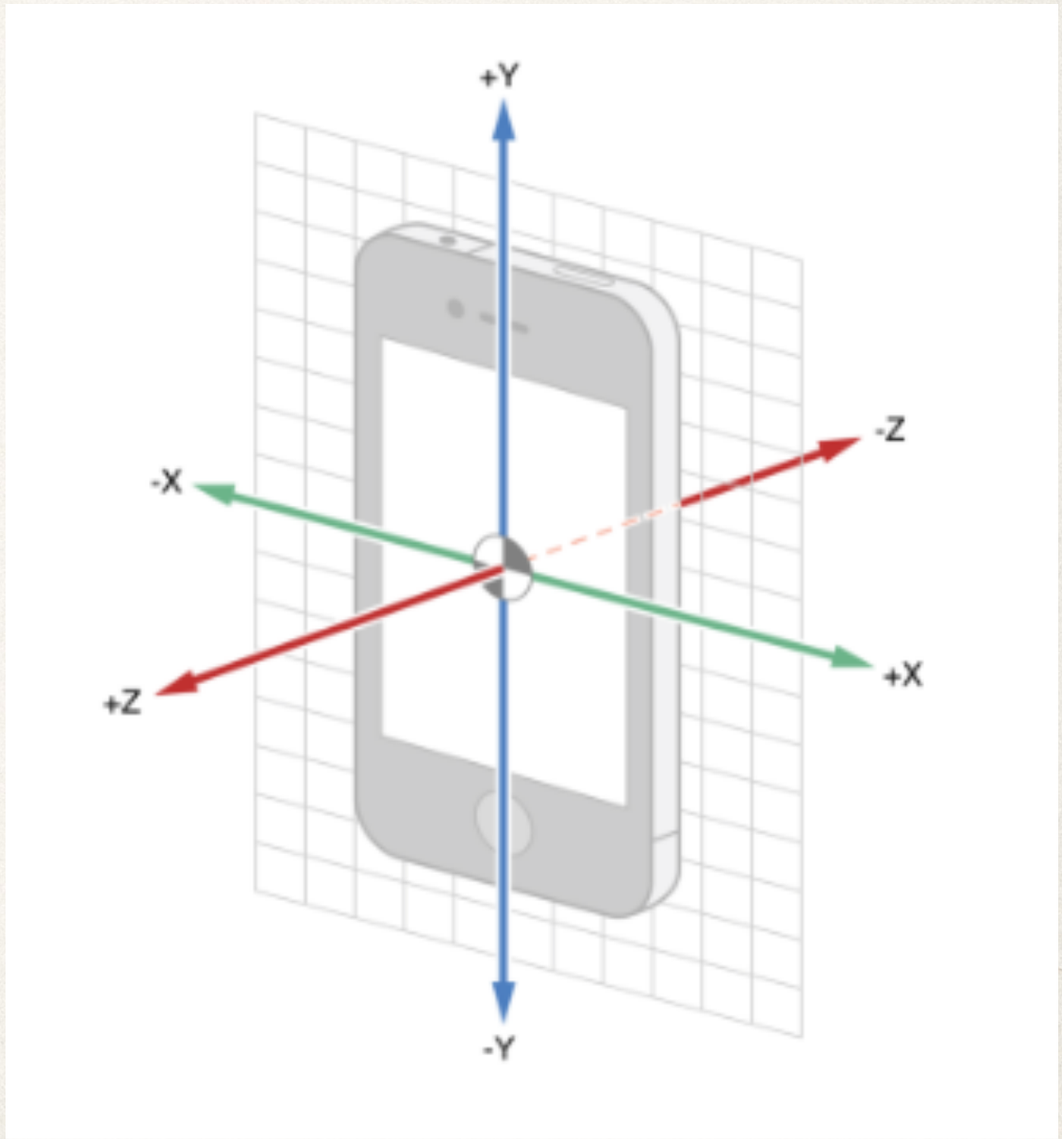
Motion Controls

Dr. Sarah Abraham

University of Texas at Austin
CS329e
Spring 2020

Motion Events

- Generated when user moves, shakes or tilts the device
- Detected by accelerometer:
  - One in each axis (X, Y, Z)
  - Measures velocity over time along a linear path
- And gyroscope:
  - Measures rate of rotation around three axes (X, Y, Z)



Device Orientation

- Basic physical orientations available in `UIDevice` class
  - `UIDeviceOrientationLandscapeLeft`
  - `UIDeviceOrientationLandscapeRight`
  - `UIDeviceOrientationPortrait`
  - `UIDeviceOrientationPortraitUpsideDown`
  - `UIDeviceOrientationFaceUp`
  - `UIDeviceOrientationFaceDown`
  - `UIDeviceOrientationUnknown`

