Quartz 2D

Tópicos selecionados de iOS 5

Helder da Rocha

Objetivos

- **Parte I**
  - Introduzir os conceitos essenciais de Quartz 2D (esta apresentação)
    - Demonstrar cada conceito separadamente
- **Parte II**
  - Introdução prática de Quartz2D através de um exemplo passo-a-passo (no XCode, em sala de aula)

Pré-requisitos

- Conhecimento básico de C (+Core Foundation) e Objective-C (+Foundation)
- Experiência prática elementar em Cocoa ou iOS (você deve ter escrito alguns programas simples para Mac ou iPhone)
- Saber usar o XCode no seu ambiente

Quartz 2D

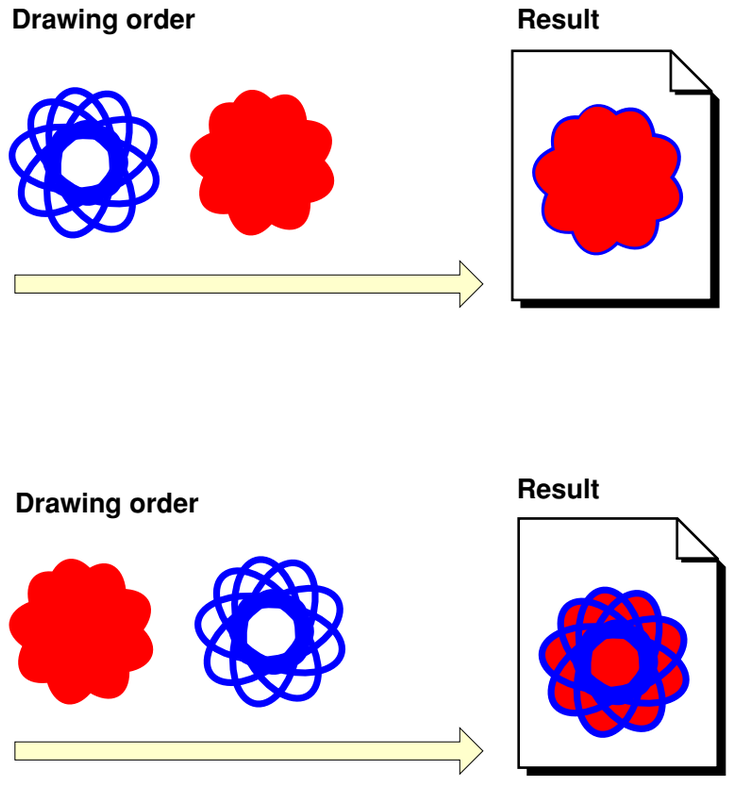
- Parte de Core Graphics
- Framework independente de resolução e de dispositivo
- Para
  - desenhar
  - permitir edição gráfica
  - criar ou exibir imagens
- Principal fonte desta apresentação:
  - Drawing With Quartz 2D (Apple)

Sumário

1. Quartz 2D

- Página: "painter's model"
  - Cada operação de desenho "pinta" em uma "página"
  - Pintura pode ser modificada aplicando-se outras operações (sobrepondo mais "tinta")

Figure 1-1 The painter's model

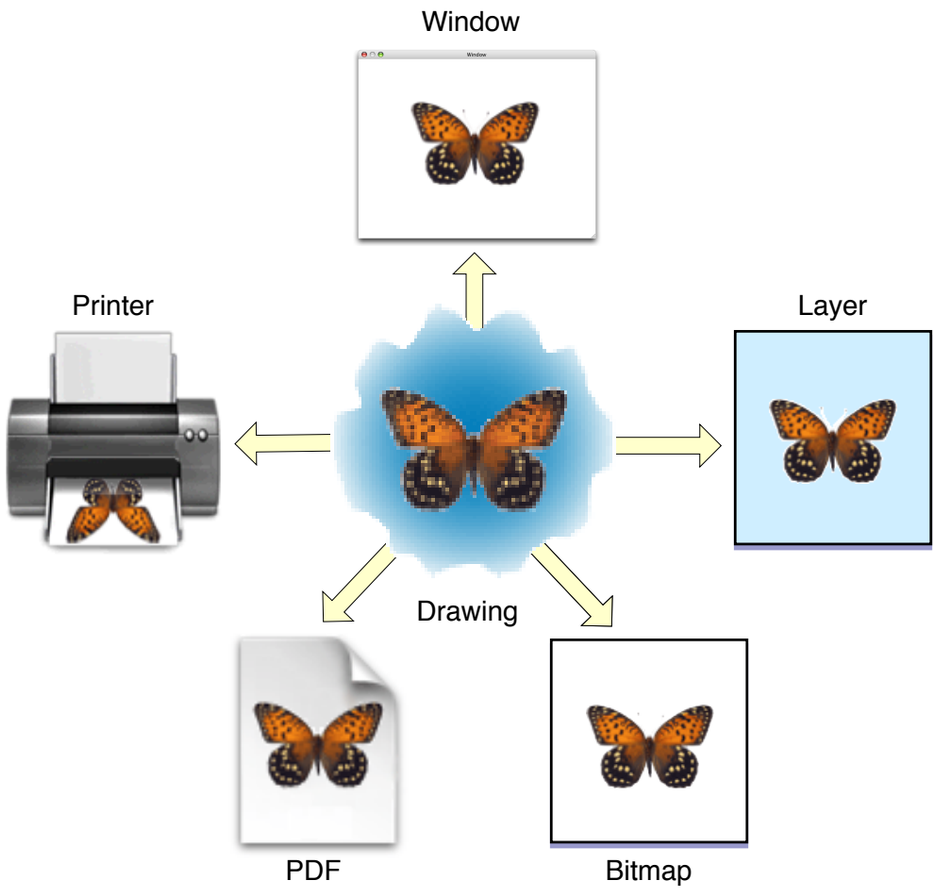


* Fonte: Drawing with Quartz 2D (Apple)

Contexto gráfico

- **CGContextRef**
  - Um tipo de dados opaco: PDF, janela, bitmap, etc.
  - Bitmap: RGB, CMYK, cinza
  - PDF: múltiplas páginas, independe de resolução
  - Janela: desenha em um UIView
  - Camada (**CGLayerRef**): performance e + facilidades em apps gráficas

Figure 1-2 Quartz drawing destinations



* Fonte: Drawing with Quartz 2D (Apple)

Tipos de dados opacos

- CGPathRef
- CGImageRef
- CGLayerRef
- CGPatternRef
- CGShadingRef e CGGradientRef
- CGFunctionRef
- CGColorRef e CGColorSpaceRef
- CGImageSourceRef e CGImageDestinationRef
- CGFontRef
- CGPDF*Ref

