Core Image

Digital Arts for Cocoa Developer

W. Lonsing CocoaHeads Aachen 26/06/2008

CoreImage

Intro

Core Image is a simple API to put creative tools into the hands of developers. It provides both a graphical and a code-based interface to implement graphic and photo editing in software.

CoreImage

Relation to QuartzComposer

CoreImage

Relation to QuartzComposer

The QuartzComposer provides a graphical interface to access the CoreImage-API.
Results are Quartz-Composition.

CoreImage

Relation to Cocoa

CoreImage

Relation to Cocoa

- CIImage vs. NSImage
- A CIImage object is not an image but rather a recipe for it. A CIImage object has all the information necessary to produce an image, but Core Image doesn't actually render an image until it is requested.

CoreImage

Relation to Cocoa

- CIImage vs. NSImage

- An NSImage-object is an empty shell. Actual image data is stored in bitmaps.

CoreImage

Relation to Cocoa

- CIImage vs. NSImage

- Objective-C Interface

➡ Additional Routines required to transform Images from CoreImage to Cocoa and from Cocoa to CoreImage

```
// create a CIIamge from an NSImage, a triple step dance

// create the TIFF-data from the NSImage
NSData *tiffData = [image TIFFRepresentation];

// create the NSBitmapImageRep from the TIFF-data
NSBitmapImageRep *bm = [NSBitmapImageRep imageRepWithData:tiffData];

 // create the final CIIamge
 CIImage *ciImage =
          [[[CIImage alloc]initWithBitmapImageRep:bm]autorelease];
```

```
// create an NSImage from a CIIamge, just another dance
// first step: determine the size
CGRect rect = [result extent];
NSSize aSize = NSMakeSize(rect.size.width, rect.size.height);

// create the NSImage-object, just an empty shell
NSImage *image = [[[NSImage alloc] initWithSize:aSize] autorelease];
// create the NSCIImageRep from the CIImage
NSCIImageRep *ciRep = [NSCIImageRep imageRepWithCIImage: ciImage];

// add the representation
[image addRepresentation:ciRep];
```

CoreImage

Filter Usage

CoreImage

Filter Usage

Most filters accept input images as sources and all provide one single output image as result. Filters can be chained.

```
CIFilter * myFilter = [CIFilter filterWithName:@"CIColorMatrix"];

// set the input image
[myFilter setValue:ciImage forKey:@"inputImage"];

// get the output imgae
ciImage = [myFilter valueForKey:@"outputImage"];
```

Creation

Ready-made and Built-in Filters

Creation

Ready-made and Built-in Filters

- Filter with undefined values
- Filter with defined values
- Filter Generator

```objc
// create the filter, an instance of CIColorMatrix
CIFilter *matrixFilter = [CIFilter filterWithName:@"CIColorMatrix"]

// all values are set to the same vector
CIVector *aVector = [CIVector vectorWithX:0.0 Y:0.0 Z:0.0 W:1.0];

// let the vector slip in
[matrixFilter setValue:aVector forKey:@"inputRVector"];
[matrixFilter setValue:aVector forKey:@"inputGVector"];
[matrixFilter setValue:aVector forKey:@"inputBVector"];

[matrixFilter setValue:aVector forKey:@"inputAVector"];
[matrixFilter setValue:aVector forKey:@"inputBiasVector"];
```

```
// all values are set to the same vector
CIVector * aVector = [CIVector vectorWithX:0.0 Y:0.0 Z:0.0 W:1.0];

// create the filter, an instance of CIColorMatrix,
// with initial values
CIFilter *matrixFilter = [CIFilter filterWithName:@"CIColorMatrix"
                    keysAndValues:@"inputRVector", aVector,
                                  @"inputGVector", aVector,
                                  @"inputBVector", aVector,
                                  @"inputAVector", aVector,
                                  @"inputBiasVector", aVector,nil];
```

```
CIFilterGenerator *generator =[CIFilterGenerator filterGenerator];

// inject the previously created filter simply by connecting it
[generator connectObject:nil withKey: kCIInputImageKey
                toObject:matrixFilter withKey:kCIInputImageKey];

// export keys as necessary, input and output
[generator exportKey:kCIInputImageKey fromObject:matrixFilter
            withName:nil];
[generator exportKey:kCIOutputImageKey fromObject:matrixFilter
                withName:nil];

// generate the filter
 CIFilter * generatedFilter = [generator filter];
// and use it as usual
[generatedFilter setValue:ciImage forKey:kCIInputImageKey];
ciImage = [generatedFilter valueForKey:kCIOutputImageKey];
```

