What’s New in Core Motion

Session 423

Chris Moore
iOS Software Engineer

Agenda

1. App idea
2. Using Core Motion
3. Let’s code!
4. Deep dive
5. Camera Control

What's New in Core Motion

1. Raw magnetometer data
2. North-referenced attitude
3. Background support

Setting the Stage

Location

Attitude

Gyroscope

Measures rotation rate

Accelerometer

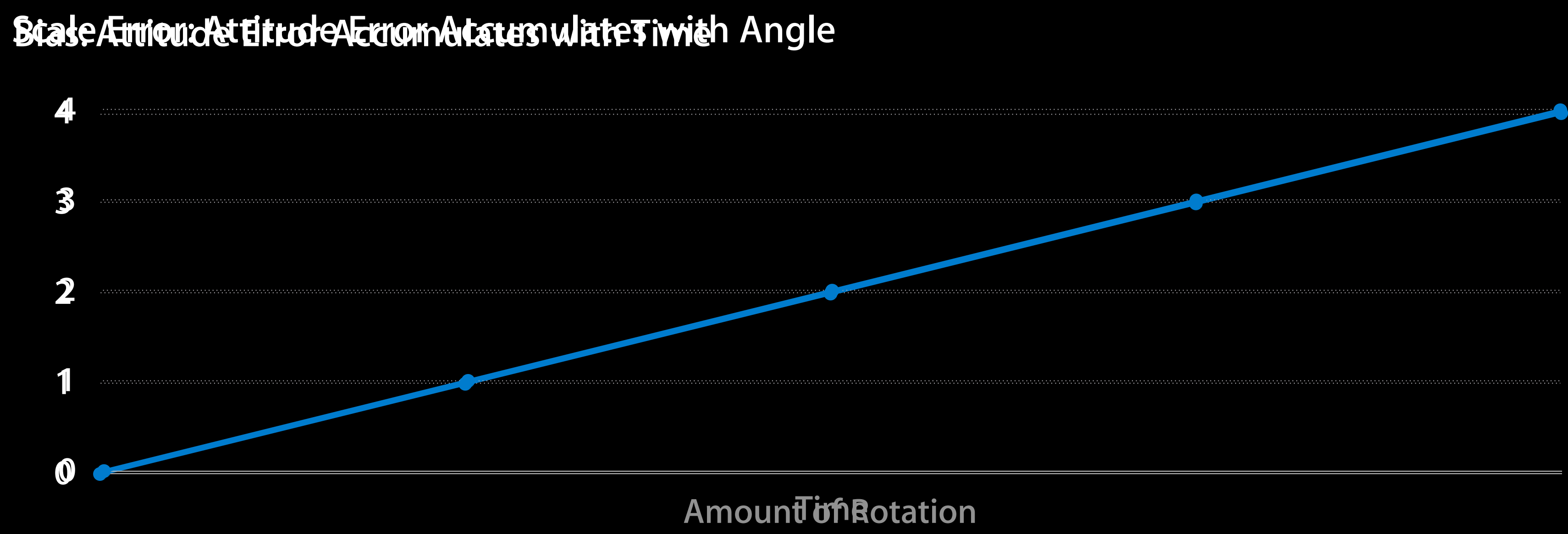
Measures gravity and user acceleration

Magnetometer

Measures magnetic field

Gyroscope Challenges

- No absolute reference
- Bias and Scale Error

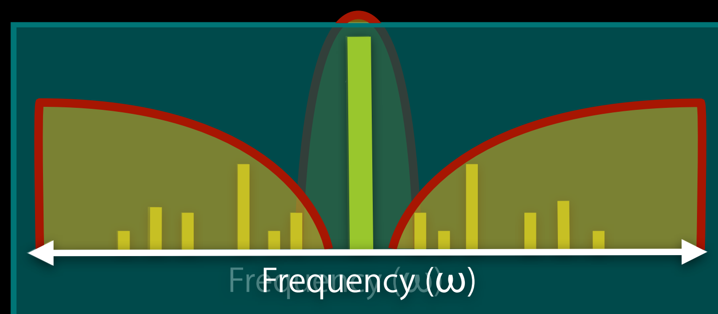


Accelerometer Challenges

- No yaw reference
- Filtering required to isolate gravity and user acceleration

High Pass Filter for User Acceleration

Low Pass Filter for Gravity



Magnetometer Challenges

- Internal interference (bias)
- External interference

Solution: Core Motion

Sensor fusion

- Full 3D attitude
  - Pitch and roll
  - Yaw
- Very responsive to changes in 3D attitude