Objetos que podem ser manipulados e exibidos no contexto

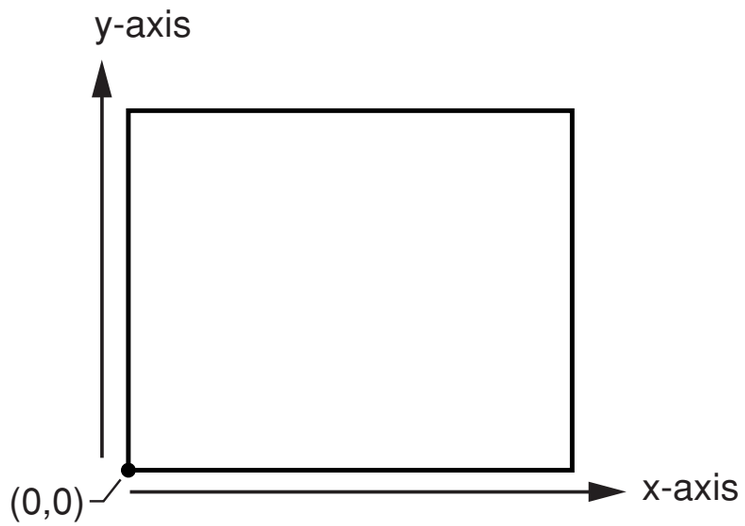
Estados gráficos

- Contexto acumula estados
  - CTM (transformadas)
  - Área de clipping
  - Atributos de linha (espessura, etc.)
  - Cor atual e valor alfa
  - Fonte
  - Filtros (blend mode)
- Pode-se guardar e recuperar o estado do contexto
  - CGContextSaveGState e CGContextRestoreGState

Sistemas de coordenadas

- User space (lógico)
- Device space (físico)
  - iPhone 3 320x480
  - iPhone 4 640x960
- **CTM** (Current Transformation Matrix) mapeia entre um e outro
  - **Affine transform**
  - Suporta redimensionamento, rotação, translação
- User space em **iOS** (em um UIView) é **invertido**

Figure 1-4 The Quartz coordinate system



Recursos gráficos do UIKit

- Objetos (afetam contexto dentro de drawRect:)
  - UIColor

```
UIColor *aColor = [UIColor redColor];
[UIColor colorWithRed:green:blue:alpha:]
```

  - Definindo cor do contexto no UIView

```
[aColor setFill];  [aColor setFillStroke];
```

  - UIImage

```
[UIImage imageNamed:]
[UIImage imageWithData:]
```

  - Texto

```
[NSString drawAtPoint:withFont:]
[NSString drawInRect:]
[UIFont fontWithName:size:]
```

- Estruturas e valores comuns

```
CGFloat x, y, height, width;
CGPoint p = CGPointMake(x, y);
CGRect r = CGRectMake(x, y, height, width);
```

Conversão de coordenadas

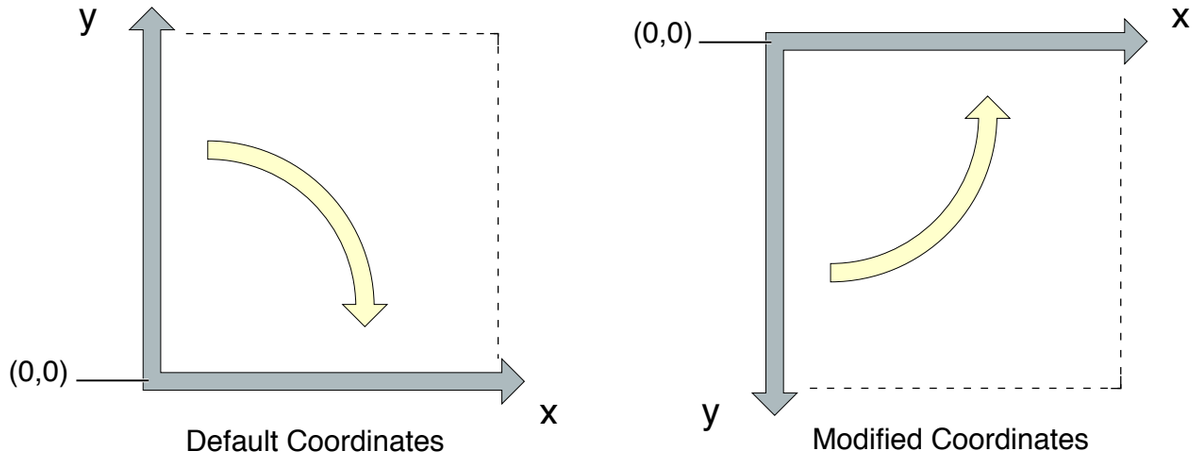
- UIImage que contém um **CGImage** já faz isto automaticamente

```
CGImageRef cgImg = [uiImg CGImage]; // uiImg.CGImage
UIImage  * uiImg = [UIImage imageWithCGImage: cgImg];
```

- Para criar novas imagens, trabalhar no contexto CG e depois gerar uma nova **UIImage**, é necessário mudar o sistema de coordenadas

```
CGContextTranslateCTM(ctx, 0, uiImage.size.height);
CGContextScaleCTM(ctx, 1.0, -1.0);
```

Figure 1-5 Modifying the coordinate system creates a mirrored image.



* Fonte: Drawing with Quartz 2D (Apple)

Gerência de memória em CF: posse dos objetos

- Quartz usa o modelo do Core Foundation (reference counting)
  - **CFRelease(objeto)** e **CFRetain(objeto)** para controlar o RC
  - Se objeto for obtido de uma função que possui as palavras **Create** ou **Copy**, a posse do objeto é **transferida** e não será mais responsabilidade do CF liberá-lo (será preciso chamar CFRelease() para liberá-lo)
  - Se objeto não for obtido de uma dessas funções, ele **não deve** ser liberado, já que a posse é retida pelo CF que tem a responsabilidade de liberar a memória.
  - Se você não possui um objeto e precisa mantê-lo, use **CFRetain** para obter a posse do objeto e depois **CFRelease** quando não precisar mais
  - Objetos CG têm métodos **CG<Objeto>Retain** e **CG<Objeto>Release** que são melhores* que **CFRetain(objeto)** e **CFRelease(objeto)**

* não falha se for NULL e é mais legível

Conversão UIKit - CF/CG

- Alguns objetos permitem cast direto (se **não** estiver usando ARC)

```
CFStringRef cfString = (CFStringRef) nsString;
NSString * nsString  = (NSString *) cfString;
```

- Se UIKit estiver usando **ARC** (Automatic Reference Counting) **não é permitido** fazer casts diretos (toll-free bridging) entre Core Foundation e Foundation/UIKit!
  - É preciso qualificar a transferência (ou não) informando a posse do objeto (para CF ou ARC) usando **__bridge**, **__bridge_transfer** ou **__bridge_retain**

```
CFStringRef cfString = (_bridge CFStringRef) nsString;
NSString * nsString  = (_bridge NSString *) cfString;
```

- **UIImage <=> CGImageRef** usa uma operação de conversão:

```
CGImageRef cgImg = [UIImage CGImage];
UIImage * uiImg  = [UIImage imageWithCGImage: cgImg];
```

