Image Processing and Effects with Core Image

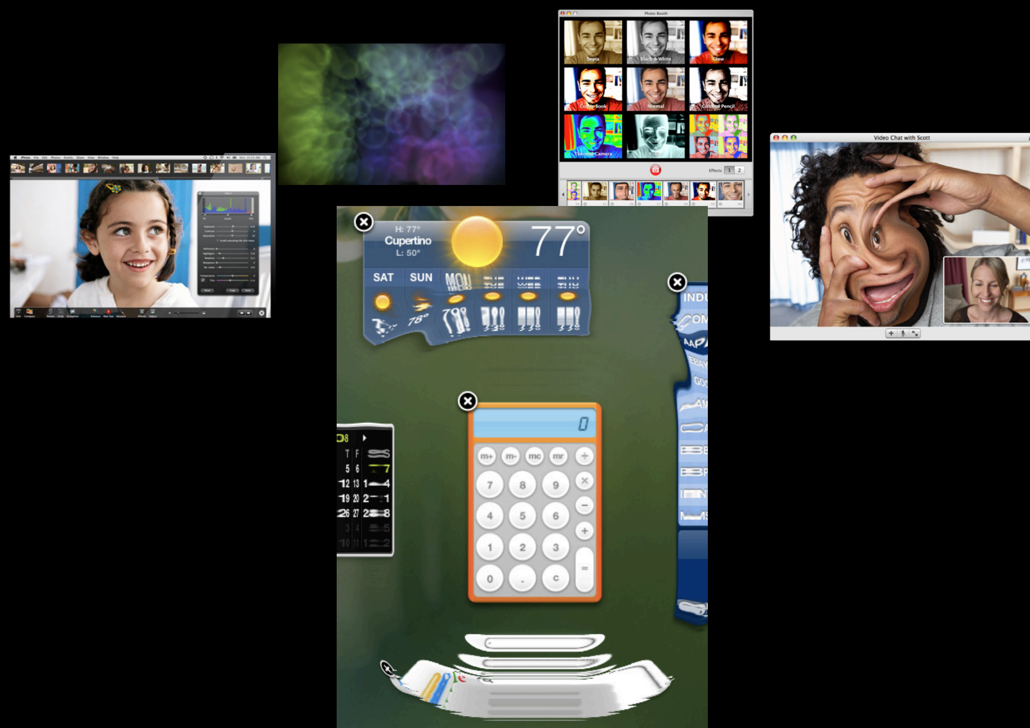
David Hayward

What We Will Discuss Today

- A quick introduction to Core Image
- Getting started with Core Image
- Using Core Image efficiently
- Writing CIFilters

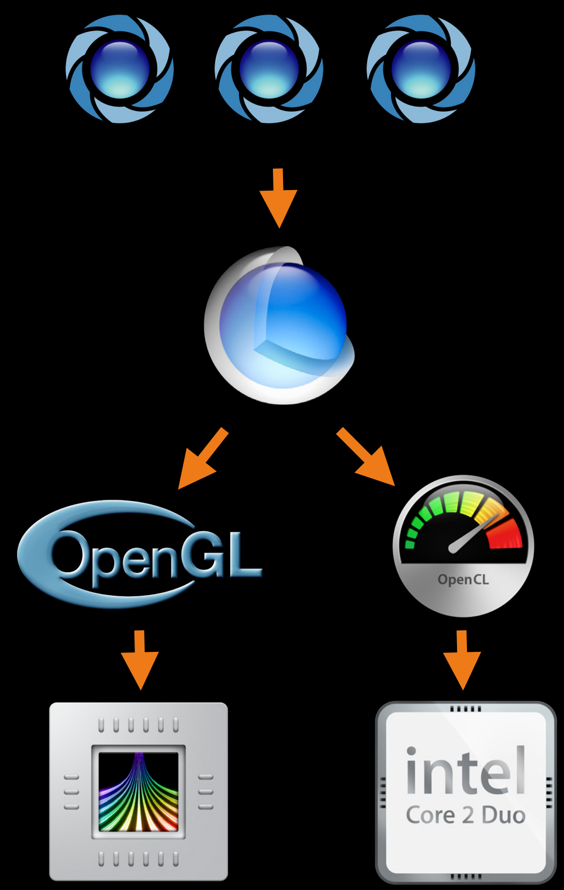
A Quick Introduction to Core Image

Core Image in Use



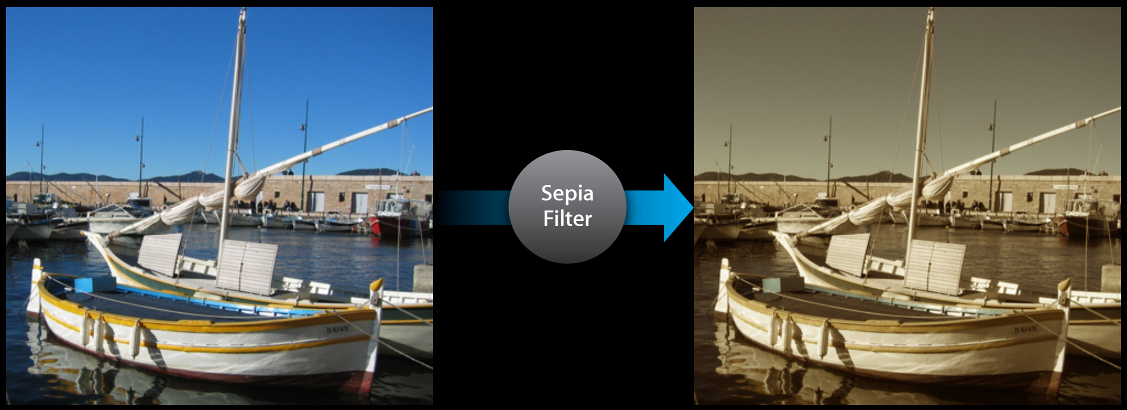
Meet Core Image

- Filter kernels are written architecture-independent
  - Language is C-like
  - Has vector types
  - Is a subset of OpenGL shading language
- Core Image can execute filters on GPU or multicore CPU as desired
- Leverages OpenGL and/or OpenCL



Basic Concept

Filters perform per pixel operations on an image



Original　　　Sepia Tone applied

The final result is a new image

Basic Concept

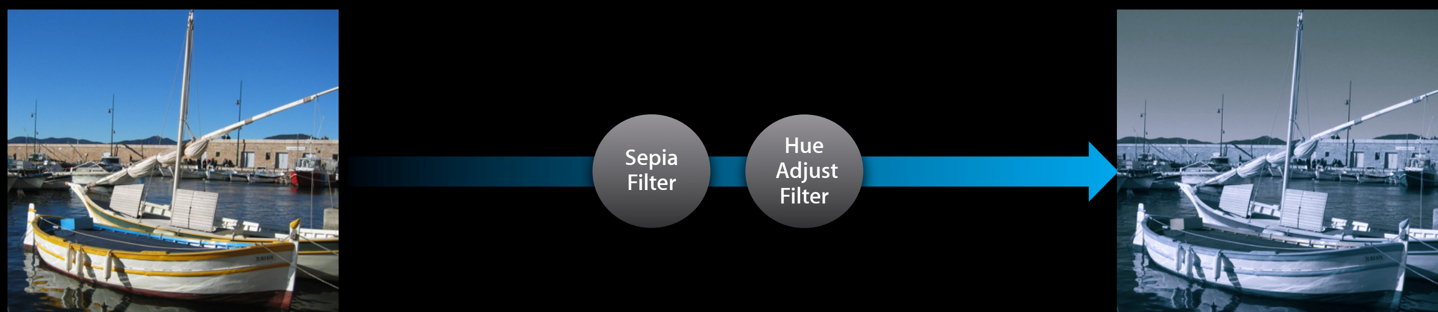
Filters can be chained together

Original — Sepia Tone applied — Sepia Tone + Hue Adjust

This allows for complex effects

Basic Concept

Filters can be concatenated



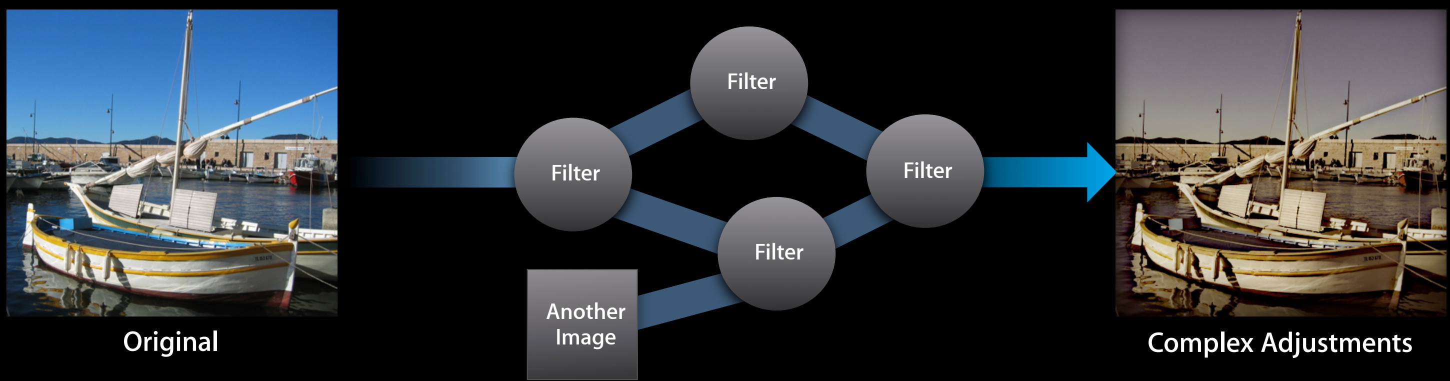
Original

Sepia Tone + Hue Adjust

This reduces intermediate buffers

Basic Concept

Complex filter trees are supported



Core Image will optimize

Built-in Filters

4 Geometry adjustments

12 Distortion effects

7 Blurs

2 Sharpens

6 Color adjustments

10 Color effects

15 Stylize

5 Halftone effects

15 Tile effects

7 Generators

9 Transitions

22 Composite operations

8 Reduction operations and more

Core Image Optimizes Your Rendering Tree

- Runtime contains a just-in-time optimizing compiler
  - Optimization is deferred until draw time
  - Evaluates just-what's-needed to display
  - Tiles large images
- Performs optimizations that general compilers can't
  - Concatenates sequential matrix operations
  - Concatenates sequential alpha-premul/alpha-unpremul
  - Reorders scale operations if possible
  - Only does color management when needed
- Optimization improves performance and precision

