



Desenvolvimento para iPhone

Usando iOS 8 SDK

Agenda

- Sistema de Coordenadas (revisão rápida);
- UIBezierPath e Views customizadas;
- Um pouco de CoreGraphics;
- UIColor;
- NSAttributedString;
- UIImage;

Objetivos do Dia

- Aprender a desenhar coisas usando UIBezierPath;
- Entender o funcionamento do UIColor;
- Saber usar o UIImage;
- Conhecer o NSAttributedString

Sistema de Coordenadas (revisão)

- Coordenadas da View pai;
- Coordenadas da View filha;
 - Coordenadas internas;
- Relação entre sistemas de coordenadas.



UIBezierPath (View)

- View (qualquer coisa que herde de **UIView**) representa um espaço retangular na tela;
 - Define um sistema de coordenadas;
 - Desenha e é capaz de responder a eventos dentro do próprio espaço;

UIBezierPath (View)

- Views são hierárquicas!
 - Uma view tem apenas uma superview;
 - Uma view tem zero ou muitas subviews (NSArray *);
 - A ordem das subviews importa!
 - As últimas posições no array estão por cima das primeiras posições
 - Uma view pode ou não “cropar” suas subviews para caber dentro de si.

UIBezierPath (View)



UIBezierPath (View)

- Essa hierarquia de *views* é, normalmente, construída graficamente usando o Xcode;
- Mas, também pode ser feita via código usando os métodos:
 - - (**void**)**addSubview**:(**UIView ***)aView;
 - Para adicionar **aView** em si;
 - - (**void**)**removeFromSuperview**;
 - Para se remover da **superview**;

UIWindow

- UIWindow é a view que está no topo da hierarquia de views;
- Em uma aplicação para iOS, em 99% dos casos, existe apenas uma UIWindow;
- Então, daqui pra frente, vamos falar apenas em UIView.

UIBezierPath (UIView)

```
#import <UIKit/UIKit.h>

@interface MinhaView : UIView

@end
```

UIBezierPath (UIView)

```
#import "MinhaView.h"

@implementation MinhaView

- (instancetype)initWithFrame:(CGRect)frame {
    self = [super initWithFrame:frame];
    if (self) {
        [self configurarMinhaView];
    }
    return self;
}

- (void)awakeFromNib {
    [self configurarMinhaView];
}

- (void) configurarMinhaView {
    //Fazer alguma coisa aqui
}

@end
```

UIBezierPath (UIView)

- Quando criar uma subclasse de UIView?
 - Quando se deseja desenhar algo customizado na tela;
 - Quando se deseja escutar um evento de *touch* especial (algo que não seja feito por algum componente padrão);
 - Quando se deseja criar uma composição **complexa e reutilizável** de Views (geralmente feito usando XIB).

UIBezierPath

- Desenhar é fácil!
- Basta criar uma classe filha de UIView e sobrescrever um único método:
 - - (**void**)drawRect:(**CGRect**)rect;
- **NUNCA, JAMAIS, NUNCA MESMO** chame o método acima!
 - Em vez disso, deixe o sistema operacional saber que você precisa atualizar o desenho da sua view usando:
 - - (**void**)setNeedsDisplay;
 - - (**void**)setNeedsDisplayInRect:(**CGRect**)aRect;

UIBezierPath

- Como implementar o método:
-(**void**)drawRect:(**CGRect**)rect; ?
- Dois jeitos:
 - **Jeito McGyver**: usando o framework **CoreGraphics** diretamente (uma API em linguagem C, estruturado);
 - **Jeito Chuck Norris**: usando o **UIBezierPath** (um jeito orientado-a-objeto). É esse que vamos usar hoje.

UIBezierPath (Conceitos para Desenhar com CoreGraphics)

- Obter um contexto de desenho;
 - O iOS vai deixar um pronto para você toda vez que o método **drawRect:** for chamado;
- Criar caminhos (linhas, arcos, pontos, etc);
- Definir cores, fontes, texturas e etc;
- Traçar e/ou preencher o desenho;
- Retornar o contexto.

UIBezierPath

- Faz a mesma coisa que o CoreGraphics, porém metade do esforço;
- O contexto de desenho determina pra onde o seu desenho vai;
 - No iOS, normalmente, o desenho vai para a tela, mas pode acabar indo para fora da tela, para um PDF ou mesmo para uma impressora

UIBezierPath

- Para desenhar, portanto, o iOS vai deixar o contexto pronto para usar;
- Esse contexto só é válido durante o tempo de execução do método **drawRect**:
- Um novo contexto é criado a cada chamada deste método;
- Dito isso, **jamaiz** guarde a instância do contexto!

UIBezierPath

- Como obter esse contexto?