Contexto gráfico

- **CGContextRef**
- Para obter um contexto gráfico no UIView
  - Implemente o método drawRect: que automaticamente cria um contexto gráfico (e o configura para as coordenadas do UIView). Para obtê-lo:

```
CGContextRef ctx = UIGraphicsGetCurrentContext();
```

- Outros contextos gráficos
  - PDF: **CGPDFContextCreateWithURL**(...) e **CGPDFContextCreate**(...)
  - Bitmap: **UIGraphicsBeginImageContextWithOptions**(...) - ideal para desenhar offscreen não é usar este objeto, mas **CGLayers**

Exemplo: desenho no contexto

```
#import "GraphicsViewControllerView.h"

@implementation GraphicsViewControllerView

- (id)initWithFrame:(CGRect)frame {
    self = [super initWithFrame:frame];
    if (self) {
        // Initialization code
    }
    return self;
}


- (void)drawRect:(CGRect)rect {

    CGContextRef ctx = UIGraphicsGetCurrentContext();

    CGContextSetRGBFillColor (ctx, 1, 0, 0, 1); //3
    CGContextFillRect (ctx, CGRectMake (0, 0, 200, 100 )); //4
    CGContextSetRGBFillColor (ctx, 0, 0, 1, .5); //5
    CGContextFillRect (ctx, CGRectMake (0, 0, 100, 200)); //6
}

@end
```

2. Caminhos

- Uma ou mais figuras vetoriais ou subcaminhos feitas de linhas e/ou curvas
  - Pode ser aberto ou fechado
- Pode ser uma figura, linha, desenho
- Tem um preenchimento (fill) e um traço (stroke)

Figure 3-2 A path that contains two shapes, or subpaths

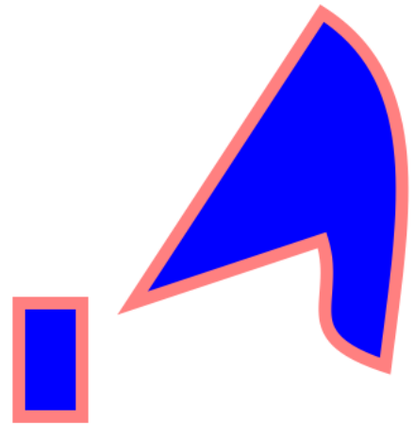


Figure 3-3 A clipping area constrains drawing

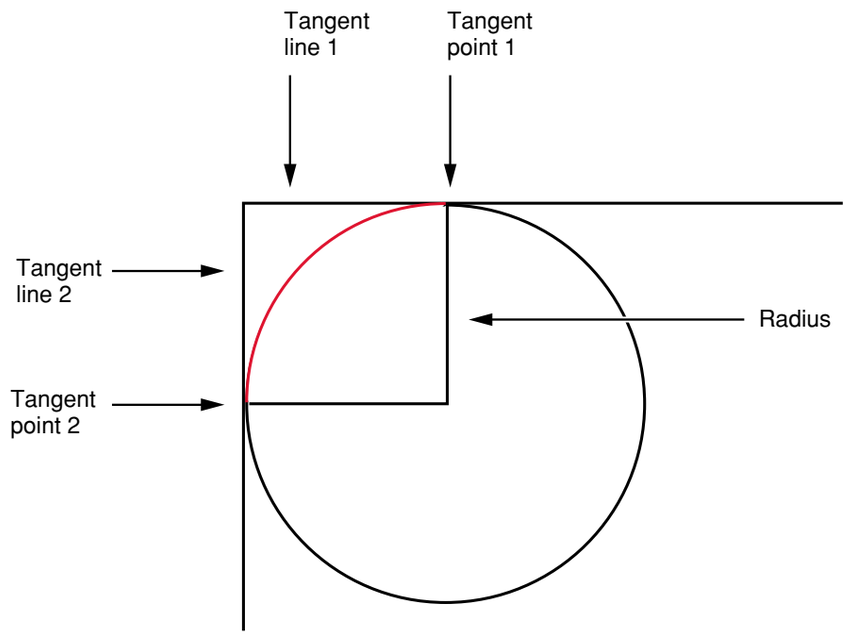
Componentes: pontos e linhas

- Pontos
  - CGPoint ou par de CGFloat
  - **CGContextMoveToPoint**(ctx, x0, y0)
    - Move para o ponto atual: current point
- Linhas
  - **CGContextAddLineToPoint**(ctx, x1, y1)
    - Desenha uma linha a partir do current point, até um ponto final, que se torna o current point
    - Chamadas sucessivas, criam um desenho formado por várias linhas conectadas

Componentes: arcos

- Arcos
  - **CGContextAddArc**(ctx, cy, cx, r, angi, angf, dir)
    adiciona um arco em uma posição cx, cy
    - cy, cx - centro
    - r - raio
    - angi, angf - ângulo inicial e final (0 a 2*M_PI faz um círculo)
    - dir - 1: sentido horário, 0: sentido anti-horário
  - **CGContextAddArcToPoint**(ctx, x1, y1, x2, y2, r)
    conecta um arco tangencialmente a duas linhas
    - x1, y1, e x2, y2 - pontos tangenciais
    - r - raio

Figure 3-5 Defining an arc with two tangent lines and a radius



Componentes: curvas

- Bezier quadrática (um ponto de controle) e cúbica (dois pontos - default); ponto inicial é o current point
- Bézier cúbica
  - **CGContextAddCurveToPoint** (ctx, cp1x, cp1y, cp2x, cp2y, x, y)
- Bézier quadrática
  - **CGContextAddQuadCurveToPoint**(ctx, cpx, cpy, x, y)