Getting Started with Core Image

The Cast:

- CIFilter:
  - A mutable object that represents an effect
  - Has image or numeric input parameters
  - Produces one output image based on current inputs
- CIImage:
  - An immutable object that represents the recipe for an image
  - Can represent a file from disk or the output of a CIFilter
- CIContext:
  - A destination where Core Image should draw results
  - Can be based on an OpenGL or Core Graphics Context

A Play In Four Acts:

1. Create a CIImage object

```
image = [CIImage imageWithContentsOfURL: myURL];
```

2. Create a CIFilter object

```
filter = [CIFilter filterWithName: @"CISepiaTone"];
[filter setValue: image forKey: kCIInputImageKey];
[filter setValue: [NSNumber numberWithFloat: 0.8f] forKey: kCIInputIntensityKey];
```

3. Create a CIContext object

```
context = [CIContext contextWithCGContext: cgcontext options: nil];
```

4. Draw the filter output image into the context

```
result = [filter valueForKey: kCIOutputImageKey];
[context drawImage: result atPoint: CGPointZero fromRect: [result extent]];
```

All Together Now:

Just a few lines of Core Image code...

```
CIImage* image = [CIImage imageWithContentsOfURL: myURL];

image = [[[CIFilter filterWithName:@"CISepiaTone" keysAndValues:
                kCIInputImageKey, image,
                kCIInputIntensityKey, [NSNumber numberWithFloat:0.8f], nil]
        valueForKey:kCIOutputImageKey];

image = [[[CIFilter filterWithName:@"CIHueAdjust" keysAndValues:
                kCIInputImageKey, image,
                kCIInputAngleKey, [NSNumber numberWithFloat:1.57], nil]
        valueForKey:kCIOutputImageKey];

[context drawImage:image atPoint:CGPointZero fromRect:[image extent]];
```

All Together Now:

... Core Image will do all this OpenGL work

```
CGLQueryRendererInfo(2, 0x00000000, 0);
CGLDescribeRenderer(0x00246210, 0, kCGLRPRendererID);
CGLChoosePixelFormat({kCGLPFAColorSize, 32, kCGLPFANoRecovery, kCGLPFAAccelerated, 96, kCGLPFARendererID, 16918024}, 0x13c398a0, 1);
CGLCreateContext(0x13c398a0, 0x00000000, 0x00825800);
CGLSetSurface(0x00825800, {20, 42, 1024, 1024}, {0, 0, 1024, 1024});
glScissor(0, 0, 1024, 1024);
glViewport(0, 0, 1024, 1024);
glViewport(0, 0, 1024, 1024);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(0, 1024, 0, 1024, -1, 1);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glClearColor(0, 0, 0, 0);
glClear(GL_COLOR_BUFFER_BIT);
glGetString(GL_RENDERER);
glGetString(GL_VERSION);
CGLGetVirtualScreen(0x00825800);
CGLGetVirtualScreen(0x00825800);
CGLQueryRendererInfo(2, 0x000cb820, -1730100884);
CGLDescribeRenderer(0x13c2cae0, 0, kCGLRPAccelerated);
CGLDescribeRenderer(0x13c2cae0, 0, kCGLRPVideoMemory);
CGLDescribeRenderer(0x13c2cae0, 1, kCGLRPAccelerated);
CGLDestroyRendererInfo(0x13c2cae0);
glGetFloatv(GL_MAX_VIEWPORT_DIMS, 0xbffff148);
glGetFloatv(GL_MAX_RECTANGLE_TEXTURE_SIZE_EXT, 0xbffff184);
glGetIntegerv(GL_MAX_3D_TEXTURE_SIZE, 0x00246b40);
glGetString(GL_VERSION);
glGetString(GL_EXTENSIONS);
glGetString(GL_RENDERER);
glGetProgramivARB(GL_VERTEX_PROGRAM_ARB, GL_MAX_PROGRAM_NATIVE_PARAMETERS_ARB, 0xbffff178);
glGetProgramivARB(GL_VERTEX_PROGRAM_ARB, GL_MAX_PROGRAM_NATIVE_TEMPORARIES_ARB, 0xbffff170);
glGetProgramivARB(GL_VERTEX_PROGRAM_ARB, GL_MAX_PROGRAM_NATIVE_ATTRIBS_ARB, 0xbffff174);
glGetProgramivARB(GL_VERTEX_PROGRAM_ARB, GL_MAX_PROGRAM_LOCAL_PARAMETERS_ARB, 0xbffff16c);
glGetProgramivARB(GL_VERTEX_PROGRAM_ARB, GL_MAX_PROGRAM_NATIVE_INSTRUCTIONS_ARB, 0xbffff168);
glGetProgramivARB(GL_VERTEX_PROGRAM_ARB, GL_MAX_PROGRAM_NATIVE_ALU_INSTRUCTIONS_ARB, 0xbffff164);
glGetProgramivARB(GL_VERTEX_PROGRAM_ARB, GL_MAX_PROGRAM_NATIVE_TEX_INSTRUCTIONS_ARB, 0xbffff160);
glGetProgramivARB(GL_VERTEX_PROGRAM_ARB, GL_MAX_PROGRAM_NATIVE_TEX_INDIRECTIONS_ARB, 0xbffff15c);
glGetIntegerv(GL_MAX_TEXTURE_COORDS_ARB, 0xbffff154);
glGetIntegerv(GL_MAX_TEXTURE_IMAGE_UNITS_ARB, 0xbffff158);
```

All Together Now:

... Core Image will do all this OpenCL work

```
aa = clCreateContextFromType( NULL,  CL_DEVICE_TYPE_CPU, NULL, NULL, 0);
clRetainContext(aa);
clGetContextInfo(aa, CL_CONTEXT_DEVICES, 0, NULL, 4);
clGetContextInfo(aa, CL_CONTEXT_DEVICES, 4, { bb }, 4);
clGetDeviceInfo(bb, CL_DEVICE_TYPE, 8, 2, 8);
clGetDeviceInfo(bb, CL_DEVICE_MAX_COMPUTE_UNITS, 4, 4, 4);
cc = clCreateCommandQueue(aa, bb, CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE | CL_QUEUE_PROFILING_ENABLE, 0)
clGetDeviceInfo(bb, CL_DEVICE_MAX_SAMPLERS, 4, 16, 0);
clGetDeviceInfo(bb, CL_DEVICE_MAX_PARAMETER_SIZE, 4, 4096, 0);
clCreateSampler(aa, CL_FALSE, CL_ADDRESS_CLAMP, CL_FILTER_LINEAR, 0);
clCreateSampler(aa, CL_FALSE, CL_ADDRESS_CLAMP_TO_EDGE, CL_FILTER_LINEAR, 0);
clCreateSampler(aa, CL_FALSE, CL_ADDRESS_REPEAT, CL_FILTER_LINEAR, 0);
gg = clCreateSampler(aa, CL_FALSE, CL_ADDRESS_CLAMP, CL_FILTER_NEAREST, 0);
clCreateSampler(aa, CL_FALSE, CL_ADDRESS_CLAMP_TO_EDGE, CL_FILTER_NEAREST, 0);
clCreateSampler(aa, CL_FALSE, CL_ADDRESS_REPEAT, CL_FILTER_NEAREST, 0);
clCreateSampler(aa, CL_TRUE, CL_ADDRESS_CLAMP, CL_FILTER_LINEAR, 0);
clCreateSampler(aa, CL_TRUE, CL_ADDRESS_CLAMP_TO_EDGE, CL_FILTER_LINEAR, 0);
clCreateSampler(aa, CL_TRUE, CL_ADDRESS_REPEAT, CL_FILTER_LINEAR, 0);
clCreateSampler(aa, CL_TRUE, CL_ADDRESS_CLAMP, CL_FILTER_NEAREST, 0);
clCreateSampler(aa, CL_TRUE, CL_ADDRESS_CLAMP_TO_EDGE, CL_FILTER_NEAREST, 0);
clCreateSampler(aa, CL_TRUE, CL_ADDRESS_REPEAT, CL_FILTER_NEAREST, 0);
clRetainContext(aa);
clRetainCommandQueue(cc);
hh = clCreateImage2D(aa,  CL_MEM_READ_WRITE | CL_MEM_USE_HOST_PTR ,  { CL_ARGB , CL_UNORM_INT8 }, 1024, 1024, 4096, pelon, 0);
clRetainMemObject(hh);
clGetImageInfo(hh, CL_IMAGE_ROW_PITCH, 4, <param_value>, 0);
clGetImageInfo(hh, CL_IMAGE_WIDTH, 4, <param_value>, 0);
clGetImageInfo(hh, CL_IMAGE_HEIGHT, 4, <param_value>, 0);
clGetImageInfo(hh, CL_IMAGE_DEPTH, 4, <param_value>, 0);
clGetImageInfo(hh, CL_IMAGE_FORMAT, 8, <param_value>, 0);
clReleaseMemObject(hh);
clGetDeviceInfo(bb, CL_DEVICE_IMAGE2D_MAX_WIDTH, 4, 8192, 0);
clGetDeviceInfo(bb, CL_DEVICE_IMAGE2D_MAX_HEIGHT, 4, 8192, 0);
clGetDeviceInfo(bb, CL_DEVICE_MAX_MEM_ALLOC_SIZE, 8, 1073741824, 0);
clGetDeviceInfo(bb, CL_DEVICE_MAX_SAMPLERS, 4, 16, 0);
clGetDeviceInfo(bb, CL_DEVICE_MAX_READ_IMAGE_ARGS, 4, 128, 0);
clGetDeviceInfo(bb, CL_DEVICE_MAX_SAMPLERS, 4, 16, 0);
clGetDeviceInfo(bb, CL_DEVICE_MAX_READ_IMAGE_ARGS, 4, 128, 0);
clGetDeviceInfo(bb, CL_DEVICE_MAX_SAMPLERS, 4, 16, 0);
clGetDeviceInfo(bb, CL_DEVICE_MAX_READ_IMAGE_ARGS, 4, 128, 0);
```

