Objective-C

Курс лекций и семинаров для студентов,
желающих научиться программировать под iPhone

Осень-Зима 2013

Лекция №10

Graphics & Animation: CoreGraphics, QuartzCore, CoreAnimation

Автор: Дмитрий Волков, iPhone Developer, Sibers

Sibers®

Что вы узнаете сегодня?

- Technologies: **UIKit**, **CoreGraphics**, **CoreAnimation**
- UIKit: **UIView**, **UIImage, UIBezierPath**
- UIView drawing
- CoreGraphics
- UIKit: Animations
- CoreAnimation

Technologies

Technologies

iOS предоставляет неплохой набор технологий для отображения и анимации графических элементов в вашем приложении:

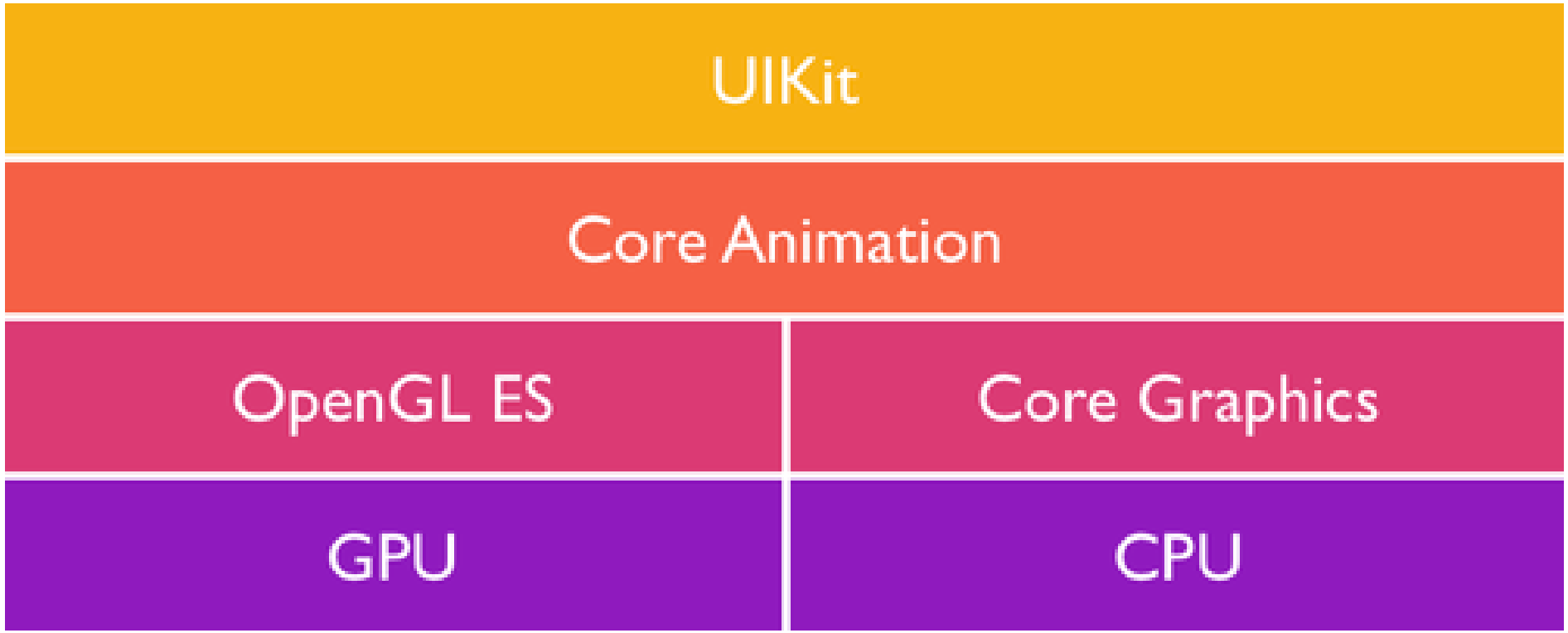
- **UIKit** — high-level Objective-C фреймворк. Предоставляет базовые возможности для рисования, вывода изображений и анимации UI.
- **CoreGraphics** — low-level C API. API для создания векторной графики, обработки и генерации изображений и PDF контента.
- **CoreAnimation** — Objective-C фреймворк. Используется для создания более сложных графических эффектов и более продвинутой анимации UI.