Figure 3-7 A cubic Bézier curve uses two control points

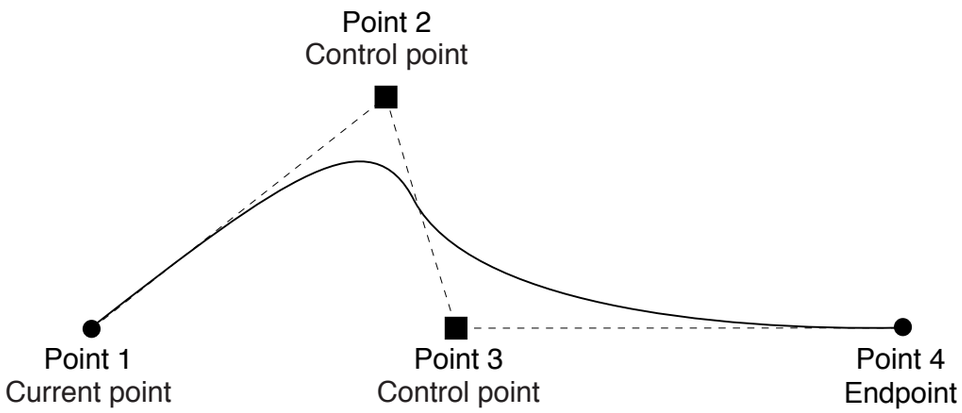
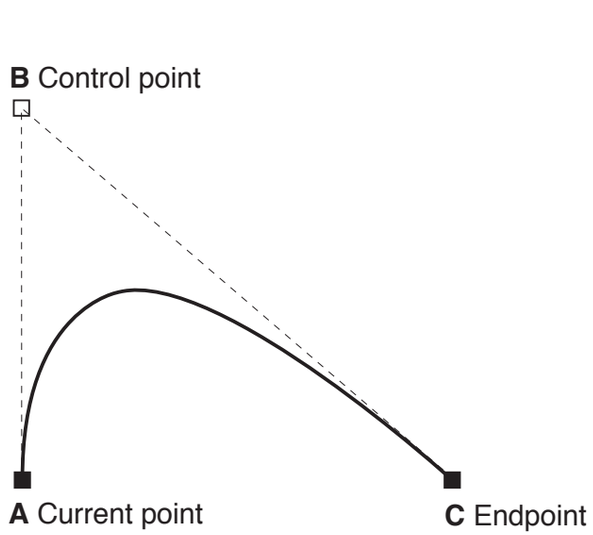
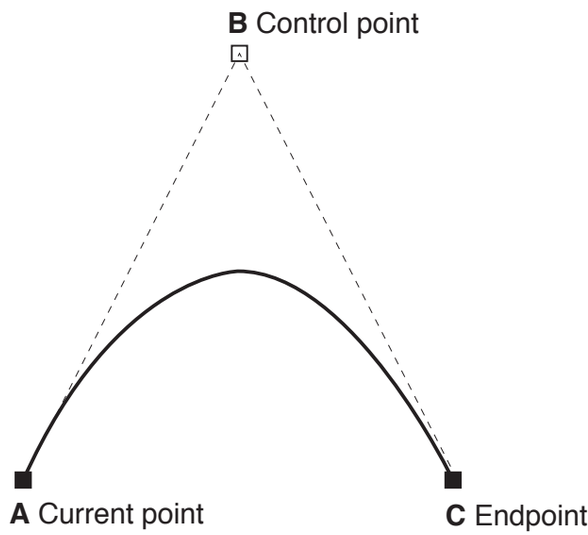


Figure 3-8 A quadratic Bézier curve uses one control point





Fechamento

- Um caminho pode ser fechado chamando-se **CGContextClosePath**(ctx)
  - Desenha uma linha fechando o caminho
  - Qualquer nova chamada a operações de movimento ou desenho de pontos, criará um novo caminho

Elipses e retângulos

- CGContextAddRect
- CGContextAddRects
- CGContextAddEllipseInRect

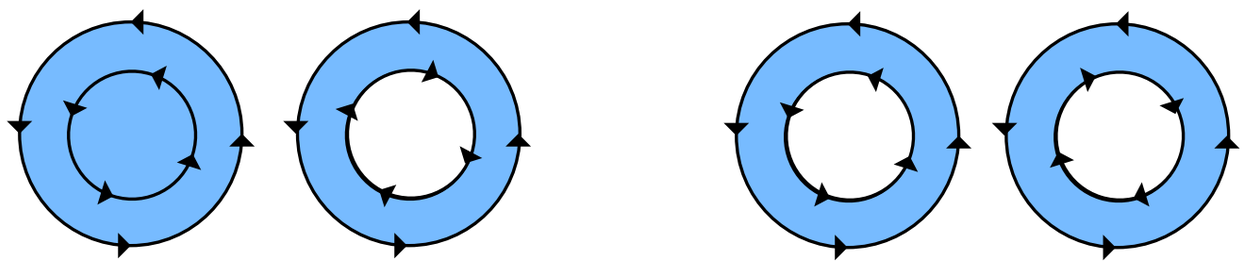
Reuso de caminhos

- Caminhos podem ser criados e reusados
- **CGContextBeginPath(ctx)**
  - Marca o início de um caminho no **contexto** atual
  - Termina com CGContextClosePath(ctx)
- **CGMutablePathRef path = CGPathCreateMutable()**
  - Referência para o caminho; métodos operam sobre o **caminho** e não sobre contexto:
    - CGPathMoveToPoint, CGPathAddLineToPoint, CGPathAddArc, etc.
  - Para fechar, **CGPathCloseSubpath(path)**
  - **CGContextAddPath** adiciona caminho no contexto
  - É preciso liberar memória **CGPathRelease(path)**

Pintura de caminhos

- **CGContextStrokePath(ctx)** - contorna o desenho com um traço
- **CGContextFillPath(ctx)** - preenche o desenho
- **CGContextDrawPath(ctx, modo)**
  - kCGPathFill, kCGPathFillStroke, kCGPathSroke, kCGPathEOFill, etc.

Figure 3-12 Concentric circles filled using different fill rules



Atributos de linha

- CGContextSetLineWidth
- CGContextSetLineJoin
- CGContextSetLineCap
- CGContextSetMiterLimit
- CGContextSetLineDash
- ...

Clipping

- Em vez de **desenhar** um caminho (**CGContextDrawPath**, etc.) pode-se defini-lo como uma área de **recorte** (clipping) associada ao contexto: **CGContextClip**
    - Ou CGContextEOClip, etc.

```
CGContextBeginPath (c);
CGContextAddArc(c, w/2, h/2, ((w>h)?h:w)/2, 0, 2*PI, 0);
CGContextClosePath (c);
CGContextClip(c);
```

- O clipping afeta o **contexto** (que não mais usa toda a área disponível para desenhar)
    - Se já houver um clipping anterior, o novo será combinado com ele
- Para recuperar o estado anterior do contexto, deve-se guardá-lo
    - CGContextSaveGState(c)
    - CGContextRestoreGState(c)

Figure 3-3 A clipping area constrains drawing

Filtros de composição (blend modes)

- Define o que acontece quando um objeto opaco (caminho, imagem, desenho) é desenhado sobre outro
- Blend-modes se acumulam no contexto
  - Se necessário, grave o estado anterior do contexto (CGContextSaveGState) para recuperar o estado anterior do contexto posteriormente (CGContextRestoreGState)
- CGContextSetBlendMode(ctx, filtro)
  - filtro = kCGBlendMode*

Flitros de composição (blend modes) kCGBlendMode*

- Normal (default)
  *resultado = (a * fg) + (1 - a) * bg*
- Multiply
  kCGBlendModeMultiply
- Screen
  kCGBlendModeScreen
- Overlay
  kCGBlendModeOverlay
- Darken
  kCGBlendModeDarken
- Lighten
  kCGBlendModeLighten
- Color Dodge
  kCGBlendModeColorDodge
- Color Burn
  kCGBlendModeColorBurn
- Soft Light
  kCGBlendModeSoftLight
- Hard Light
  kCGBlendModeHardLight
- Difference
  kCGBlendModeDifference
- Exclusion
  kCGBlendModeExclusion
- Hue
  kCGBlendModeHue
- Saturation
  kCGBlendModeSaturation
- Color
  kCGBlendModeColor
- Luminosity
  kCGBlendModeLuminosity

