# Stanford CS193p

## Developing Applications for iOS
## Spring 2016

# Today

- Animation Continued
  Dynamic Animation Demo - DropIt
- CoreMotion
  Where is the device in space?

# Demo

- Dropit
  Sort of a "pre-Tetris" demo

# Core Motion

- API to access motion sensing hardware on your device
- Primary inputs: Accelerometer, Gyro, Magnetometer
  - Not all devices have all inputs (e.g. only later model devices have a gyro)
- Class used to get this input is `CMMotionManager`
  - Use only one instance per application (else performance hit)
  - It is a "global resource," so getting one via a class method somewhere is okay
- Usage
  1. Check to see what hardware is available
  2. Start the sampling going and poll the motion manager for the latest sample it has

  ... or ...

  1. Check to see what hardware is available
  2. Set the rate at which you want data to be reported from the hardware
  3. Register a closure (and a queue to run it on) to call each time a sample is taken

# Core Motion

- Checking availability of hardware sensors

  ```
  var {accelerometer,gyro,magnetometer,deviceMotion}Available: Bool
  ```

  The "device motion" is a combination of all available (accelerometer, magnetometer, gyro).

  We'll talk more about that in a couple of slides.

- Starting the hardware sensors collecting data

  You only need to do this if you are going to poll for data.

  ```
  func start{Accelerometer,Gyro,Magnetometer,DeviceMotion}Updates()
  ```

- Is the hardware currently collecting data?

  ```
  var {accelerometer,gyro,magnetometer,deviceMotion}Active: Bool
  ```

- Stop the hardware collecting data

  It is a performance hit to be collecting data, so stop during times you don't need the data.

  ```
  func stop{Accelerometer,Gyro,Magnetometer,DeviceMotion}Updates()
  ```

# Core Motion

- Checking the data (polling not recommended, more later)

```
var accelerometerData: CMAccelerometerData
```

CMAccelerometerData object provides `var acceleration: CMAcceleration`

```
struct CMAcceleration {
    var x: Double  // in g (9.8 m/s/s)
    var y: Double  // in g
    var z: Double  // in g
}
```

This raw data includes acceleration due to gravity

So, if the device were laid flat, `z` would be `1.0` and `x` and `y` would be `0.0`

# Core Motion

- Checking the data (polling not recommended, more later)

```
var gyroData: CMGyroData
```

CMGyroData object provides `var rotationRate: CMRotationRate`

```
struct CMRotationRate {
    var x: Double  // in radians/s
    var y: Double  // in radians/s
    var z: Double  // in radians/s
}
```

Sign of the rotation data follows right hand rule

The data above will be biased

# Core Motion

- Checking the data (polling not recommended, more later)

```
var magnetometerData: CMMagnetometerData
CMMagnetometerData object provides var magneticField: CMMagneticField
struct CMMagneticField {
    var x: Double  // in microteslas
    var y: Double  // in microteslas
    var z: Double  // in microteslas
}
```

The data above will be biased

# CMDeviceMotion

- Acceleration Data in CMDeviceMotion

```
var gravity: CMAcceleration
var userAcceleration: CMAcceleration // gravity factored out using gyro
```

- Rotation Data in CMDeviceMotion

```
var rotationRate: CMRotationRate // bias removed from raw data using accelerometer
var attitude: CMAttitude         // device's attitude (orientation) in 3D space
class CMAttitude: NSObject       // roll, pitch and yaw are in radians
    var roll: Double    // around longitudinal axis passing through top/bottom
    var pitch: Double   // around lateral axis passing through sides
    var yaw: Double     // around axis with origin at CofG and ⊥ to screen directed down
}
// other mathematical representations of the device's attitude also available
```

# CMDeviceMotion

- Magnetic Field Data in CMDeviceMotion

```
var magneticField: CMCalibratedMagneticField
struct CMCalibratedMagneticField {
    var field: CMMagneticField
    var accuracy: CMMagneticFieldCalibrationAccuracy
}
```

accuracy can be …

- CMMagneticFieldCalibrationAccuracyUncalibrated
- CMMagneticFieldCalibrationAccuracyLow
- CMMagneticFieldCalibrationAccuracyMedium
- CMMagneticFieldCalibrationAccuracyHigh

# Core Motion

- Registering a block to receive Accelerometer data

```
func startAccelerometerUpdatesToQueue(queue: NSOperationQueue,
                                      withHandler: CMAccelerometerHandler)
typealias CMAccelerationHandler = (CMAccelerometerData?, NSError?) -> Void
```

queue can be an NSOperationQueue() you create or NSOperation.mainQueue (or currentQueue)

- Registering a block to receive Gyro data

```
func startGyroUpdatesToQueue(queue: NSOperationQueue,
                             withHandler: CMGyroHandler)
typealias CMGyroHandler = (CMGyroData?, NSError?) -> Void
```

- Registering a block to receive Magnetometer data

```
func startMagnetometerUpdatesToQueue(queue: NSOperationQueue,
                                     withHandler: CMMagnetometerHandler)
typealias CMMagnetometerHandler = (CMMagnetometerData?, NSError?) -> Void
```

# Core Motion

- Registering a block to receive DeviceMotion data

```
func startDeviceMotionUpdatesToQueue(queue: NSOperationQueue,
                                     withHandler: CMDeviceMotionHandler)
typealias CMDeviceMotionHandler = (CMDeviceMotion?, NSError?) -> Void
```

queue can be an NSOperationQueue() you create or NSOperation.mainQueue (or currentQueue)

Errors … CMErrorDeviceRequiresMovement
CMErrorTrueNorthNotAvailable
CMErrorMotionActivityNotAvailable
CMErrorMotionActivityNotAuthorized

# Core Motion

- Setting the rate at which your block gets executed

```
var accelerometerUpdateInterval: NSTimeInterval
var gyroUpdateInterval: NSTimeInterval
var magnetometerUpdateInterval: NSTimeInterval
var deviceMotionUpdateInterval: NSTimeInterval
```

- It is okay to add multiple handler blocks
  - Even though you are only allowed one `CMMotionManager`
  - However, each of the blocks will receive the data at the same rate (as set above)
  - (Multiple objects are allowed to poll at the same time as well, of course.)

# Demo

- More DropIt
  - Make DropIt's gravity match real gravity