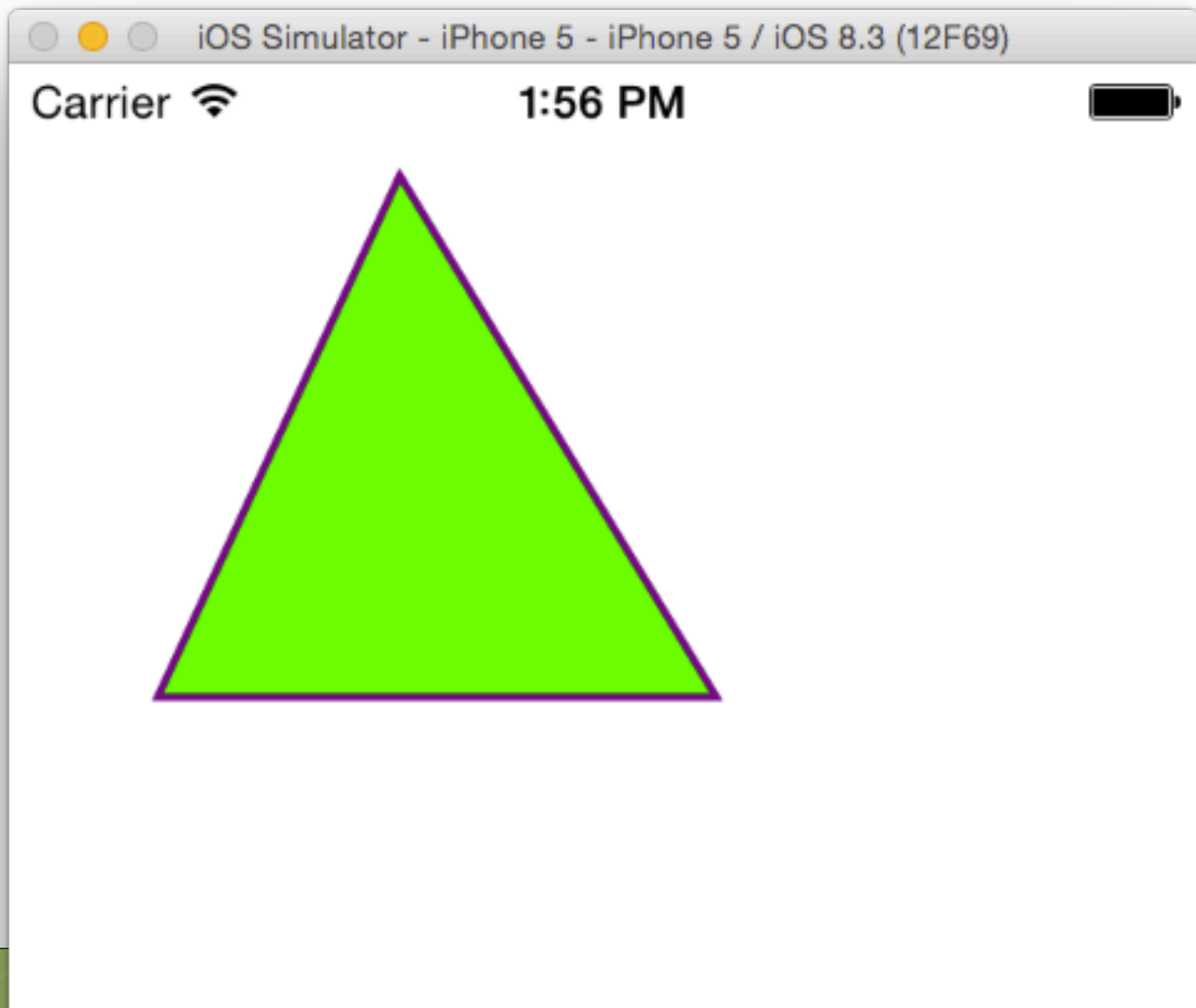
```
CGContextRef context = UIGraphicsGetCurrentContext();
```

- Mas não se preocupe com isso!
- O **UIBezierPath** desenha sempre no contexto atual;
- Então, enquanto você estiver usando ele, não precisará de obter a instância do contexto.