3. Cores e color spaces

- Cores são representadas por um conjunto de valores que têm significado em determinado color space

Espaço	Componentes	Valores
RGBA	red, green, blue, alpha	CGFloat
BGR	blue, green, red	CGFloat
CMYK	cyan, magenta, yellow, black	CGFloat
HSB	Matiz, saturação, brilho	ângulo, %, %

Cores e color spaces

- iOS suporta apenas color spaces dependentes do hardware
- Para criar
  - CGColorSpaceCreateDeviceGray
  - CGColorSpaceCreateDeviceRGB
  - CGColorSpaceCreateDeviceCMYK
- Para liberar
  - CGColorSpaceRelease(colorSpace)
- Em iOS, UIColor usa RGB e Gray, e pode ser construído ou decomposto em componentes RGB, monocromáticos ou HSB
  - Para a maior parte das aplicações práticas, usar UIColor é suficiente

UIColor e CGColor

- Obtendo um UIColor partir de componentes do color space
  - [UIColor colorWithRed:green:blue:alpha:]
  - [UIColor colorWithCGColor]
  - [UIColor colorWithHue:saturation:brightness:alpha:]
- Usando valores pré-definidos
  - [UIColor redColor]
  - [UIColor blueColor]
  - [UIColor orangeColor]
  - ...
- Operações no contexto
  - [UIColor set], [UIColor setFill], [UIColor setStroke]
- CGColor: obtenção dos componentes: 0- red, 1- green, 2- blue, 3- alpha
  - const CGFloat * array = CGColorGetComponents(cgColor)

4. Transformadas

- Pode-se operar na matriz de transformações (CTM) para realizar
  - Redimensionamento
  - Rotação
  - Translação
- As operações são feitas em um sistema de coordenadas lógico (que é mapeado ao sistema físico do dispositivo usado)
- As operações são cumulativas, ordenadas e afetam o estado do contexto
- Transformadas também podem ser realizadas em uma matriz que pode depois ser aplicada à CTM ou outro componente (affine transforms)

Figure 5-7 An image that is translated, scaled, and then rotated

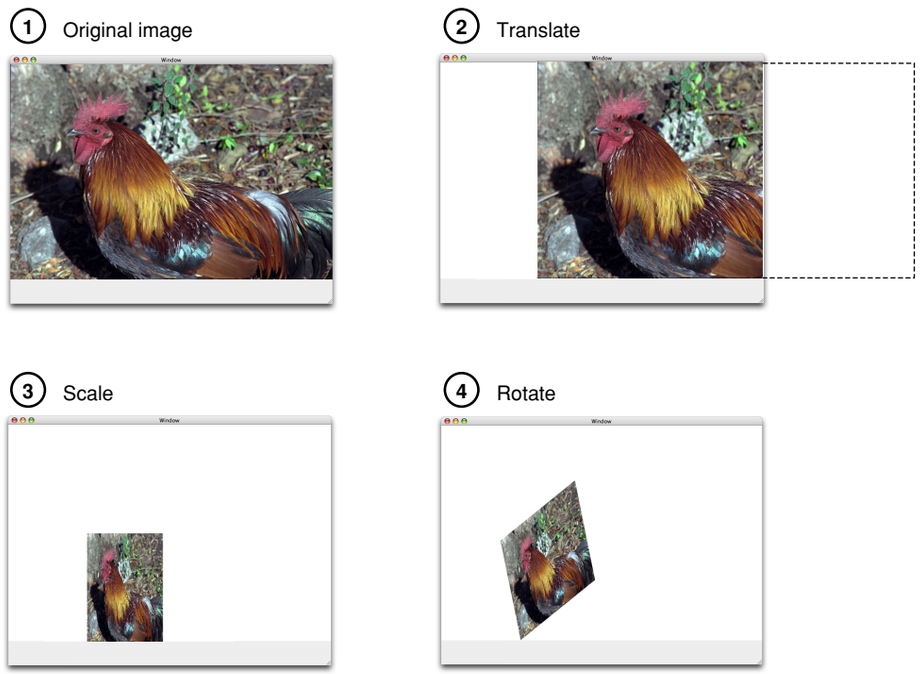


Figure 5-8 An image that is rotated, scaled, and then translated

Transformações CTM e AffineTransform

- Translação
  - CGContextTranslateCTM(c, dx, dy);
  - CGAffineTransform **transform** =
    CGAffineTransformMakeTranslation(dx, dy);
    CGPathAddRect(path, **&transform**, rect);
- Rotação
  - CGContextRotateCTM(c, radianos);
  - CGAffineTransformMakeRotation...
- Redimensionamento
  - CGContextScaleCTM(c, px, py);
  - CGAffineTransformMakeScale...

5. Padrões, sombras e gradientes

- **Padrão**: célula contendo um desenho que pode ser usada como tinta para pintar
- **Sombra**: uma imagem pintada por baixo de um objeto gráfico, e frequentemente deslocada e borrada
- **Gradiente**: uma função de pintura que varia de uma cor para outra; pode ser radial ou linear

Figure 6-2 A pattern cell

Figure 6-1 A pattern drawn to a window

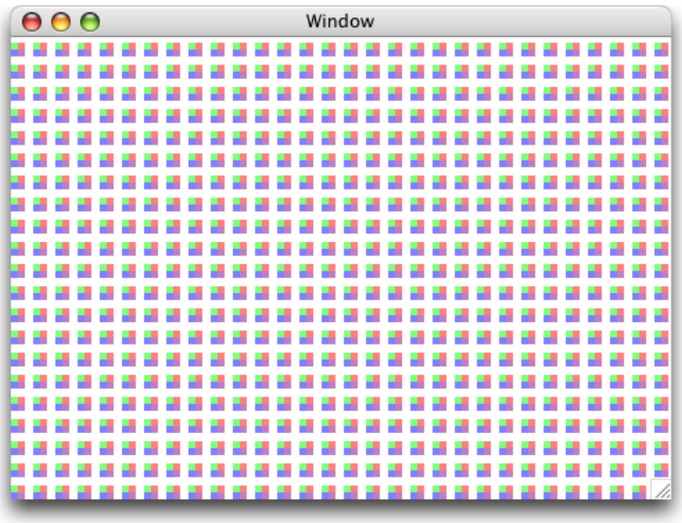


Figure 7-1 A shadow

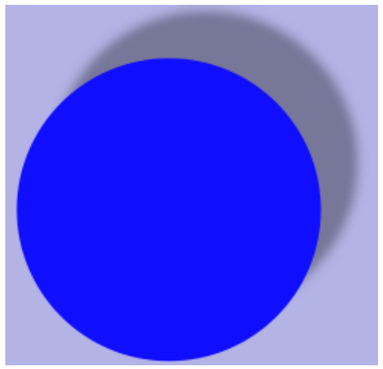
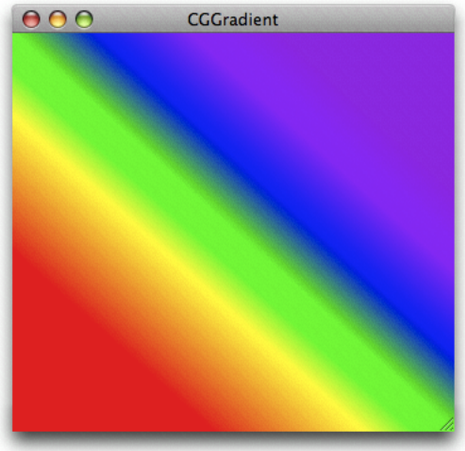