Examples

based on Ready-made Filters

Examples

based on Ready-made Filters

- Mask from Alpha
- Badged Image
- Reflected Image

```
// all values are set to the same vector
CIVector * aVector = [CIVector vectorWithX:0.0 Y:0.0 Z:0.0 W:1.0];

// create the filter, an instance of CIColorMatrix,
// with initial values
CIFilter *matrixFilter = [CIFilter filterWithName:@"CIColorMatrix"
                    keysAndValues:@"inputRVector", aVector,
                                  @"inputGVector", aVector,
                                  @"inputBVector", aVector,
                                  @"inputAVector", aVector,
                                  @"inputBiasVector", aVector,nil];
```

Badged Image

Filters in Use

Badged Image

Filters in Use

- Affine Transform
- Lanczos Scale Transform
- Gaussian Blur
- Composite Source Over

```
CIImage *result;
NSAffineTransform *transform = [NSAffineTransform transform];
// presume values are set here
CIFilter *filter = [CIFilter filterWithName:@"CIAffineTransform"];
      [filter setValue: badgeCIImage forKey:@"inputImage"];
      [filter setValue: transform forKey:@"inputTransform"];
      result = [filter valueForKey:@"outputImage"];

filter = [CIFilter filterWithName:@"CIGaussianBlur"];
      [filter setValue:bgnCIImage forKey:@"inputImage"];
      [filter setValue:[NSNumber numberWithFloat:1.1]
                                          forKey:@"inputRadius"];
      bgnCIImage = [filter valueForKey:@"outputImage"];

filter = [CIFilter filterWithName:@"CISourceOverCompositing"];
[filter setValue: bgnCIImage forKey:@"inputBackgroundImage"];
[filter setValue: badgeCIImage forKey:@"inputImage"];

result = [filter valueForKey:@"outputImage"];
```

Reflected Image

Filters in Use

Reflected Image

Filters in Use

- Linear Gradient (white-clear)
- Perspective Transform
- Crop (multiple)
- Affine Transform (multiple)
- Composite Source Over/at Top

CoreImage

Parts of an Image Unit

CoreImage

Parts of an Image Unit

- Kernel Routine File
- Objective-C Filter Files
- Plug-in Files

Kernel Routine File

Custom-made Filters

Kernel Routine File

Custom-made Filters

- Core Image Language
- Evaluation and Testing in QC
- Plugin: Code as *.cikernel -File
- Data Types mapped to Classes

```
  kernel vec4 weightedGrayscaleKernel(sampler image,
          float red, float green, float blue)
{
 // Get source pixel
 vec4   p = sample(image, samplerCoord(image));

 // Calculate the intensity
 float calc = clamp(red* p.r + green* p.g + blue* p.b, 0.0, 1.0);

 // Return the destination pixel based on intensity
 return vec4(calc, calc, calc, p.a);
}
```

Objective-C Files

Custom-made Filters

Objective-C Files

Custom-made Filters

- Xcode Template
- Subclass of CIFilter
- Methods to implement:
  - (id)init
  - (NSDictionary *)customAttributes
  - (CIImage *)outputImage

```
- (id)init
{
 if(_WeightedGrayScaleFilterKernel == nil)
 {
 NSBundle *bundle = [NSBundle
      bundleForClass:NSClassFromString(@"WeightedGrayScaleFilter")];

 NSString *code = [NSString stringWithContentsOfFile:
          [bundle pathForResource:@"WeightedGrayScaleFilterKernel"
                           ofType:@"cikernel"]];

 NSArray *kernels = [CIKernel kernelsWithString:code];

 _WeightedGrayScaleFilterKernel = [[kernels objectAtIndex:0] retain];
 }
    return [super init];
}
```

```
- (NSDictionary *)customAttributes
{
 return [NSDictionary dictionaryWithObjectsAndKeys:
...
    [NSDictionary dictionaryWithObjectsAndKeys:
          [NSNumber numberWithDouble:  0.00], kCIAttributeMin,
          [NSNumber numberWithDouble:  1.00], kCIAttributeMax,
          [NSNumber numberWithDouble:  0.00], kCIAttributeSliderMin,
          [NSNumber numberWithDouble:  1.00], kCIAttributeSliderMax,
          [NSNumber numberWithDouble:  0.33], kCIAttributeDefault,
          [NSNumber numberWithDouble:  1.00], kCIAttributeIdentity,
         @"NSNumber",                         kCIAttributeClass,
         kCIAttributeTypeScalar,              kCIAttributeType,
              nil],
     @"inputGreenWeight",
...
    nil];
}
```

```
- (CIImage *)outputImage
{
    CISampler *src = [CISampler samplerWithImage:inputImage];

    return [self apply:_WeightedGrayScaleFilterKernel,
            src,
            inputRedWeight,
            inputGreenWeight,
            inputBlueWeight,
            nil];
}
```

Finalize

Final Steps for Custom-made Filters

Finalize

Final Steps for Custom-made Filters

- **Validation**
  ImageUnitAnalyzer in Terminal

- **Installation**
  Copy Plugin into
  ”~/Graphics/Image Units”

Demos

Demos

- Mirror (in Quartz Composer)
- Color Tracking (Xcode)