Using Core Image Efficiently

Daniel Eggert
Software Engineer

Using Core Image Efficiently

- Demo application
- Five things to keep in mind
- Debugging tips

Demo Application

A simple Cocoa application using Core Image

- Opens images
- Custom NSView
- Applies the CIPointillize filter to the image inside -drawRect:

Available as sample code at:
http://developer.apple.com/wwdc/attendee/

Demo

Daniel Eggert
Software Engineer

Five Things to Keep in Mind

1. NSImage vs. CGImageRef
2. CPU vs. GPU
3. Reuse CIContext instances
4. Color Management
5. Threading

1 NSImage

Cocoa image object

- Both source and destination
- Mutable pixel container
- Content can change to adapt it to given rendering situation
- Can abstract one or many images of different types

1 Use CGImageRef

Quartz image object

- Immutable pixel container
- Exactly one bitmap based image
- Best fidelity for image processing

CPU vs. GPU

Both have their place

- CPU
  - Fidelity
  - Background Friendly
- GPU
  - Performance
  - Offloads the CPU

② CPU vs. GPU

When to use which?

- CPU
  - Good for Saving
- GPU
  - Good for Interacting

2 Choosing the CPU

- Force usage of the CPU:

```
options = [NSDictionary dictionaryWithObject:[NSNumber numberWithBool:YES]
                                      forKey:kCIContextUseSoftwareRenderer];
```

```
context = [CIContext contextWithCGContext:cgContext
                                  options:options];
```

- If not specified, runs on the GPU
- On 10.6 and later the software renderer uses OpenCL (on the CPU)

3 Keep Your CIContext Around

- CIContext instances hold onto a lot of state and caches.
- Re-using CIContext instance is usually the single change leading to the largest performance win

3 NSView and CIContext

Inside an NSView's -drawRect:

- Using

```
CIContext *context = [[NSGraphicsContext currentContext] CIContext];
```

will do the right thing

3 Retain Your CIContext

When you create your own instance

- Store it in a member variable:

```
if (context == nil) {
    context = [CIContext contextWithCGContext:cgContext options:options];
    context = [context retain];
}
```

- Re-use it. Often

- Release it when you're done:

```
[context release];
```

4 Color Management

Basic story

- Automatic color management
- Respects input images' color space
- Respects destination context's color space
- Filters are applied in a linear working space

4 Disabling Color Management

You can turn it off

- Pass in [NSNull null] as the color space

```
options = [NSDictionary dictionaryWithValue:[NSNull null]
                                     forKey:kCIImageColorSpace];
image = [CIImage imageWithCGImage:cgImage
                          options:options];
```

```
options = [NSDictionary dictionaryWithValue:[NSNull null]
                                     forKey:kCIContextOutputColorSpace];
image = [CIContext contextWithCGContext:cgImage
                                options:options];
```

4 Display Color Profile Changes

Two reasons for change

- Window dragged to another display
- Changing display profile in System Preferences

4 Being Prepared for Change

- Call

```
[window setDisplaysWhenScreenProfileChanges:YES];
```

to make sure all views are redrawn

- Inside

```
- (void)drawRect:(NSRect)dirtyRect;
```

use

```
CIContext *context = [[NSGraphicsContext currentContext] CIContext];
```

- Works for multi-display systems. Does not work for user changing the display profile in the system preferences

Being Prepared for Change

Invalidating off screen caches

- The window's delegate needs to implement

```
- (void)windowDidChangeScreenProfile:(NSNotification *)aNotificaton;
```

- Or register for

```
center = [NSNotificationCenter defaultCenter];
token = [center addObserverForName:NSWindowDidChangeScreenProfileNotification
                            object:nil
                             queue:nil
                        usingBlock:^(NSNotification *aNotification){
                                // Clear cache here.
                            }];
[token retain];
```

5 Threading

- Core Image uses all available cores for CPU contexts
- Multi-threading “for free”

5 Threading

- Calling into Core Image on multiple threads:
  - Use separate instances of CIContext for each thread, or
  - Use proper locking

5 Running on a Background Thread

Increase your stack size

- Core Image actively uses the stack
- Increase the stack size of background threads if you have complex filter trees

```
thread = [[NSThread alloc] initWithTarget:target
                                 selector:@selector(runWithObject:)
                                   object:object];
[thread setStackSize:64 * 1024 * 1024]; // 64 MB
```

Probing—What Is Core Image up to?

Printing render tree to console

- Allows you to see what Core Image is doing
- Console output each time an image is drawn into a context

Behind the Scenes

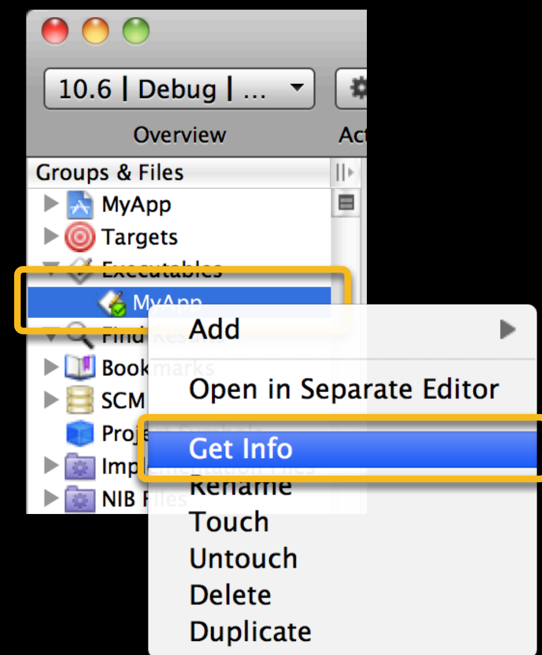
Enabling tree printing

- Set the CI_PRINT_TREE environment variable to 1
- Each "draw" will dump the filter tree that is being drawn
- Large output sizes might cause multiple draws due to tiling

Setting CI_PRINT_TREE

- Select the executable in the "Groups & Files" table

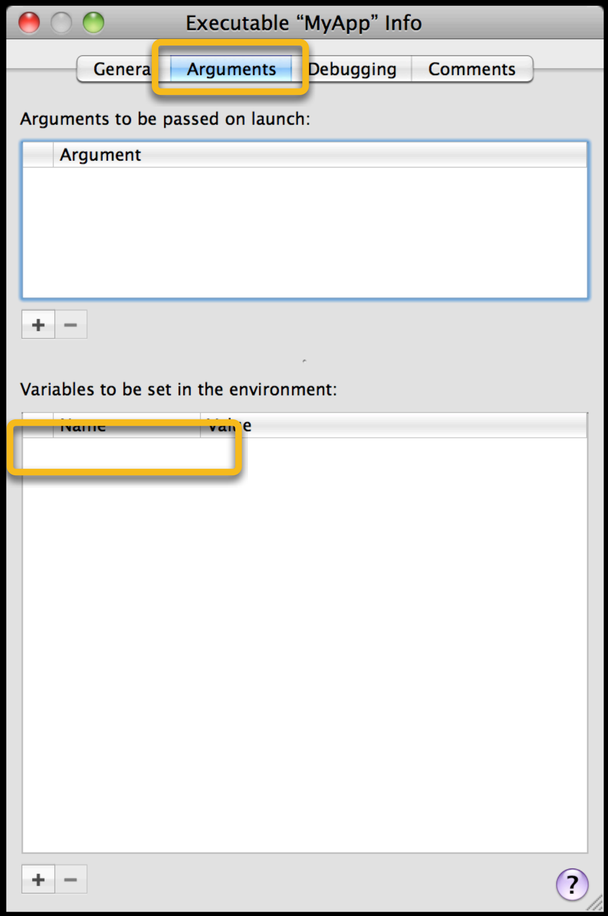
- Right-click and select "Get Info"