Device vs Interface Orientations

- Device orientation is related to the physical orientation of the device
- Interface orientation is related to the interface display's orientation for the viewer:
  - `UIInterfaceOrientationPortrait`
  - `UIInterfaceOrientationPortraitUpsideDown`
  - `UIInterfaceOrientationLandscapeLeft`
  - `UIInterfaceOrientationLandscapeRight`
- Use device orientation for motion events, use interface orientation for designing displays

Shake Gesture

- Accelerometer determines that shake gesture occurred
- Operating system creates `UIEvent` to pass to active apps
- Event includes:
  - Motion start
  - Motion stop
  - Timestamp
- Object in app designated the “first responder” handles this event

Motion Event Handling

- Appropriate view controller made first responder:

```
func canBecomeFirstResponder() -> Bool { return true }
```

- Implement motion handling:

```
func motionBegan(motion: UIEventSubtype, withEvent: UIEvent
event)

func motionEnded(motion: UIEventSubtype, withEvent: UIEvent
event)

func motionCancelled(motion: UIEventSubtype, withEvent:
UIEvent event)
```

Core Motion

- Framework for handling more generalized motion inputs
- Supports access to both raw and processed accelerometer data
- Wide range of sources
  - Accelerometer, pedometer, magnetometer, altitude, attitude, motion activity etc
- Not available to test in simulator — must use a device

CMMotionManager

- Shared instance throughout app to handle motion data
- Provides interface for four motion data types:
  - Accelerometer
  - Gyro
  - Magnetometer
  - deviceMotion

Motion Types

- Accelerometer
  - Instantaneous acceleration in 3 dimensions
- Gyroscope
  - Instantaneous rotation in 3 dimensions
- Magnetometer
  - Device orientation relative to Earth's magnetic field
- Device-motion
  - Processed motion inputs (acceleration, rotation, orientation, etc) for device

Using CMMotionManager

1. Declare `import CoreMotion`

2. Instantiate `CMMotionManager` as a property within the necessary view controller

   - `let manager = CMMotionManager()`

3. Check for data on given operation queue

   - Uses closure functionality

Checking for Accelerometer Data

```
if manager.isAccelerometerAvailable {

    manager.accelerometerUpdateInterval = 0.1

    manager.startAccelerometerUpdates(to: .main) {
    (data, error) in

        guard let data = data, error == nil else {

         /* guard ensure nil values caught so handle nil values
         here */ }

        }

    /* perform actual processing of data here */

}
```

startAccelerometerUpdates

- `to:` takes an `OperationQueue`
  - `.main` puts the check for updates on the main operation queue
- OperationQueues maintain a list of Operations to complete and prioritize execution of these tasks
  - A new Queue will always executed on a separate thread
- OperationQueues use the Dispatch framework to initiate execution
  - `DispatchQueue.main.async` exercises a given task asynchronously on the main thread

Optimizing Motion Data

- Retrieve motion data on its own thread and dispatch results asynchronously to main thread:

```
let queue = DispatchQueue(label: "motion")
manager.startDeviceMotionUpdates(to: queue) {
    (data, error) in

    /* motion processing here */

    DispatchQueue.main.async {
        /*update main thread here*/
    }
}
```

Guards

- Statements usually used to prevent unwrapping (or passing) nil values
- If condition is not met, else block is called
  - Usually transfer control out of scope with a return statement
- Consider:

```
guard let data = data, error == nil else {

  return

}
```