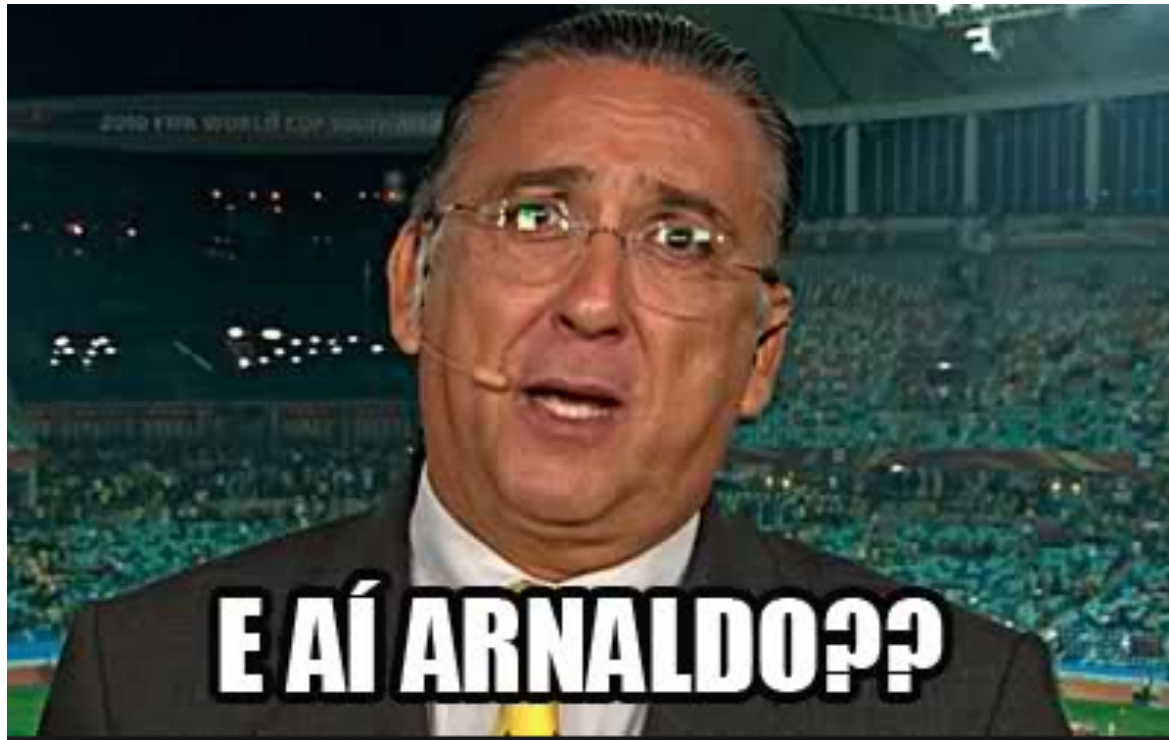
UIBezierPath



```
- (void)drawRect:(CGRect)rect {  
  
    /*Não vamos precisar disso, então, comenta!  
    CGContextRef context = UIGraphicsGetCurrentContext();  
    */  
  
    UIBezierPath *path = [[UIBezierPath alloc] init];  
  
    [path moveToPoint:CGPointMake(75, 10)];  
    [path addLineToPoint:CGPointMake(160, 150)];  
    [path addLineToPoint:CGPointMake(10, 150)];  
  
    //faz fechar o triângulo. Não é estritamente necessário.  
    //Poderíamos fechar com addLineToPoint:, mas assim é mais legal  
    [path closePath];  
  
    //As linhas abaixo controlam a cor de fundo e a cor do traçado  
    UIColor *corDeFundo = [UIColor greenColor];  
    UIColor *corDaLinha = [UIColor purpleColor];  
  
    [corDeFundo setFill]; //não é obrigatório  
    [corDaLinha setStroke]; //não é obrigatório  
  
    [path setLineWidth:2.0]; //espessura da linha em pontos (não pixels)  
  
    [path fill]; //faz aplicar a cor de fundo  
    [path stroke]; //faz aplicar a cor do traçado (cor da linha)  
}
```



UIBezierPath



UIBezierPath

- Você pode desenhar várias coisas:
- Retângulos;
- Retângulos arredondados;
- Ovais;
- Arcos angulados; e
- Adicionar um “clip”!
 - Isso faz com que um desenho seja desenhado dentro de outro.

Cuidados com UIBezierPath

- É possível desenhar com transparência, mas tenha atenção com a performance!
 - **UIColor** tem componente alpha
 - **UIView** tem a propriedade **backgroundColor**, que pode ser definida com cores transparentes (padrão é preto)
- Se for fazer desenhos transparentes, defina a propriedade **opaque** como **NO**;
- A propriedade **alpha** muda a transparência de toda a view (útil para fazer uma espécie de “disabled”)

Cuidados com UIBezierPath

- O desenho padrão é **opaque=YES**;
 - Isso é mais barato em termos de performance. Transparência custa caro.
- Você pode esconder sua *view* completamente usando a propriedade **hidden**

UIBezierPath




Views Customizadas

- Além de desenhos customizados com o **UIBezierPath**, podemos:
- Desenhar texto;
- Desenhar imagens;
- Para texto ou imagem sozinhos, não é necessário uma view customizada!
 - UILabel
 - UIImageView

NSAttributedString

- É uma *string*, onde cada caractere possui informação de estilo (cor, borda, sombra, fonte, etc.);
- É “desenhável” e sabe estimar o próprio tamanho:
 - Método **drawAtPoint:**
 - Método (**CGSize**) **size**
- Assim como a **NSString**, é imutável;
- Assim como a NSString, possui uma variante mutável:

NSMutableAttributedString



Estes dois métodos são oriundos de um mecanismo chamado **categoria**. Assunto das próximas aulas!

UIImage

- Como obter uma instância?
 - `imageNamed:@"nomeDoArquivoNoBundle.png"`
 - `initWithContentsOfFile:@"caminhoAbsoluto"`
 - `initWithData:(NSData *) byteArrayDalmagem`
- Para desenhar uma imagem:
 - **`drawAtPoint:`**
 - **`drawInRect:`**
 - **`drawAsPatternInRect:`**
- Tirar uma foto da minha View:
 - `UIGraphicsGetImageFromCurrentContext();`

Hora de Brincar!

- View customizada com UIBezierPath;
- View customizada com Texto;
- View customizada com Imagem;