the position `point` defined in working space coordinates (that is, after sampler transformations are applied).

sin_

```
genType sin_ (genType x)
```

Similar to `sin (x)` except that `x` must be in the [`-pi`, `pi`] range. Note that `genType` is a placeholder for an arbitrary vector type.

sincos

```
vec2 sincos (float x)
```

Returns `vec2 (sin (x), cos (x))`.

sincos_

`vec2 sincos_ (float x)`

Returns `vec2 (sin (x), cos (x))`. This function expects `x` to be in the [`-pi`, `pi`] range.

tan_

`genType tan_ (genType x)`

Similar to `tan (x)` except that `x` must be in the [`-pi`, `pi`] range. Note that `genType` is a placeholder for an arbitrary vector type.

unpremultiply

`vec4 unpremultiply (vec4 color)`

If the alpha component of the `color` parameter is greater than 0, divides the red, green and blue components by alpha. If alpha is 0, this function returns `color`.

Data Types

`sampler`

Specifies a sampler passed in from CISampler that is used to get samples from data.

`__color`

Specifies a type for kernel parameters that need to be color matched to the current CIContext working color space.

`__table`

Specifies a flag for a sampler that fetches values from a lookup table.

The `__table` flag must precede the `sampler` type. The flag ensures that Core Image does not sample the table values using world coordinates.

For example, to use a lookup table sampler in a kernel named `shadedmaterial`, the kernel declaration would be:

```
kernel vec4 shadedmaterial(sampler heightfield, __table sampler envmap, float surfaceScale, vec2 envscaling)
```

Using the `__table` flag prevents the `envmap` sampler values from being transformed, even if the shaded material kernel gets inserted into a filter chain with an affine transform. If you don't tag the sampler this way and you chain the shaded material filter to an affine transform for rotation, then looking up values in the environment map results in getting rotated values, which is not correct because the lookup table is simply a data collection.

Keywords

`kernel`

Specifies a kernel routine. Kernel routines are extracted and compiled by the CIKernel class. A kernel encapsulates the computation required to compute a single pixel in the output image.

Each kernel is tagged by the `kernel` keyword in its return type. The underlying return type of the kernel must be `vec4` . Core Image requires this type in order to return the output pixel for the input pixel currently being evaluated.

All parameters to the kernel are implicitly marked `uniform`. Parameters marked `out` and `inout` are not allowed.

You can pass the following types to a kernel routine:

- `sampler`: Requires a CISampler object when applied.
- `__table`: A qualifier for a `sampler` type.
- `float`, `vec2`, `vec3`, `vec4`: Requires an NSNumber or CIVector.
- `__color`: A color that will be matched to the CIContext working color space when passed into the program. It requires a CIColor object when applied. To the kernel program it appears to be a `vec4` type in premultiplied RGBA format.

Unsupported Items

Core Image does not support the OpenGL Shading Language source code preprocessor. In addition, the following are not implemented:

- Data types: `mat2`, `mat3`, `mat4`, `struct`, `arrays`
- Statements: `continue`, `break`, `discard`. Other flow control statements (`if`, `for`, `while`, `do while`) are supported only when the loop condition can be inferred at the time the code compiles.
- Expression operators: `% << >> | & ^ || && ^^ ~`
- Built-in functions: `ftransform`, `matrixCompMult`, `dfdx`, `dfdy`, `fwidth`, `noise1`, `noise2`, `noise3`, `noise4`, `refract`

Document Revision History

This table describes the changes to *Core Image Kernel Language Reference*.

Date	Notes
2008-06-09	Updated for Mac OS X v10.5.
	Added information about the destination and sampler coordinate spaces.
2006-06-28	New document that describes the symbols for writing image-processing kernels.
	The content in this book was published previously as an appendix in *Core Image Programming Guide*.

Document Revision History