Technologies: How it works

В общем представлении, графическая система iOS работает так:

- Сердцем графической системы iOS является OpenGL ES:
- Рендеринг всего UI происходит на GPU.
- Любая UIView — тонкая обертка над объектом типа CALayer — абстракция OpenGL текстуры, предоставляемая фреймворком CoreAnimation.
- Так как графическая система использует OpenGL текстуры, то, для большей производительности (если это критично), следует использовать изображения для создания UI.

Technologies stack



Technologies: UIKit.

UIKit предоставляет базовый и простой в использовании набор компонентов для взаимодействия с графической системой:

- **UIView** — основной компонент, использующийся для отображения графического контента. Определяет прямоугольную область, в которой и рисуется контент.
- **UIImage** — абстракция изображения. Рекомендуемый для использования формат — PNG. Но, система поддерживает так же и другие форматы.
- **UIBezierPath** — абстракция набора векторных элементов. Вы добавляет в «путь» различные геометрические примитивы (точки, окружности, прямоугольники) которые затем можете отобразить в UI.

Technologies: UIView drawing

Базовый класс UIView определяет метод ***- (void) drawInRect:(CGRect) rect***, в котором и происходит отрисовка графического контента.

ВАЖНО: Никогда не вызывайте метод drawRect сами. Система сама вызывает данный метод в нужный момент.

В какие моменты вызывается drawRect?

- Добавление / удаление UIView в иерархию UI компонентов.
- Выставление свойства ***hidden*** в NO.
- Вызов метода ***setNeedsDisplay*** или ***setNeedsDisplayInRect:***
- Добавление / удаление другого объекта UIView, который перекроет ваш UIView.

Technologies: Graphics context

- Любые операции рисования происходят в графическом контексте (***CGContextRef***).
- Контекст — абстракция контейнера ваших графических данных. Грубо говоря – буфер байт, в который вы рисуете.
- При вызове метода drawRect: система создает для вас контекст, в котором вы и будете рисовать свой контент.
- Так же, при необходимости, вы можете создавать свои контексты и рисовать в них изображения, пути, градиенты и многое другое.

Technologies: Graphics Context

```
- (void) drawRect:(CGRect) rect
{
    CGContextRef context = UIGraphicsGetCurrentContext();
    CGContextSetFillColorWithColor(context,
    self.backgroundColor.CGColor);
    CGContextFillRect(context, rect);
}
```

Technologies: Core Graphics

CoreGraphics — фреймворк для взаимодействия с графической системой. C-based API — в нем не существует объектов, существуют «непрозрачные» типы данных.

- Функции и типы данных имеют префикс CG*.

Основные типы данных в **CoreGraphics**:

- ***CGContextRef*** — тип данных, описывающий графический контекст, в который будут применяться операции рисования.
- ***CGPathRef / CGMutablePathRef*** — «путь» в контексте. Позволяет добавлять в контекст графические фигуры, которые затем можно отрисовать.
- ***CGImageRef*** — low-level представление изображений.
- ***CGColorRef*** — low-level представление цвета.

UIKit / CoreGraphics

Многие классы из UIKit позволяют получить свое CoreGraphics представление:

UIImage:

@property(nonatomic, readonly) CGImageRef CGImage;

UIColor:

@property(nonatomic, readonly) CGColorRef CGColor;

UIBezierPath:

@property(nonatomic, readonly) CGPathRef CGPath;

UIKit / CoreGraphics

Также, UIKit предоставляет методы для создания графических контекстов:

//создание и удаление

UIGraphicsBeginImageContext(CGSize size);

UIGraphicsBeginImageContextWithOptions(CGSize size, BOOL opaque, CGFloat scale);

UIGraphicsEndImageContext();

//получение текущего контекста

UIGraphicsGetCurrentContext();

//получение изображения из текущего контекста

UIGraphicsGetImageFromCurrentImageContext();

//управление стеком контекстов

UIGraphicsPushContext(CGContextRef context);

UIGraphicsPopContext();

Sibers®

CoreGraphics: Usage

```
UIGraphicsBeginImageContext(CGSizeMake(100,100));
CGContextRef context = UIGraphicsGetCurrentContext();
UIImage* img = ...;
CGMutablePathRef arcPath = CGPathCreateMutable();
CGPathAddArc(arcPath, NULL, 50, 50, 40, 0, 2*M_PI, 1);

CGContextAddPath(context, arcPath);
CGContextSaveGState(context);
CGContextClip(context);
CGContextDrawImage(context, CGRectMake(0,0,100,100), img.CGImage);
CGContextRestoreGState(context);
```

CoreGraphics: Usage

```
CGContextAddPath(context, arcPath);
CGContextSetStrokeColorWithColor(context, [UIColor blueColor].CGColor);
CGContextStrokePath(context);
UIImage* result = UIGraphicsGetImageFromCurrentImageContext();
UIGraphicsEndImageContext();
CGPathRelease(arcPath);
```

CoreGraphics: Result

CoreGraphics

Каждый контекст хранит стек «состояний» — набор параметров, применяемых к текущему пути. К состояниям относятся:

1. Текущая трансформация контекста (CGAffineTransform)
2. Настройки заливки фигур и цвета линий
3. Настройки рисования линий
4. Clip – пути и т. д.