Setting CI_PRINT_TREE

- Select the "Arguments" tab
- Add an environment variable

```
CI_PRINT_TREE 1
```



What You Get

```
** v7 fe-context-cl-cpu 0x2b33200, pass 68, rendering **
```

```
APPLY (_CIClampToAlpha _CIPremultiply) DOD [0,0 36.5x54.75] ROI [0,0 36x54] ARGB_8 #4
 AFFINE [0.5 0 0 0.502294 0 0] DOD [0,0 37x55] ROI [0,0 36x54] ARGB_8 #3
  APPLY _CIDownsample DOD [0,0 73x109] ROI [0,0 73x109] ARGB_8 #2
   IMAGE CIImage:0x1e7fc980 DOD [0,0 292x438] ROI [0,0 292x438] ARGB_8 #1
```

```
** v7 fe-context-gl 0x188daa00, pass 69, rendering **
```

```
APPLY (_CIColorCurve4 _CIMatrixNobias) ROI [0,0 1200x900] RGBA_F #5
 OVER ROI [0,0 1200x900] RGBA_H #4
  APPLY (_CIMatrixNobias _CIColorCurve3) DOD [0,0 1200x900] opaque ROI [0,0 1200x900]
RGBA_H #2
   IMAGE CIImage:0x100296760 DOD [0,0 1200x900] opaque ROI [0,0 1200x900] ARGB_8 #1
  FILL opaque ROI [0,0 1200x900] RGBA_H #3
```

What You Get

```
** v7 fe-context-cl-cpu 0x2b33200, pass 68, rendering **
```

```
APPLY (_CIClampToAlpha _CIPremultiply) DOD [0,0 36.5x54.75]
ROI [0,0 36x54] ARGB_8 #4
 AFFINE [0.5 0 0 0.502294 0 0] DOD [0,0 37x55] ROI [0,0 36x54] ARGB_8 #3
  APPLY _CIDownsample DOD [0,0 73x109] ROI [0,0 73x109] ARGB_8 #2
   IMAGE CIImage:0x1e7fc980 DOD [0,0 292x438] ROI [0,0 292x438] ARGB_8 #1
```

- CIImage: Size of the input image(s)
- “cl-cpu” denotes an OpenCL CPU context

What You Get

```
** v7 fe-context-gl 0x11382ae00, pass 5, rendering **

APPLY (_CIColorCurve4 _CIMatrixNobias) ROI [0,0 1200x900] RGBA_F #5
 OVER ROI [0,0 1200x900] RGBA_H #4
  APPLY (_CIMatrixNobias _CIColorCurve3) DOD [0,0 1200x900] opaque ROI [0,0
1200x900] RGBA_H #2
   IMAGE CIImage:0x100296760 DOD [0,0 1200x900] opaque ROI [0,0 1200x900] ARGB_8 #1
  FILL opaque ROI [0,0 1200x900] RGBA_H #3
```

- CIImage: Size of the input image
- “gl” denotes a GPU context

Number of Renders

- Identify the render invocations by their input images
- Check if the number of renders matches your expectations

Is Core Image Using the CPU or GPU?

- Identify the render invocations by their input images
- Check for cl-cpu/gl

```
** v7 fe-context-cl-cpu 0x101034a08, pass 4, rendering **
```

  - CPU

```
** v7 fe-context-gl 0x11382ae00, pass 5, rendering **
```

  - GPU

Writing CIFilters

Begin at the beginning

Alexandre Naaman
Lead Drastefarian

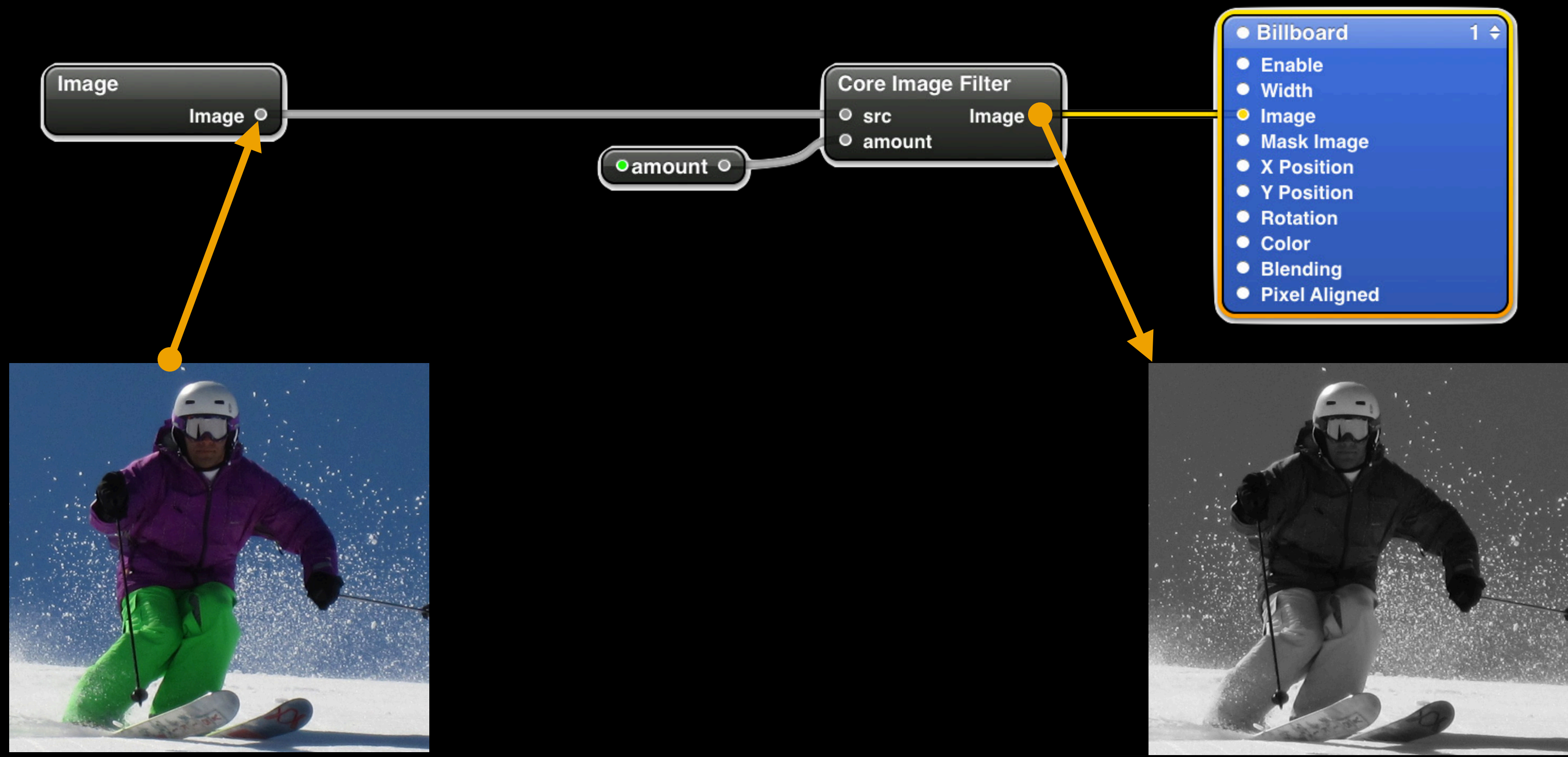
Why Write Your Own Core Image Filter?