Accelerometer Data

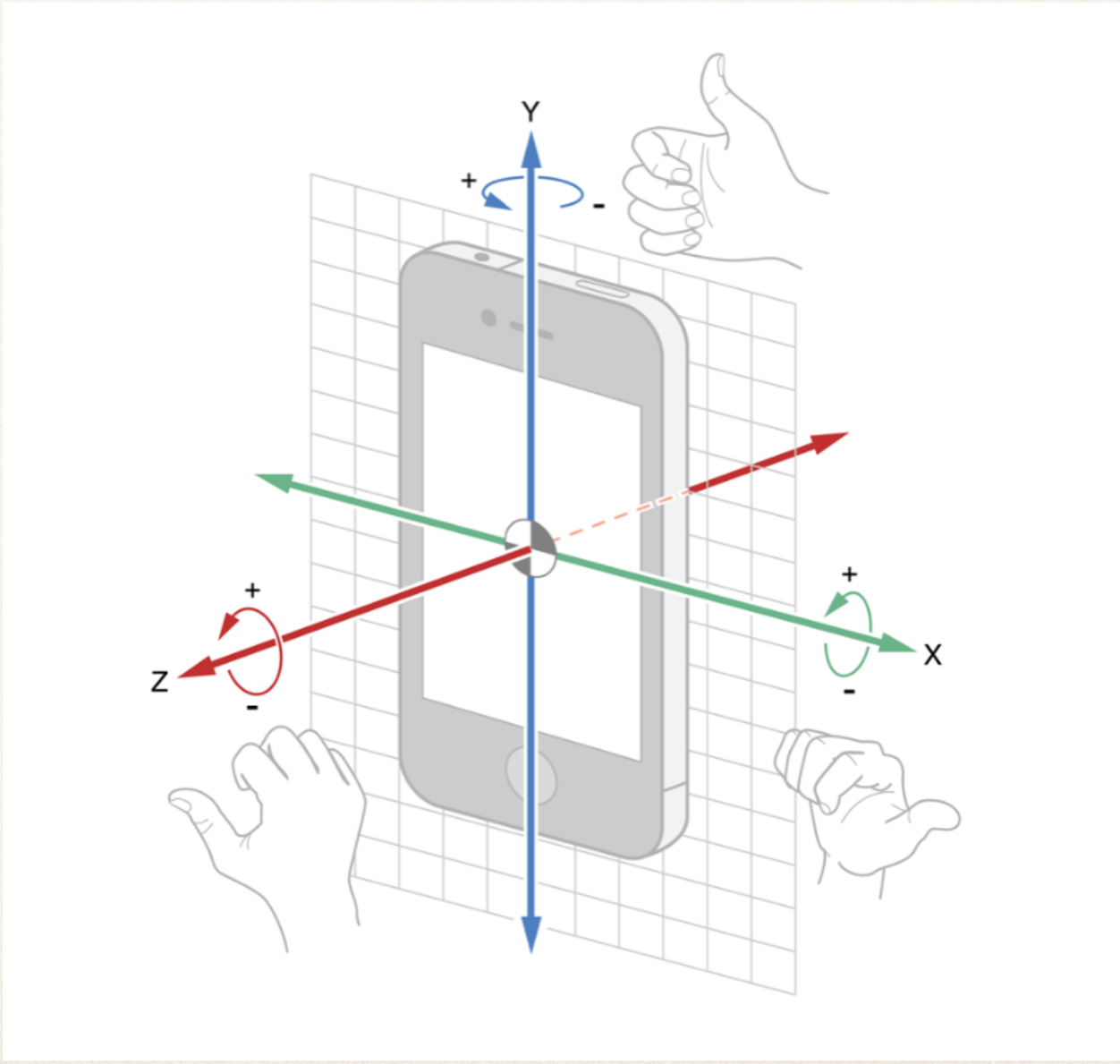
- Closure with accelerometer data called based on update interval

- `CMAccelerometerData` includes x, y and z
  - Represents amount of acceleration in G-forces

- Can process these values as angles
  - Angle of acceleration vector along x, y, and z axes respectively

Euler Angles

- Pitch (rotation around the X-axis)
- Roll (rotation around the Y-axis)
- Yaw (rotation around the Z-axis)



Gyroscope Data

- Similar to retrieving accelerometer data
- `startGyroUpdates` to start
- `gyroUpdateInterval` sets polling interval
- `gyroData` contains rotation information along x, y, z axes
  - Measured in radians per second

Magnetometer Data

- Same concept as accelerometer and gyroscope but with data on surrounding magnetic field
- Provides data along x, y, and z axes
- To measure changes in magnetic field, must store previously polled data to current data
- Aids detection of orientation and position in world
  - Used in conjunction with GPS data for navigation

Device Motion Data

- Provides unified access to device's motion data
- Similar start up to other modes of data
- Previously discussed data stored in `accelerometerData`, `gyroData`, and `magnetometerData` respectively
- Provides access to attitude or device orientation using `CMAttitude`

CMAttitude Data

- Provides 3 representations of data:
  - Euler angles (standard yaw, pitch and roll)
  - Quaternion (avoids gimbal lock)
  - Rotation matrix (representation used in graphics)
- Data exists within a frame of reference based on the device's resting orientation
  - Developer picks reference based on needs

CMAttitude Frame of Reference

- CMAttitudeReferenceFrameXArbitraryZVertical
  - X axis aligned with orientation during first call to motion
- CMAttitudeReferenceFrameXArbitraryCorrectedZVertical
  - Corrects orientation over time using magnetometer
- CMAttitudeReferenceFrameXMagneticNorthZVertical
  - X axis oriented toward magnetic north
- CMAttitudeReferenceFrameXTrueNorthZVertical
  - Corrects orientation for true north using GPS and magnetometer

Using Motion Data

- What are some applications that use motion data?
- How can the data we discussed help us achieve those results?

Instapoll Question: Core Motion

- What does `gyroData` contain?
  - 3 floats that represent x, y and z position
  - 3 floats that represent x, y and z velocity
  - 3 floats that represent x, y and z acceleration
  - 3 floats that represent angular velocity around x, y and z
  - Rotation data represented by Euler angles, a quaternion and a matrix

References

- Code examples from http://nshipster.com/cmdevicemotion/