Figure 8-2 An axial gradient created with seven locations and colors



Padrões: como usar

1. Escrever
   a. uma **função de callback** que desenha o padrão, e
   b. um **ponteiro para uma estrutura** CGPatternCallbacks
2. Definir o colorspace como sendo um pattern (**CGColorSpaceCreatePattern**) e anatomia (propriedades do padrão) em um objeto **CGPattern**
3. Especificar o padrão como uma tinta (fill ou stroke)
4. Desenhar com o padrão

Callback e struct

```
#define PSIZE 16

static void drawStencilStar (void *info, CGContextRef c) {
  int k;
  double r, theta;
  r = 0.8 * PSIZE / 2;
  theta = 2 * M_PI * (2.0 / 5.0); // 144 degrees

  CGContextTranslateCTM (c, PSIZE/2, PSIZE/2);
  CGContextMoveToPoint(c, 0, r);
  for (k = 1; k < 5; k++) {
   CGContextAddLineToPoint (c,
                    r * sin(k * theta),
                    r * cos(k * theta));
  }
  CGContextClosePath(c);
  CGContextFillPath(c);
}

static const CGPatternCallbacks
            callbackStruct = {0, &drawStencilStar, NULL};
```

```
struct CGPatternCallbacks {
   unsigned int
      version;
   CGPatternDrawPatternCallback
      pattern;
   CGPatternReleaseInfoCallback
      info;
}
```

Color space e anatomia

- Definir o color space do preenchimento (ou traço) como sendo um pattern.

```
CGColorSpaceRef patternSpace =
                        CGColorSpaceCreatePattern(NULL);

CGContextSetFillColorSpace(context, patternSpace);

// Ou, se for usar o padrão para desenhar o contorno
// CGContextSetStrokeColorSpace(context, patternSpace);

CGColorSpaceRelease(patternSpace);
```

- Criar um objeto CGPattern especificando as propriedades do padrão e passando o struct

```
CGPatternRef pattern = CGPatternCreate(nil,
                        CGRectMake(0, 0, 16, 16),
                        CGAffineTransformIdentity, 16, 16,
                        kCGPatternTilingNoDistortion,
                        FALSE,
                        &callbackStruct);
```

Usar o padrão

```
float color = {1.0, 0.0, 0.0, 1.0};

CGContextSetFillPattern (context, pattern, &color);

CGContextFillRect(context, CGRectMake(160,160,150,150));
```

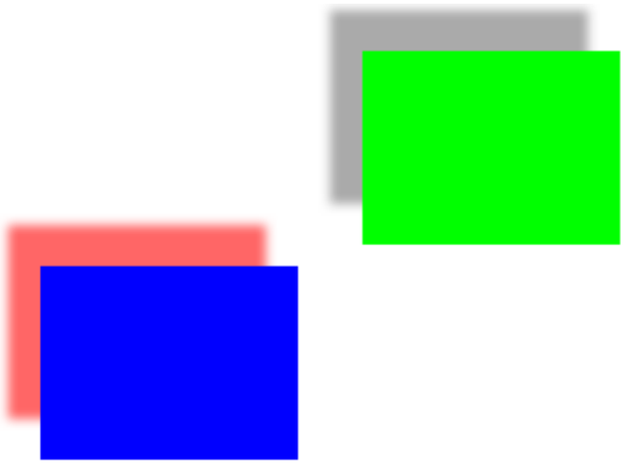
Não esqueça de

```
CGPatternRelease(pattern);
```

Sombras

- **CGContextSetShadow** afeta todo o contexto
  - Recebe: ctx, offset (CGSize) e blur (gaussian)
  - Todos os objetos ganham uma sombra (objeto desenhado com RGBA 0,0,0,1)
- Para evitar que isto ocorra, guarde o estado do contexto antes de aplicar a sombra
- Sombra pode ter cor **CGContextSetShadowWithColor**

Figure 7-3 A colored shadow and a gray shadow

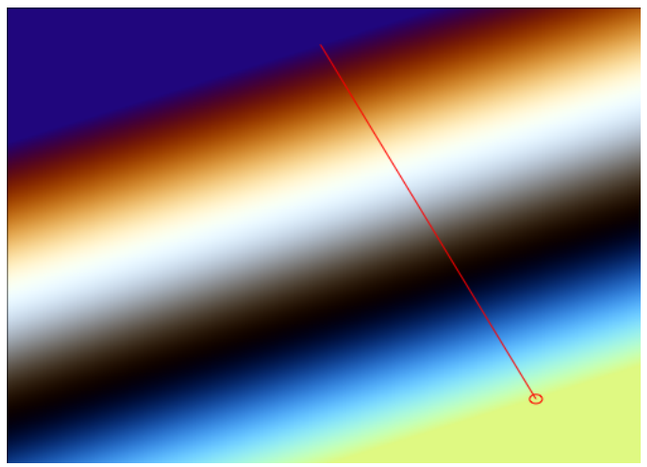


```
CGSize offset = CGSizeMake(5.0, 5.0);
CGFloat blur = 0.5;

CGContextSetShadow(context, offset, blur);

...
```

Gradientes

- Há dois objetos para construir gradientes
  - **CGShadingRef** e **CGGradientRef**
  - O mais simples é CGGradientRef
- Há dois tipos de gradiente
  - **Radial** (varia entre dois círculos ou ponto-círculo)
  - **Linear** (varia ao longo de uma linha)
- Gradientes podem ter várias cores ao longo do eixo, variar cores e alfa, e estender cores além dos limites