- Filter does not exist in the existing set
- Effect cannot be produced by chaining existing filters

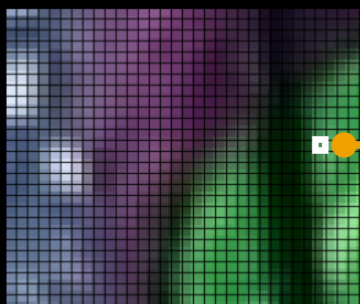
Where to Begin

- Typical method for developing imaging algorithms
  - Quartz Composer + kernel code
  - Objective-C code + kernel code

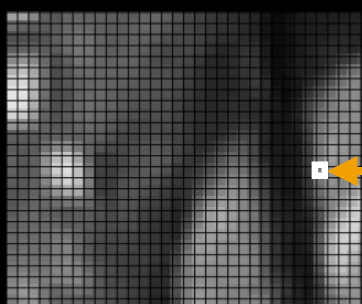
Using Quartz Composer for Rapid Prototyping (Desaturation Filter)



Write Your Own Kernel



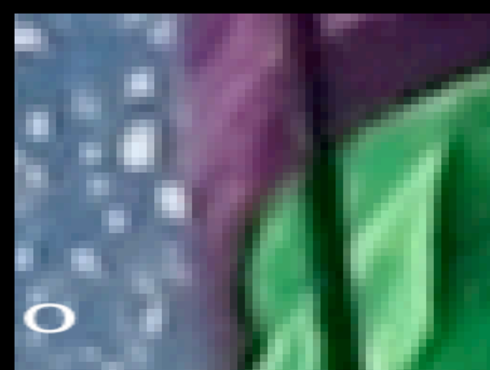
Source image subrect



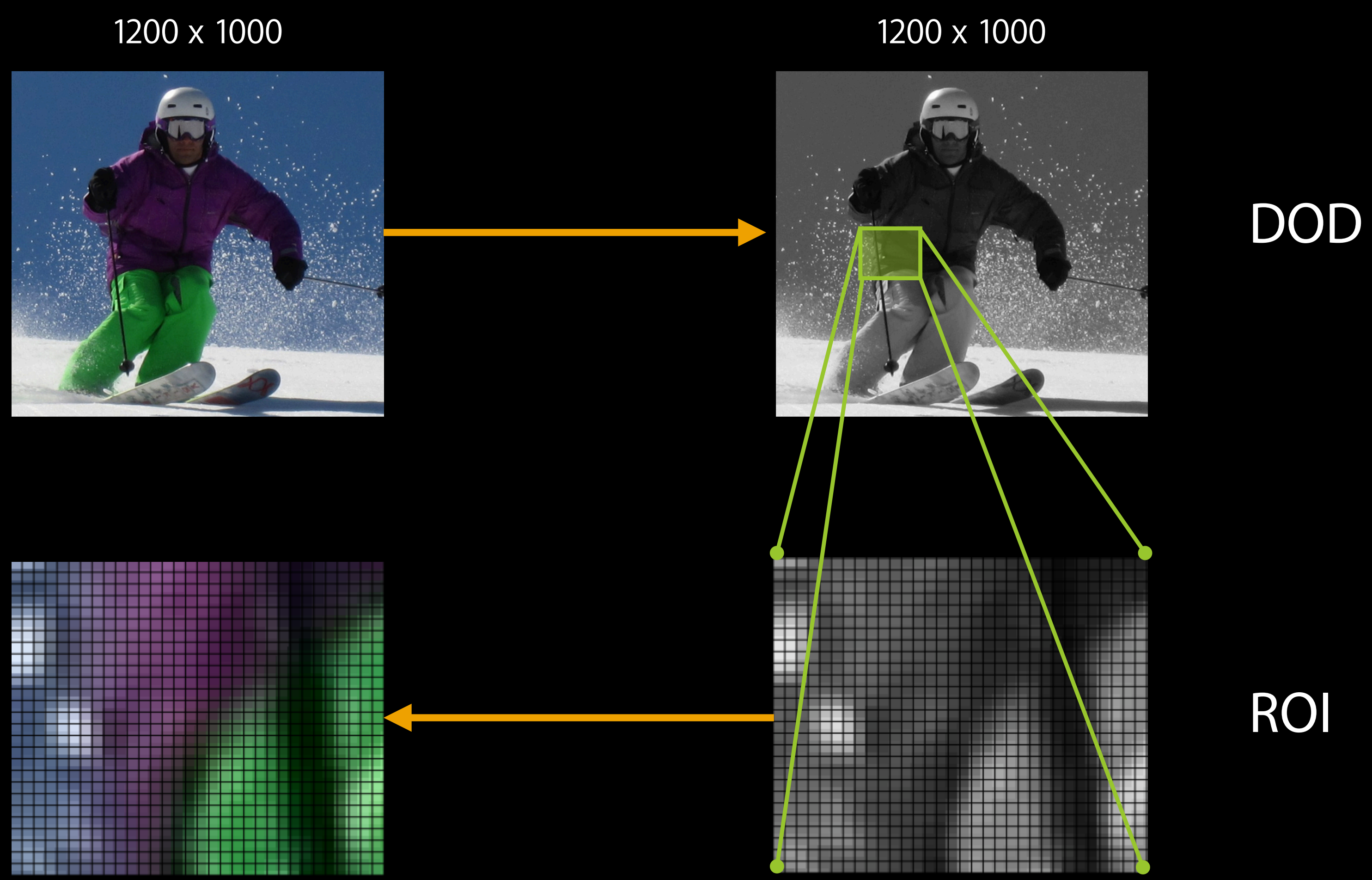
Destination image subrect

for each pixel, the kernel gets asked to produce output pixel at X,Y

```
kernel vec4 desaturate ( sampler src, float amount )
{
    vec4 orig = unpremultiply (sample(src,samplerCoord(src)));
    float Y = orig.r * 0.222 + orig.g * 0.717 + orig.b * 0.061;
    vec4 desaturatedColor = vec4 (Y, Y, Y, orig.a);
    return premultiply ( mix (orig, desaturatedColor, amount));
}
```



Domain of Definition & Region of Interest

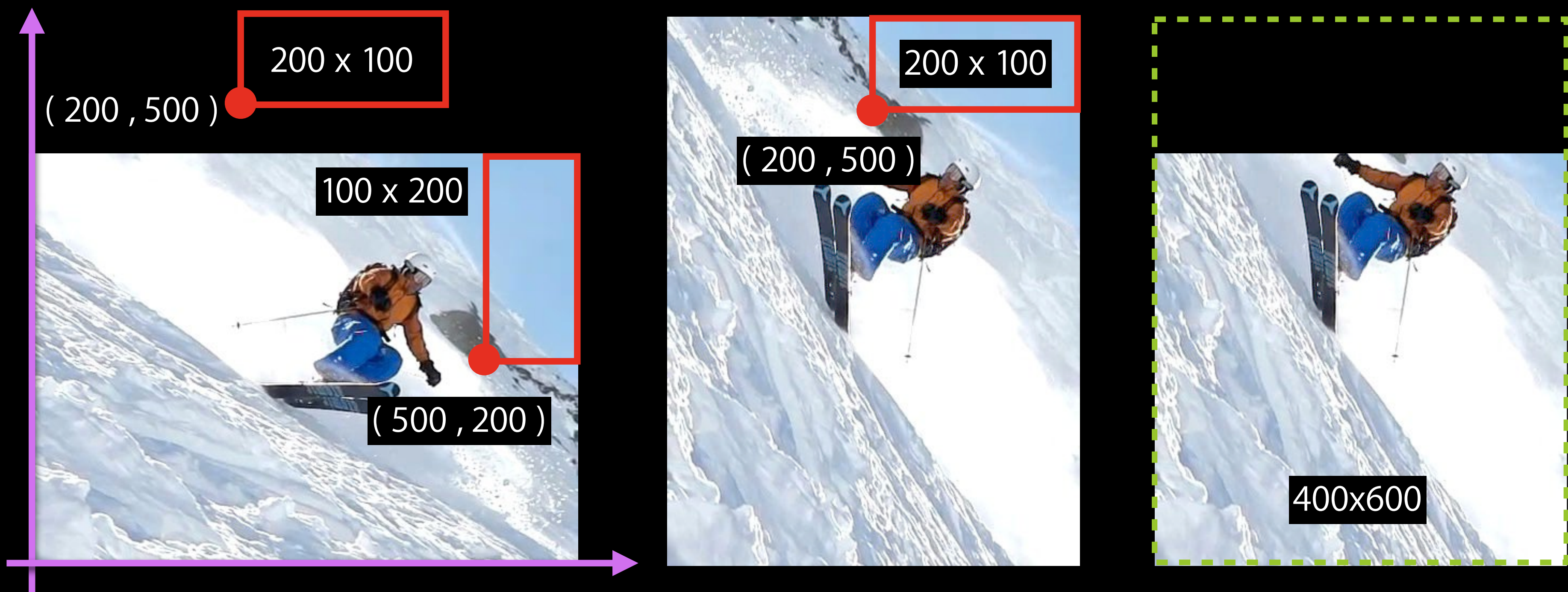


DOD for Transpose (x , y) ➟ (y , x)

```
kernel vec4 transpose ( sampler image )
{ return sample ( image, samplerTransform ( image, destCoord().yx ) ); }
```


```
CGRect r = [inputImage extent];
CIFilterShape *shape = [CIFilterShape shapeWithRect:
      CGRectMake ( r.origin.y, r.origin.x, r.size.height, r.size.width )];
CIImage *outputImage = [self apply: ..... options:
                           options: kCIApplyOptionDefinition, shape, nil];
```

ROI for Transpose Filter




```
- (CGRect) regionOf:(int)samplerIdx destRect:(CGRect)r userInfo:(id)i
{
   return CGRectMake ( r.origin.y,    r.origin.x,
                       r.size.height, r.size.width );
}
```