Для сохранения и восстановления состояния используются функции ***CGContextSaveGState(CGContext ctx)*** и ***CGContextRestoreGState(CGContext ctx)***.

CoreGraphics: Memory Management

Типы данных Core Graphics так же используют механизм подсчета ссылок для управления временем жизни объектов (как и в CoreFoundation):

- Объекты, создаваемые функциями типа ***CG*Create*** имеют счетчик ссылок = 1.
- Объекты, получаемые функциями типа ***CG*Copy**** так же имеют счетчик ссылок = 1.
- Для того, чтобы удалить созданные объекты, вызываются функции типа ***CG*Release(Ref)***
- Так же, можно посылать сообщения типа ***CG*Retain(Ref)*** для увеличения счетчика ссылок объекта.
- Объекты, получаемые функциями типа ***CG*Get***, освобождать не надо.

CoreGraphics: Memory Management

```
CGMutablePathRef mutablePath = CGPathCreateMutable();
CGPathRef copiedPath = CGPathCreateCopy(mutablePath);
CGPathRelease(mutablePath);
CGPathRelease(copiedPath);

CGColorSpaceRef colorSpace = CGColorSpaceCreateDeviceRGB();
CGColorSpaceRelease(colorSpace);

CGRect = CGContextGetClipBoundingBox(context);
```

UIKit: Animations

- Плавные анимации UI (а начиная с iOS 7, еще и физический движок, делающий их более живыми) — основная фича приятности использования iOS.
- UIKit предоставляет простые и удобные API для простых анимаций UIView объектов.
- Так же, UIKit предоставляет API для анимации переходов между различными UIView, UIViewController, keyframe анимации.

UIKit: UIView animations

```
UIView* view = [UIView new];
view.frame = CGRectMake(0,0,100,100);
[self.view addSubview: view];
[UIView animateWithDuration:0.25 animations:^{
    view.center = CGPointMake(50,50);
    view.backgroundColor = [UIColor redColor];
    view.alpha = 0;
}];
```