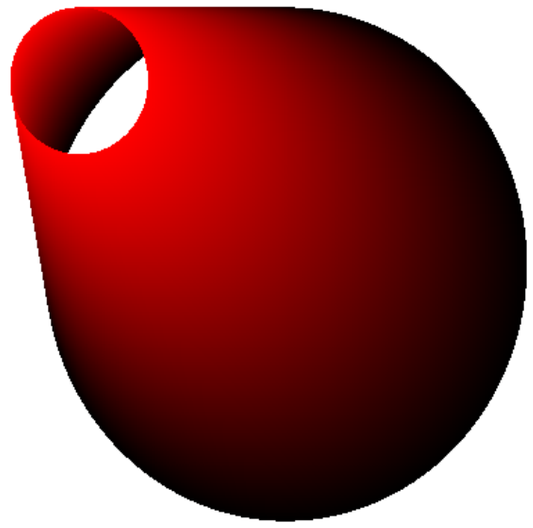
Figure 8-7 Extending an axial gradient



Como usar um CGGradient

1. Crie um **CGGradientRef** (CGGradientCreateWithColorComponents) com
   - color space
   - array de dois ou mais componentes de cor (quatro elementos cada)
   - array de duas ou mais localizações (ponto onde a cor é aplicada)
   - número de ítens em cada array
2. Pinte o gradiente com
   - **CGContextDrawLinearGradient**, ou **CGContextDrawRadialGradient**
   - Contexto, opções, início e fim do gradiente
3. Libere o objeto quando terminar

Figure 8-3 A radial gradient that varies between two circles



Obtenção dos componentes (rgba) de cada cor

```
UIColor *startColor  = [UIColor brownColor];
UIColor *middleColor = [UIColor redColor];
UIColor *endColor    = [UIColor whiteColor];

 // ponteiro para componentes (array de CGFloat)
 CGFloat *inicio  =
(CGFloat *)CGColorGetComponents([startColor CGColor]);

 CGFloat *meio =
(CGFloat *)CGColorGetComponents([middleColor CGColor]);

 CGFloat *fim    =
(CGFloat *)CGColorGetComponents([endColor CGColor]);
```

Criação do gradiente

```
// 1) componentes de cor
CGFloat colorComponents[12] = {
    inicio[0], inicio[1], inicio[2], inicio[3],
    meio[0],meio[1], meio[2], meio[3],
    fim[0], fim[1], fim[2], fim[3]
};

// 2) indices de localização
CGFloat colorIndices[3] = { 0.0, 0.2, 1.0 };

// 3) color space
CGColorSpaceRef colorSpace = CGColorSpaceCreateDeviceRGB();

// 3) gradiente
CGGradientRef gradiente =
    CGGradientCreateWithColorComponents(
       colorSpace,
       (const CGFloat *)&colorComponents,
       (const CGFloat *)&colorIndices,
       3);

CGColorSpaceRelease(colorSpace);
```

Pintura linear

```
CGGradientRef gradiente = ...

CGContextRef ctx = UIGraphicsGetCurrentContext();

CGRect bounds = [[UIScreen mainScreen] bounds];

CGPoint startPoint, endPoint;
startPoint =
   CGPointMake(bounds.size.width/2.0, bounds.size.height);
endPoint   =
   CGPointMake(bounds.size.width/2.0, 0.0);

CGContextDrawLinearGradient(ctx, gradiente, startPoint, endPoint, 0);

CGGradientRelease(gradiente);
```

6. Camadas de transparência

- Agrupamento: dois ou mais objetos combinados formando um único objeto
- Crie com marcadores:

1. CGContextBeginTransparencyLayer(ctx, NULL)
2. Desenhe os objetos que devem fazer parte do grupo
3. CGContextEndTransparencyLayer (ctx)

Figure 9-1 Three circles as a composite in a transparency layer

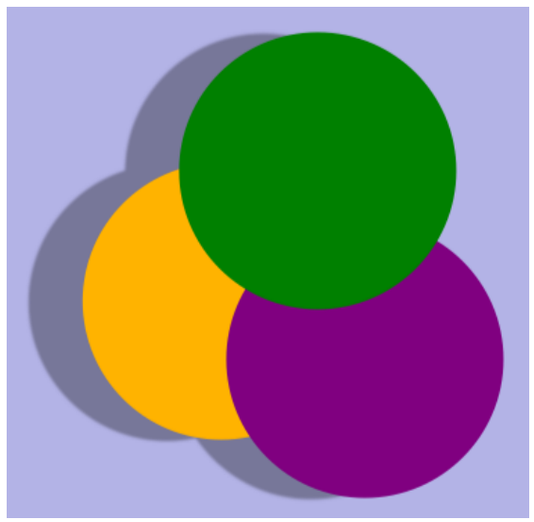
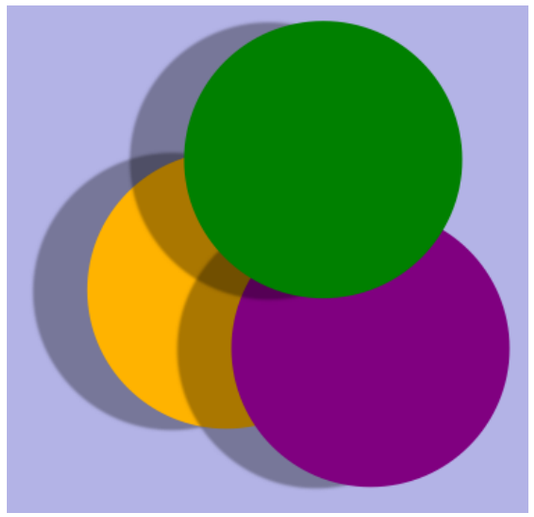


Figure 9-2 Three circles painted as separate entities



7. Imagens e máscaras

- **CGImageRef**
  - Representa imagens (bitmap) ou máscaras de imagens
    - **CGImageCreate**: permite especificar todos os detalhes da criação de um bitmap
- Máscaras
  - Um bitmap que especifica uma área a ser pintada, mas não sua cor

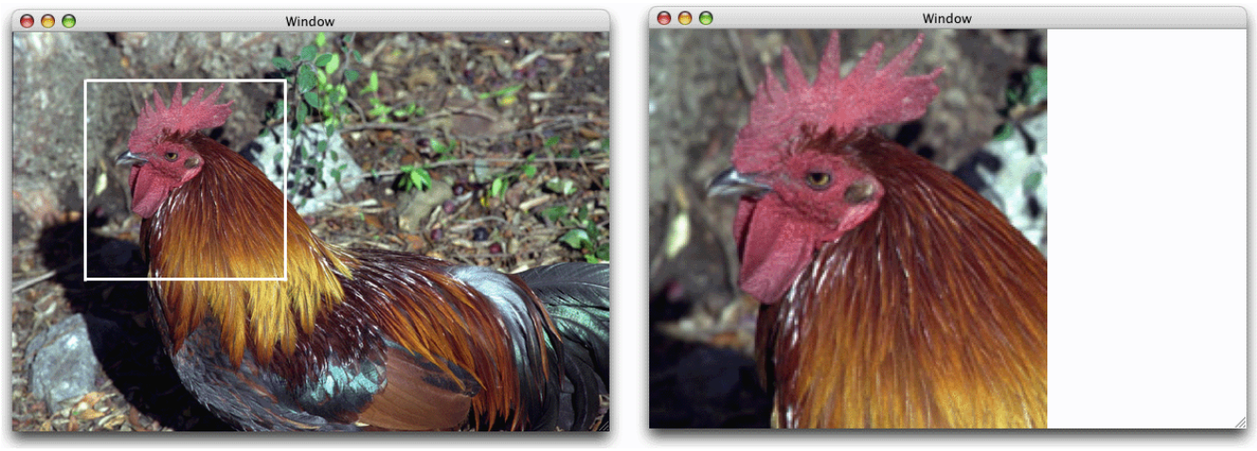
Criação de imagens

- Como **obter** uma imagem?
  - Via UIImage: URL, localmente (bundle), diretório do sistema de arquivos, NSData. Ex: `UIImage *image = [UIImage imageNamed:@"invader.png"];`
- Como **criar** uma imagem
  - **CGImageCreate**
    - Precisa especificar toda a informação sobre a imagem (bitmap info, etc.)
  - **CGImageSourceCreateImageAtIndex**
    - Cria a partir de uma fonte
  - **CGBitmapContextCreateImage**
    - Cria imagem a partir de um contexto
  - **CGImageCreateWithImageInRect**
    - Cria imagem recortando de outra imagem