The Droste Effect

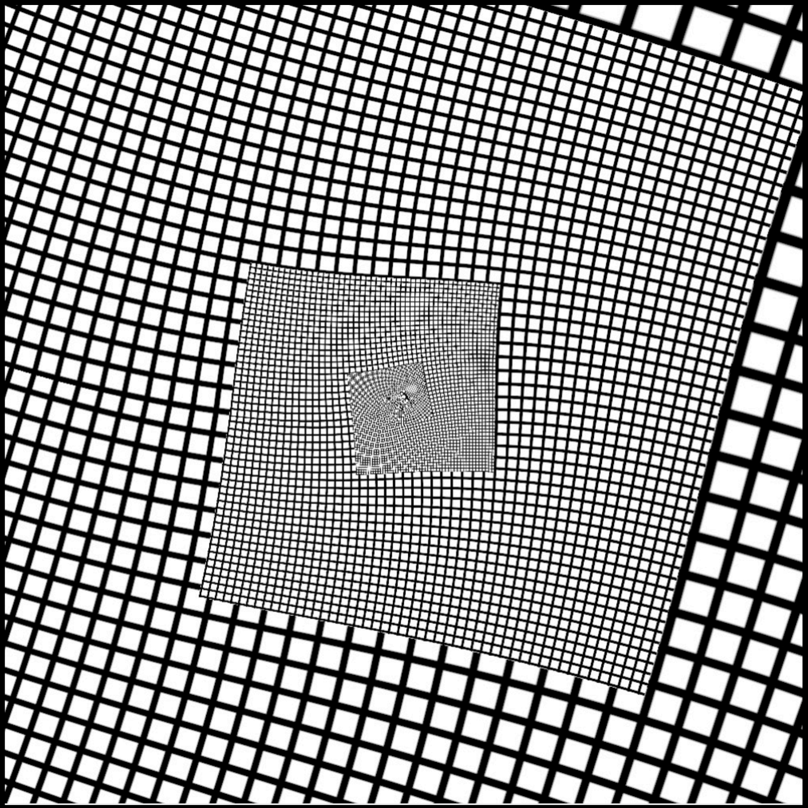
A “Complex” filter example

Droste Filter Sample

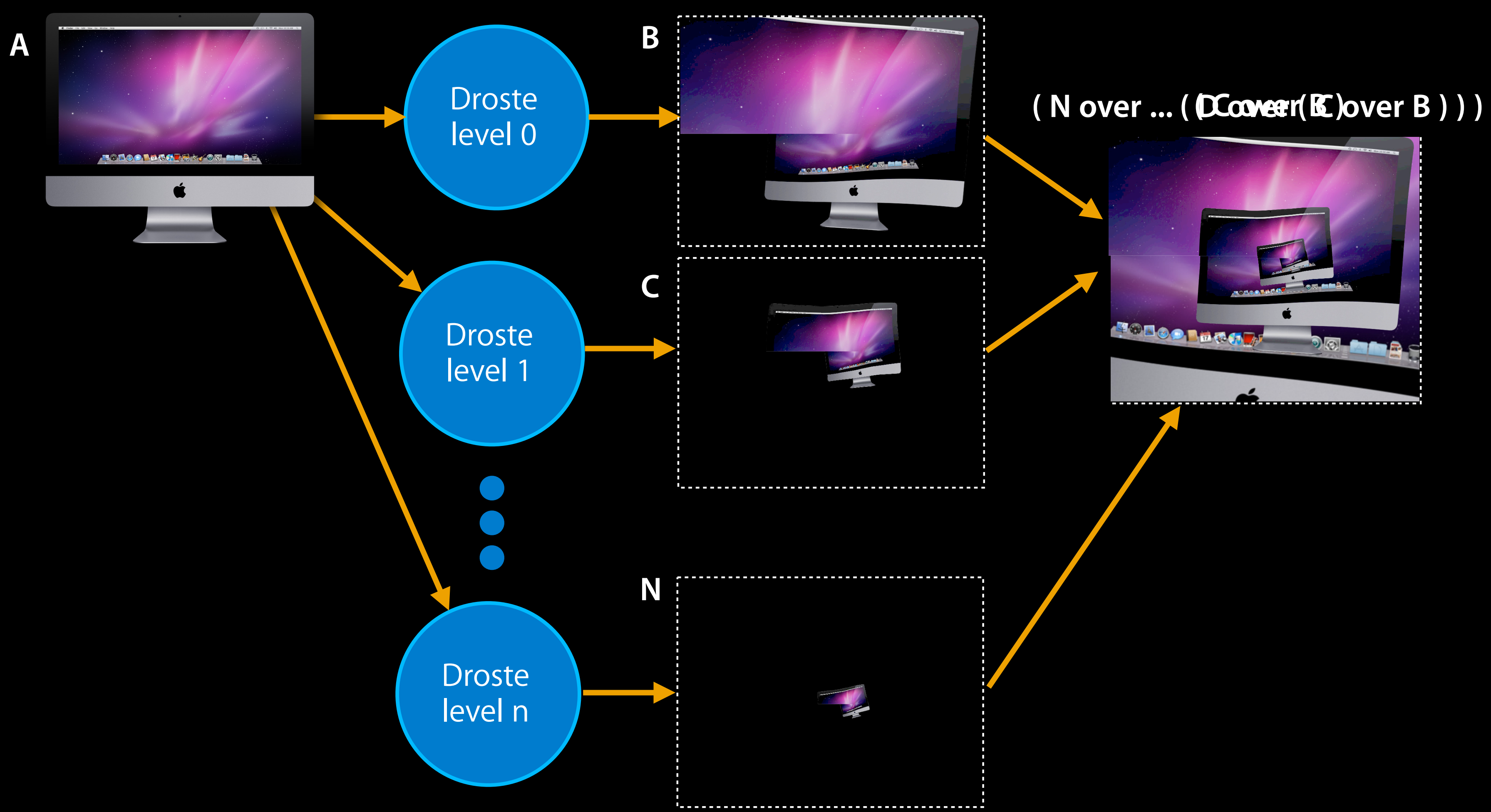
- Original idea from M.C. Escher (1898 -1972)
- Effect named after Dutch chocolate (Cocoa!)
- http://developer.apple.com/mac/library/samplecode/Droste