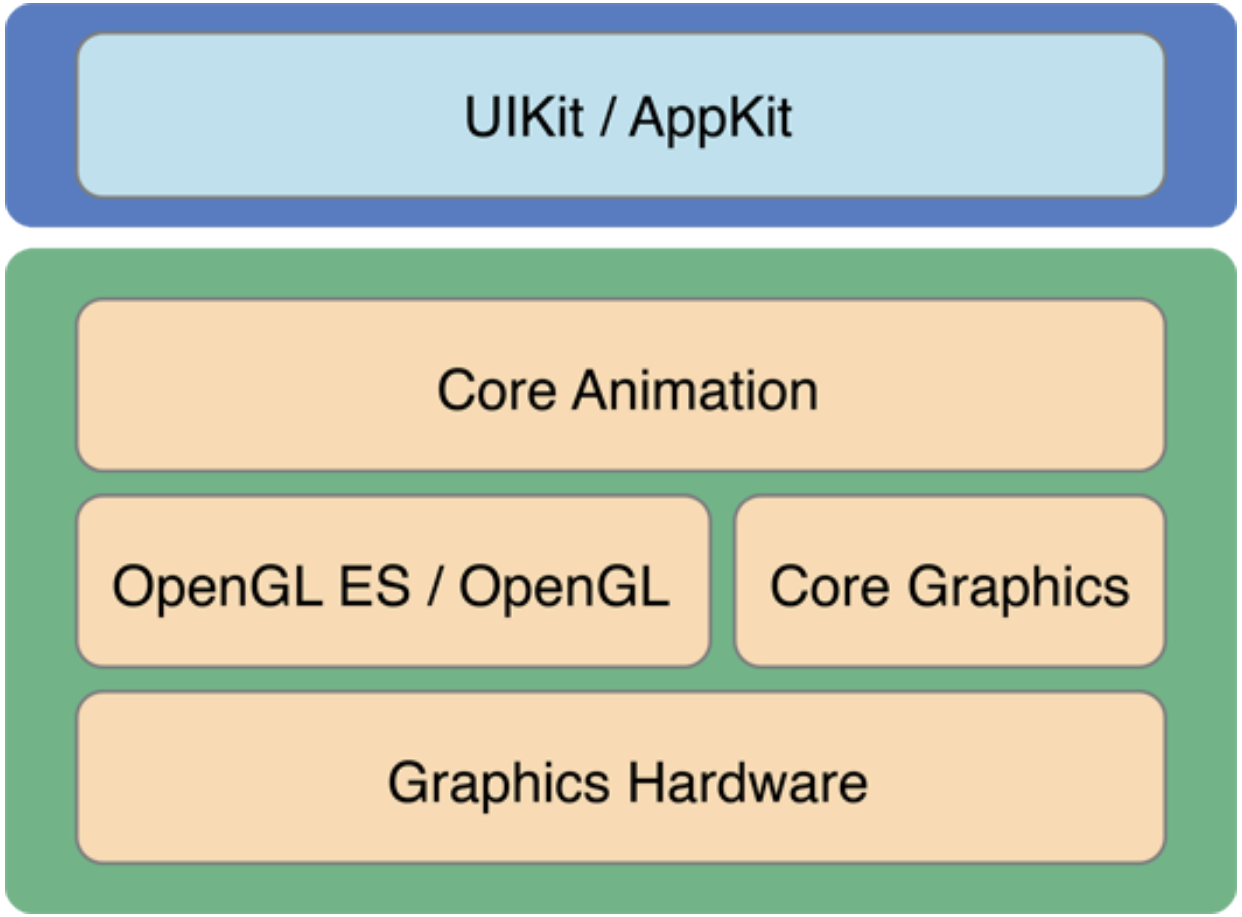
UIKit: UIView animations

```
[UIView animateWithDuration:1. animations:^{
    view.backgroundColor = [UIColor blueColor];
    view.alpha = 0.5;
} completion:^(BOOL finished){
    view.backgroundColor = [UIColor yellowColor];
}];

[UIView beginAnimations:nil context:nil];
[UIView setAnimationDuration: 5];
view.center = CGPointMake(80,80);
[UIView commitAnimations];
```

UIKit: UIView animations

```
[UIView transitionFromView:firstView toView:secondView
duration: 5
options:UIViewAnimationOptionAllowUserInteraction |
UIViewAnimationOptionTransitionFlipFromLeft
completion:^(BOOL finished){

    NSLog(@"transition completed");
}];
```

CoreAnimation



CoreAnimation

CoreAnimation — фреймворк, заведующий анимацией.

- Также, управляет передачей контента на GPU и его отображением.
- Сердце фреймворка — CALayer.
- CALayer — объектная абстракация OpenGL текстуры.
- CoreAnimation использует GPU для анимации и отображения контента.

CoreAnimation

В iOS все UIView-объекты имеют связанный с ними CALayer, который и занимается рендерингом контента.

- @property(nonatomic, readonly) CALayer* layer;

На Mac OS X использование CALayer для NSView опционально, на iOS же они используются всегда.

CoreAnimation — не система рисования сама по себе, но технология, обрабатывающая и передающая данные на GPU.

CoreAnimation

CoreAnimation так же позволяет создавать более изощренные анимации для CALayer:

```
CABasicAnimation* animation = [CABasicAnimation animationWithKeypath:@”background”];
animation.fromValue = [UIColor redColor].CGColor;
animation.toValue = [UIColor blueColor].CGColor;
animation.duration = 2.;
animation.autoReverses = YES;
animation.repeatCount = FLT_MAX;
[layer addAnimation: animation forKey:nil];
```

References

CoreAnimation:

https://developer.apple.com/library/ios/documentation/cocoa/conceptual/coreanimation_guide/Introduction/Introduction.html#//apple_ref/doc/uid/TP40004514-CH1-SW1

Quartz 2D:

https://developer.apple.com/library/ios/documentation/2DDrawing/Conceptual/DrawingPrintingiOS/Introduction/Introduction.html#//apple_ref/doc/uid/TP40010156

Заключение

Сердцем iOS является ее быстрая и отзывчивая графическая система.

iOS предоставляет возможность работы с графикой на любом уровне:

низкоуровневый OpenGL и объектно-ориентированный UIKit.

Также, используется специальный фреймворк CoreAnimation, позволяющий создавать сложные графические эффекты и анимации.

Система CoreGraphics позволяет создавать свои сложные графические элементы и выводить их в изображения.

Спасибо за внимание!

Жду ваших вопросов.