Criando imagem a partir de outra

```
myImageArea = CGRectMake (rooster_head_x_origin,
                          rooster_head_y_origin,
                          myWidth,
                          myHeight);
mySubimage = CGImageCreateWithImageInRect (myRoosterImage, myImageArea);

myRect = CGRectMake(0, 0, myWidth*2, myHeight*2);

CGContextDrawImage(context, myRect, mySubimage);
```

Figure 11-4 An image, a subimage taken from it and drawn so it's enlarged



Aplicação de máscaras

- Permite controlar que parte da imagem é pintada
- **CGImageCreateWithMask**
  - Cria uma imagem resultante da aplicação de uma máscara de imagem

Figure 11-5 The original image

Figure 11-6 An image mask

Figure 11-7 The image that results from applying the image mask to the original image

Máscaras

- Máscara de cor: **CGImageCreateWithMaskingColors**
  - Uma imagem
  - Um array de componentes de cor para mascarar na imagem

Figure 11-9 Chroma key masking

- Máscara recortando o contexto: **CGContextClipToMask**
  - O contexto gráfico que será recortado
  - Retângulo onde a máscara será aplicada
  - Uma máscara de imagem

Figure 11-14 A masking image

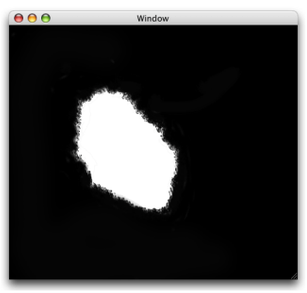


Figure 11-15 An image drawn to a context after clipping the content with an image mask

Filtros em imagens

- Desenhe o fundo
- Defina um blend mode para composição da imagem com o fundo
- Desenhe a imagem

```
CGContextSetBlendMode (myContext, kCGBlendModeDarken);
CGContextDrawImage (myContext, myRect, myImage2);
```

Figure 11-17 Background drawing (left) and foreground image (right)

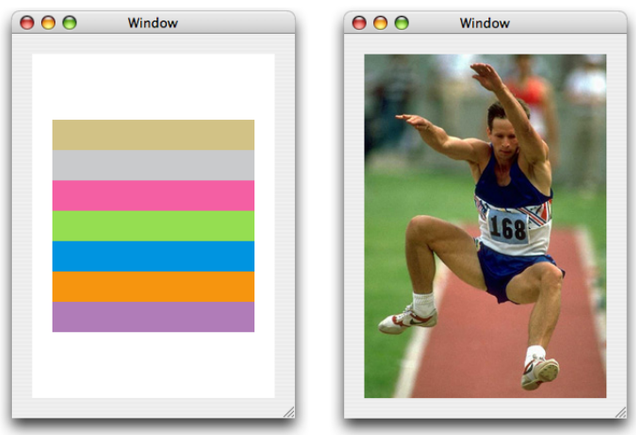


Figure 11-20 Drawing an image over a background using screen blend mode

8. CGLayer

- Permite desenhar offscreen: sistema faz cache de CGLayers; performance!
- Ideal para reusar desenhos (repetir desenhos, animação, etc.)
- CGLayerRef layer =
  CGLayerCreateWithContext(ctx, CGSize, NULL)
  - Cria um layer que herda todas as características atuais do contexto
- CGContextRef ctx = **CGLayerGetContext**(layer)
  - Retorna contexto gráfico associado com o layer
- **CGContextDrawLayerAtPoint**(...) e **CGContextDrawLayerInRect**(...)
  - Desenham o layer em um contexto gráfico

Figure 12-1 Repeatedly painting the same butterfly image

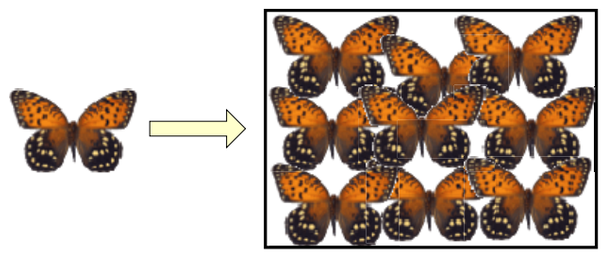
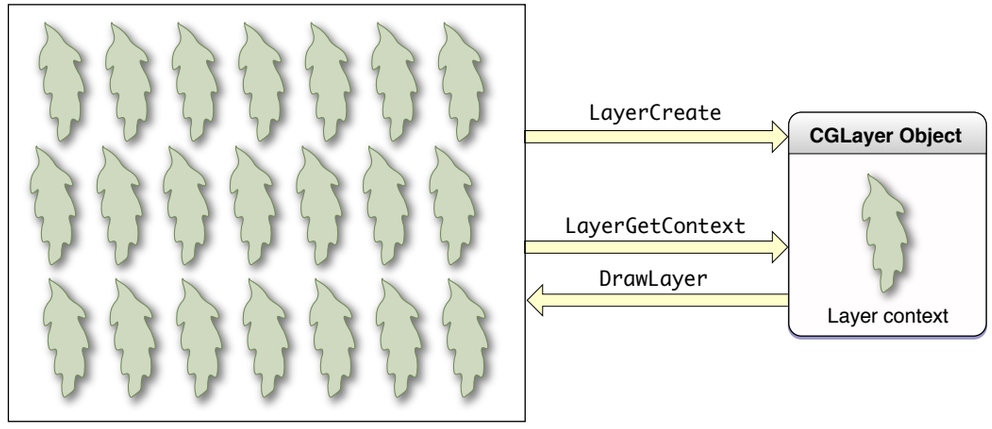


Figure 12-2 Layer drawing



9. Texto

- Esta seção será apresentada apenas como demonstração no XCode
  - Não haverá exercícios usando texto
- Como referência, use o documento "Drawing with Quartz 2D", capítulo 17.

Fontes de referência

- Apple. *Drawing with Quartz 2D Programming Guide*. 11/12/2007
- Apple. *Color Management Overview*. 07/07/2005
- Apple. *Core Image Programming Guide*. 12/10.2011