Let’s Animate the Deformation

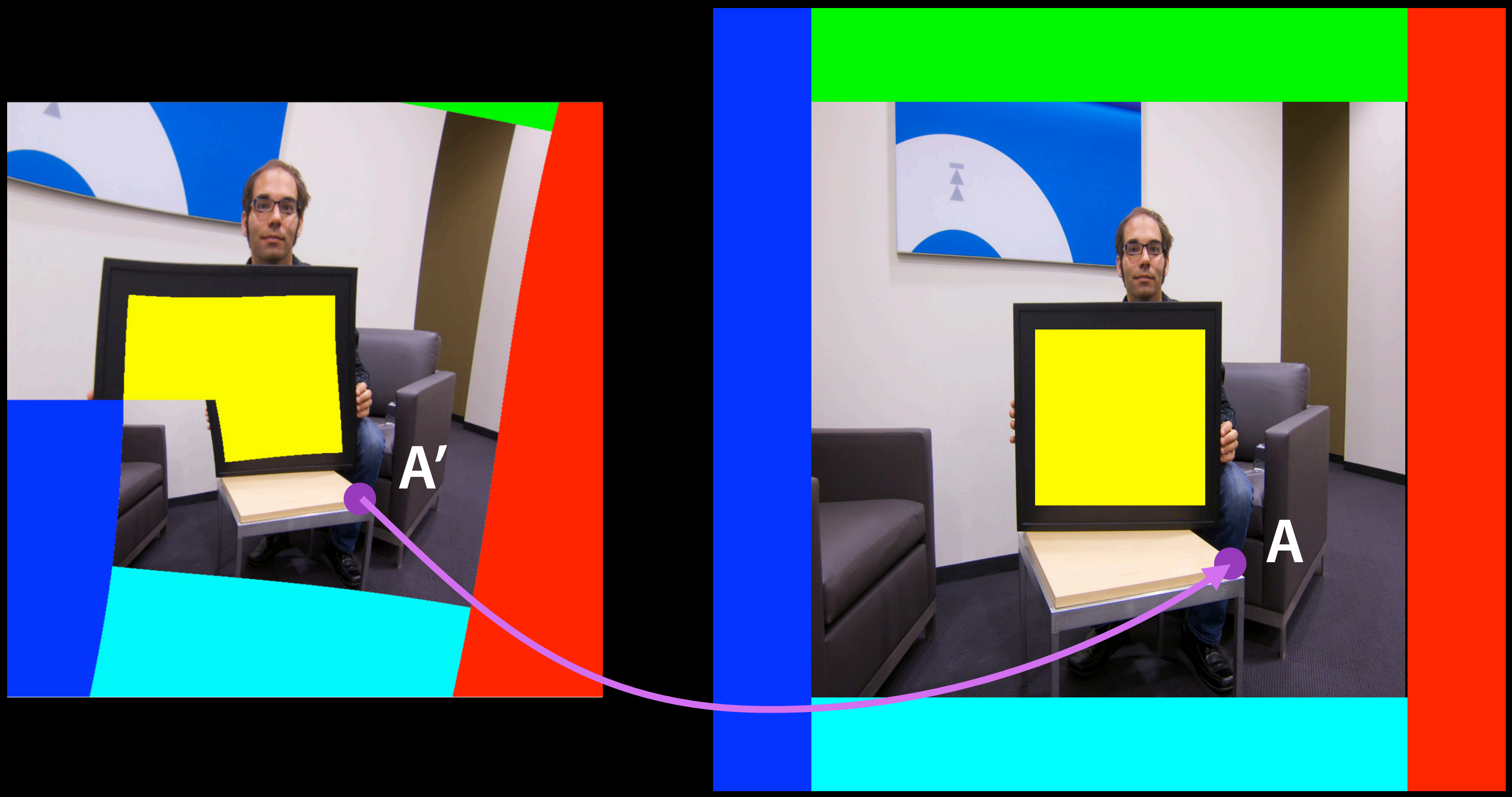
Let’s Assemble the Layers



A First Approach: Source Over



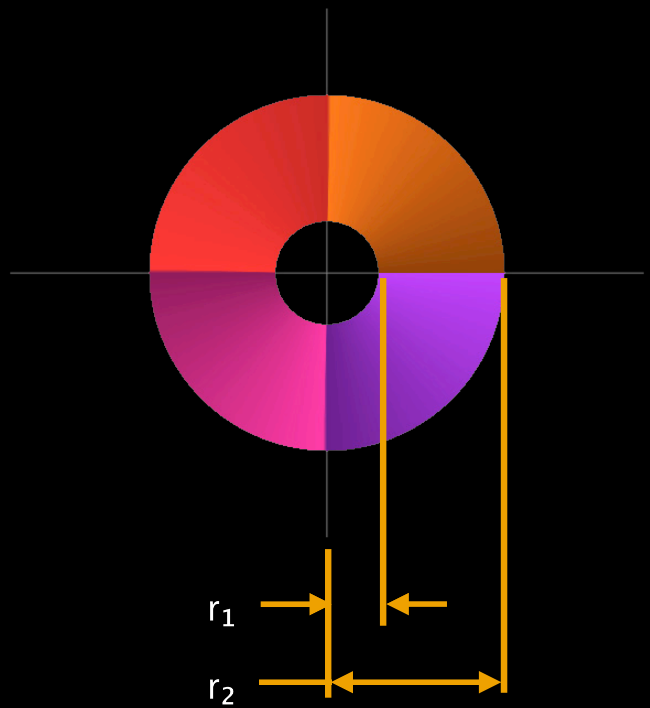
Where's the Source?



Output image

Input image

The Math Behind the Madness



Logarithmic spiral in polar coordinates:

$$r = a\, e^{b\theta}$$

when $\theta == 0$, $e^0 == 1$ therefore $a = r_1$

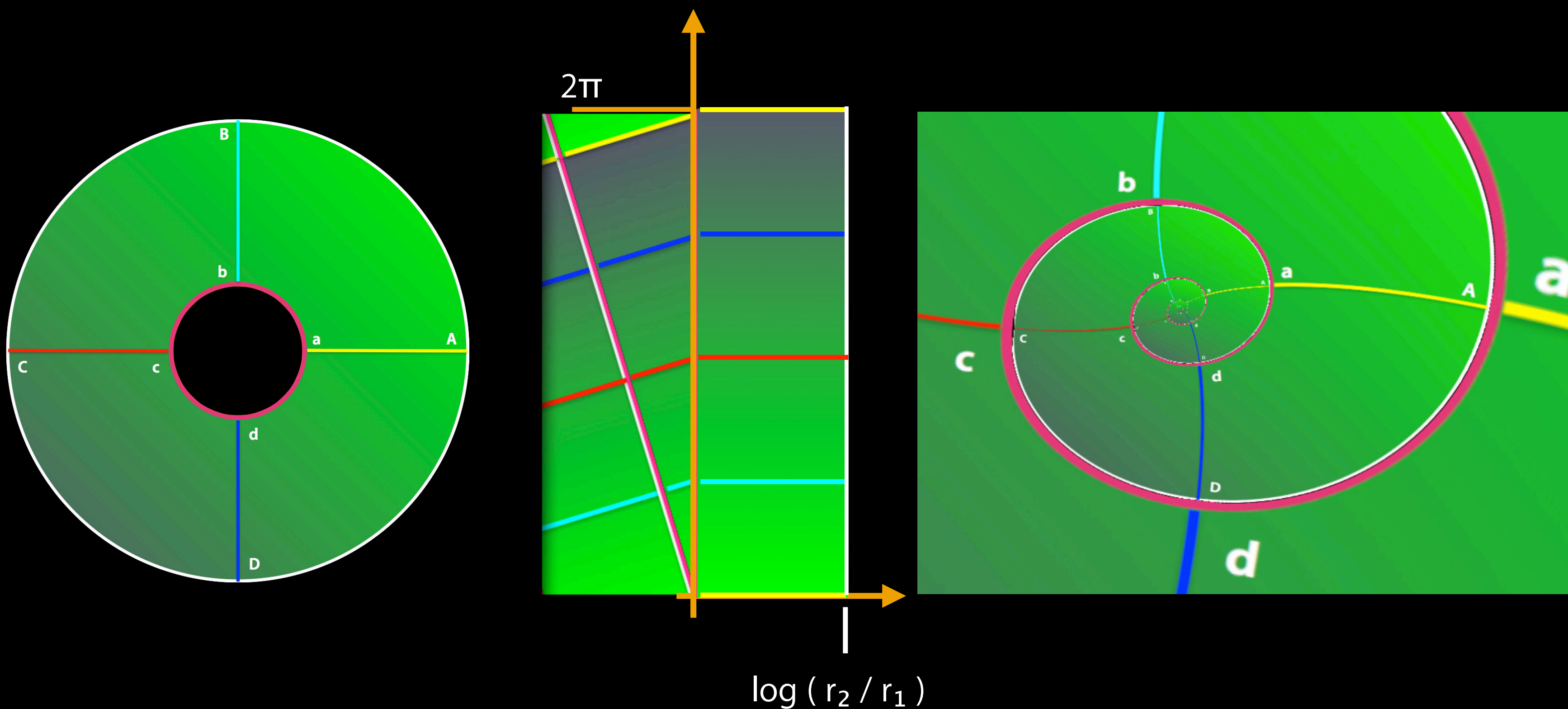
when $\theta == 2\pi$:

$$r_2 = r_1\, e^{b\, 2\pi}$$

$$b = \log\,(\, r_2 \,/\, r_1) \,/\, 2\pi$$

Too easy, not a conformal map;
doesn't preserve angles

Coordinate Transforms

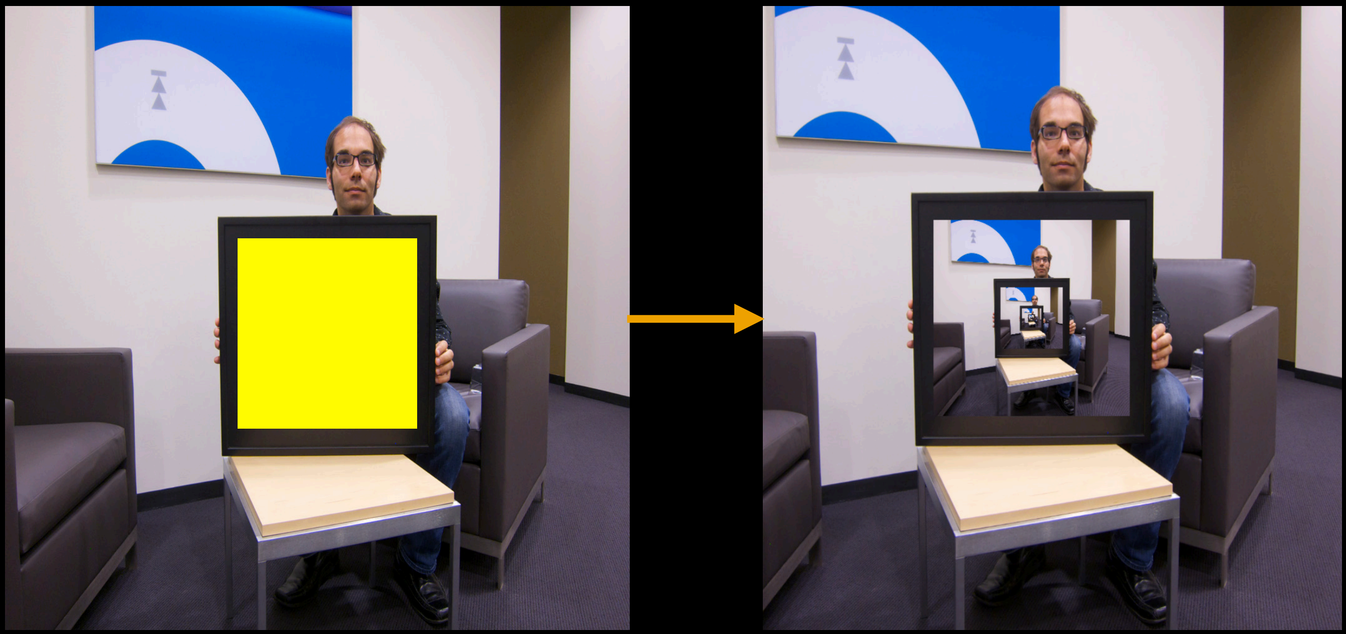


Kernel Code

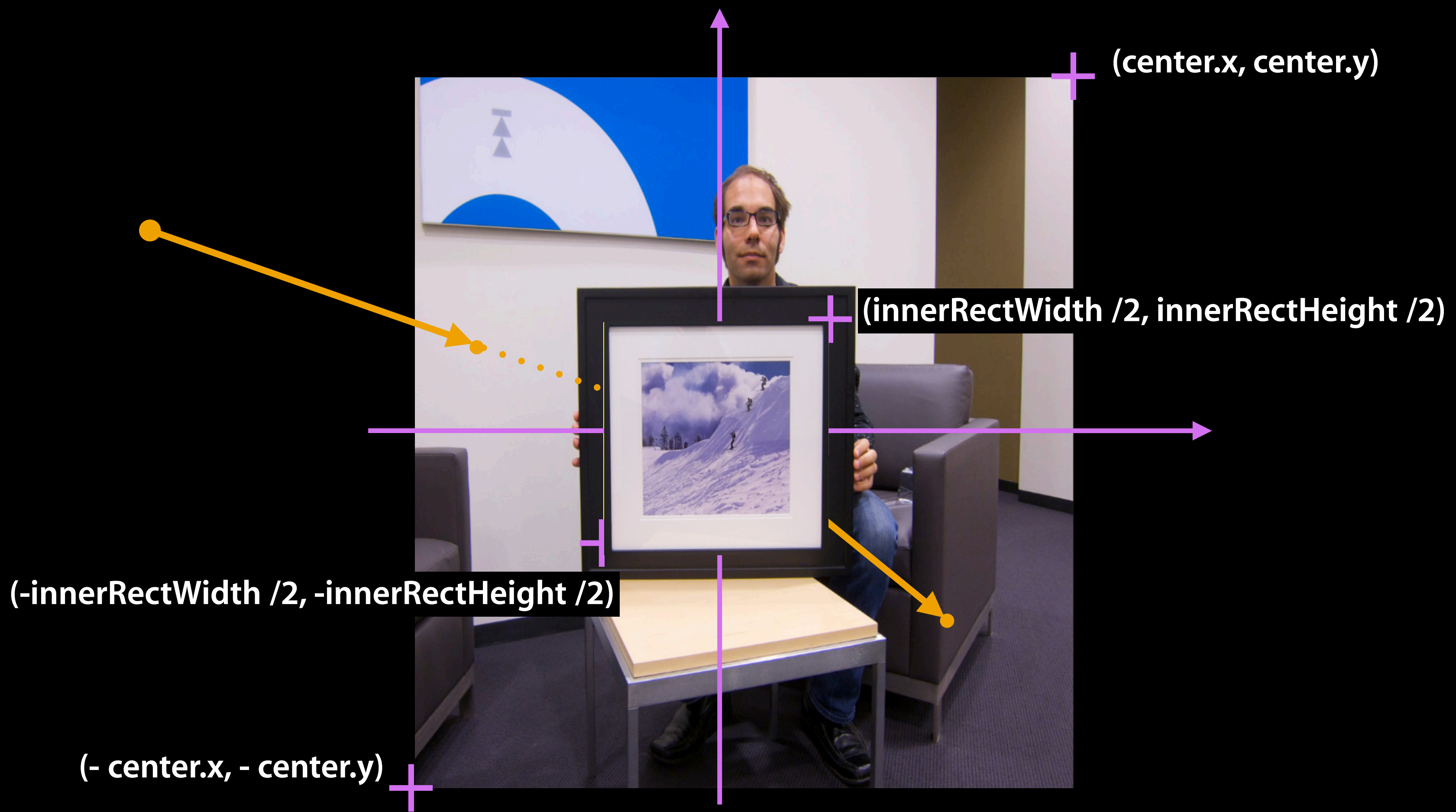
```
// r.x = a
// r.y = log ( r₂ / r₁ ) / 2π
// scale = imageWidth / innerRectWidth

kernel vec4 droste ( sampler image, vec2 center, vec2 r, float scale )
{
  vec2 distorted = destCoord() - center;
  float  theta = atan ( distorted.y, distorted.x );
  vec2 polar = vec2( 0.5*log ( distorted.x * distored.x +
                               distorted.y * distorted.y ) , theta );

  vec2 rotated = vec2 ( polar.x * r.x - polar.y * r.y,
                        polar.x * r.y + polar.y * r.x );

  vec2 coord = vec2 ( exp ( rotated.x ) * cos ( rotated.y ),
                      exp ( rotated.x ) * sin ( rotated.y ) );

  return sample ( image, coord * scale + center );
}
```

First Stab at Implementing Algorithm

- Works, but requires many passes over the data
- Can we do the hall of mirrors effect in a single pass (b == 0) ?



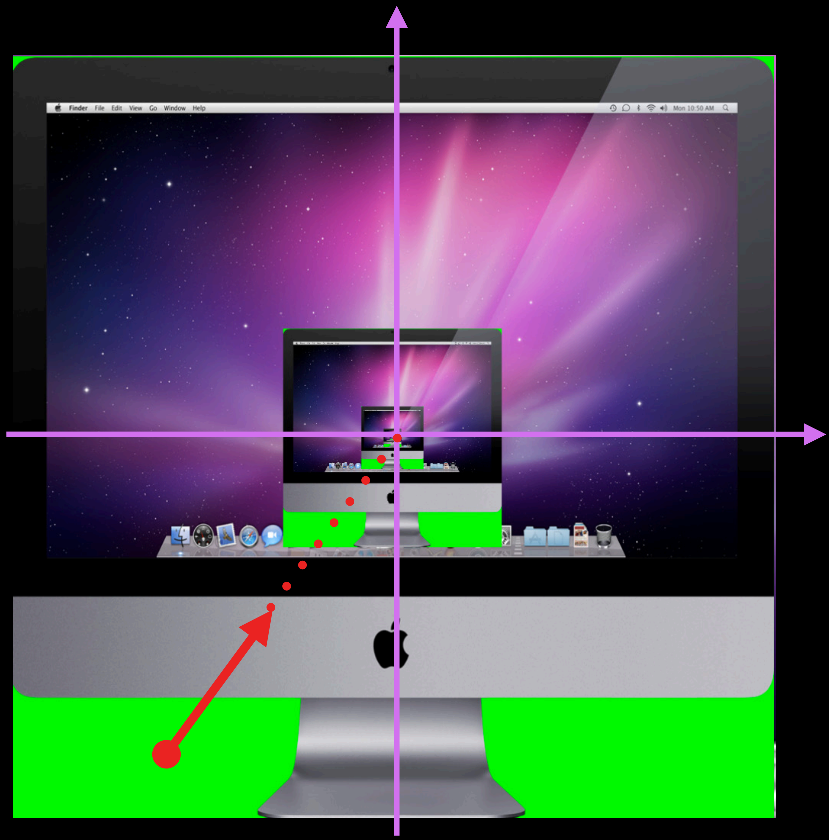
Hall of Mirrors in a Single Pass



Le Code

```
kernel vec4 droste ( sampler image, vec2 center, vec2 r, float scale,
                     vec2 halfInnerRect )
{
   /* ... same code as before ... */
   int i;

   for ( i = 0; i < 16; i++ )
     coord /=
       coord.x >  center.x ? scale :
       coord.y >  center.y ? scale :
       coord.x < -center.x ? scale :
       coord.y < -center.y ? scale : 1.0 ;

   for ( i = 0; i < 16;  i++ )
      coord *=
        coord.x < -halfInnerRect.x ? 1.0 :
        coord.x >  halfInnerRect.x ? 1.0 :
        coord.y < -halfInnerRect.y ? 1.0 :
        coord.y >  halfInnerRect.y ? 1.0 : scale ;

  return sample ( image, coord + center );
}
```

Alpha Blending



Le Code, Part Deux

- Porter Duff alpha blending

```
kernel droste ( ... ) { // all the same once again
  vec4 color = sample ( image, coord + center );

  float s = scale; // current scale factor

  for ( i = 0; i < 4; i++ )
  {
    color = color.a < 1.0 ?
       color * color.a + (1.0 - color.a) * sample (image, coord / s + center ) :
       color;

     s *= scale; // new scale factor for following iteration
  }

  return color;
}
```

A Few Caveats/Notes on Performance

- ROI not tile-able
- Could use scale + rotate of initial Droste level to generate subsequent levels

  $alpha = \arctan(\log(r_2 / r_1)) / 2\pi$
  $scale = \cos(alpha)$

- Single pass method could use some anti-aliasing in order to get rid of hard edges along borders of inner rectangle
- Be sure to center & scale image to match inner rectangle before performing effect!

Demo of Droste Effect

A Few More Resources Online

- More madness (aka: math)
  http://escherdroste.math.leidenuniv.nl/
- American Mathematical Society paper
  Artful Mathematics: The Heritage of M. C. Escher
  by B. de Smit and H. W. Lenstra Jr.
- A gentler approach from Jos Leys:
  http://www.josleys.com/article_show.php?id=82
- GLSL quick reference:
  http://www.khronos.org/files/opengl-quick-reference-card.pdf

Throwing the Book(s) at it

- "Digital Image Warping"
  - George Wolberg, 1992
- "GPU Gems 3"
  - Addison-Wesley, 2007
- "Digital Image Processing"
  - Pearson Prentice Hall, 2008

Labs

Core Image	Graphics & Media Lab A Thursday 4:30PM

More Information

Allan Schaffer
Graphics and Imaging Evangelist
aschaffer@apple.com

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

WWDC10