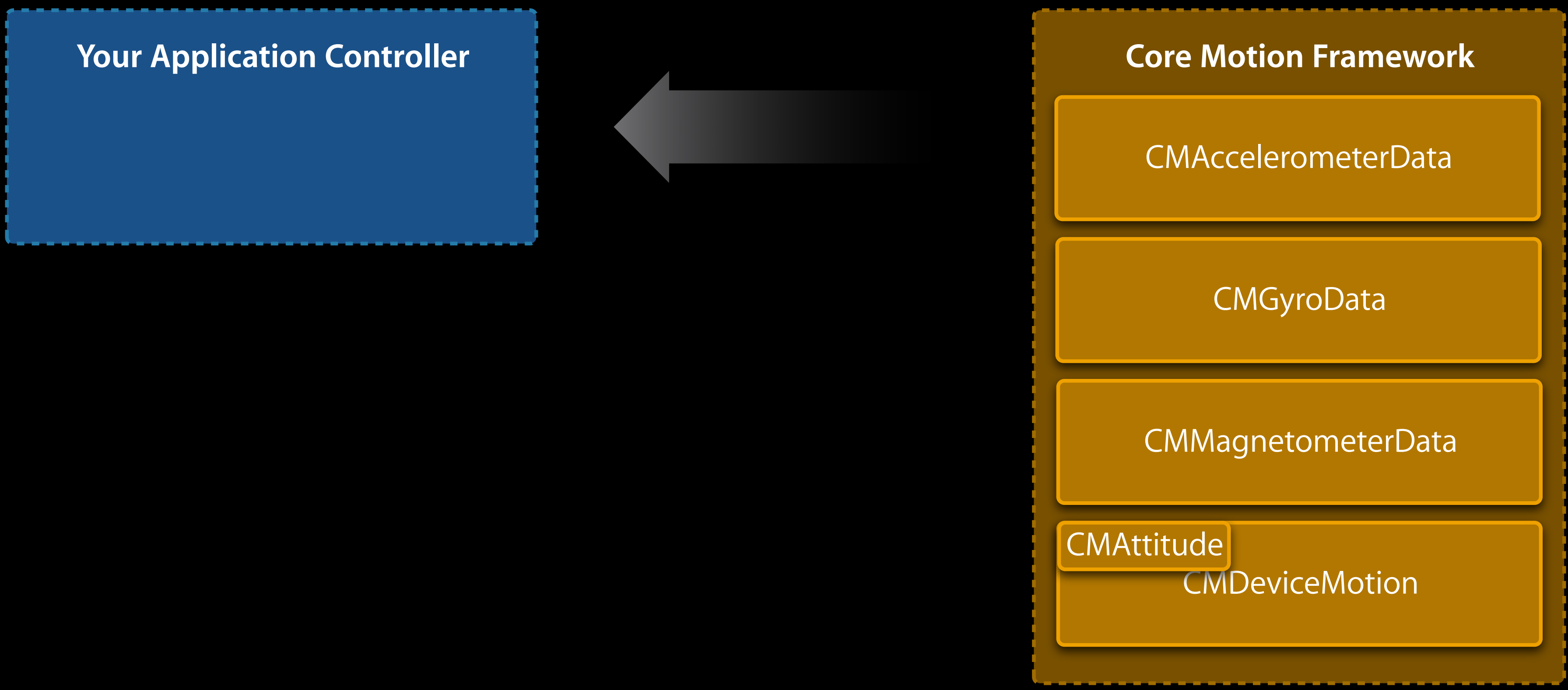
Using Core Motion

Core Motion Objects

Motion Manager

Core Motion Objects

Sensor data objects



Retrieving Data

Push and pull

- Push
  - Must provide NSOperationQueue and block
- Pull
  - Periodically ask CMMotionManager for latest sample
  - Often done when view is updated

Retrieving Data

Push vs. pull tradeoffs

	Advantages	Disadvantages	Recommendation
Push	Never miss a sample	Increased overhead Often best to drop samples	Data collection apps
Pull	More efficient Less code required	May need additional timer	Most apps and games

Threading

- Core Motion creates its own thread to:
  - Handle raw data from sensors
  - Run device motion algorithms
- Pushing data
  - Only your block will execute on your threads
- Pulling data
  - Core Motion will never interrupt your threads to send them data

Outline for Using Core Motion

1. Setup
2. Retrieve data
3. Clean-up

Step 1: Setup

```
-(void) startAnimation
{

    // Create a CMMotionManager instance
    motionManager = [[CMMotionManager alloc] init];

    // Ensure that the data we're interested in is available
    if (!motionManager.isAccelerometerAvailable) {
        // Fail gracefully
    }

    // Set the desired update interval (60Hz in this case)
    motionManager.accelerometerUpdateInterval = 1.0 / 60.0;

    // Start updates
    // Note: We could call the following here instead:
    // [motionManager startAccelerometerUpdatesToQueue:withHandler:]
    [motionManager startAccelerometerUpdates];

}
```

Step 2: Retrieving Data

```
-(void) drawView:(id)sender
{

    CMAccelerometerData *newestAccel = motionManager.accelerometerData;

    // ...
}
```

Step 3: Cleaning Up

```
-(void) stopAnimation
{

    [motionManager stopAccelerometerUpdates];
    [motionManager release];

    //...
}
```

Using Core Motion

Summary

- Two methods to receive data
  - Push
  - Pull
- Processing done on Core Motion's own thread
- Three steps to use Core Motion
  - Setup
  - Retrieve data
  - Clean-up

Demo

Xiaoyuan Tu
iOS Software Engineer and Scientist

From UIKit to Core Motion

UIKit Object	Core Motion Object
UIAccelerometer	CMMotionManager
UIAcceleration	CMAccelerometerData
UIAccelerationValue	double

Deep Dive into Device Motion

CMDeviceMotion Properties

CMDeviceMotion

- attitude
- gravity
- userAcceleration
- rotationRate
- magneticField

Attitude

```
@property(readonly, nonatomic) CMAttitude *attitude;
```

- Orientation of the device in 3D
- Ways to express
  - Rotation matrix
  - Quaternion
  - Euler angles (pitch, roll, yaw)

Reference Frame Choices

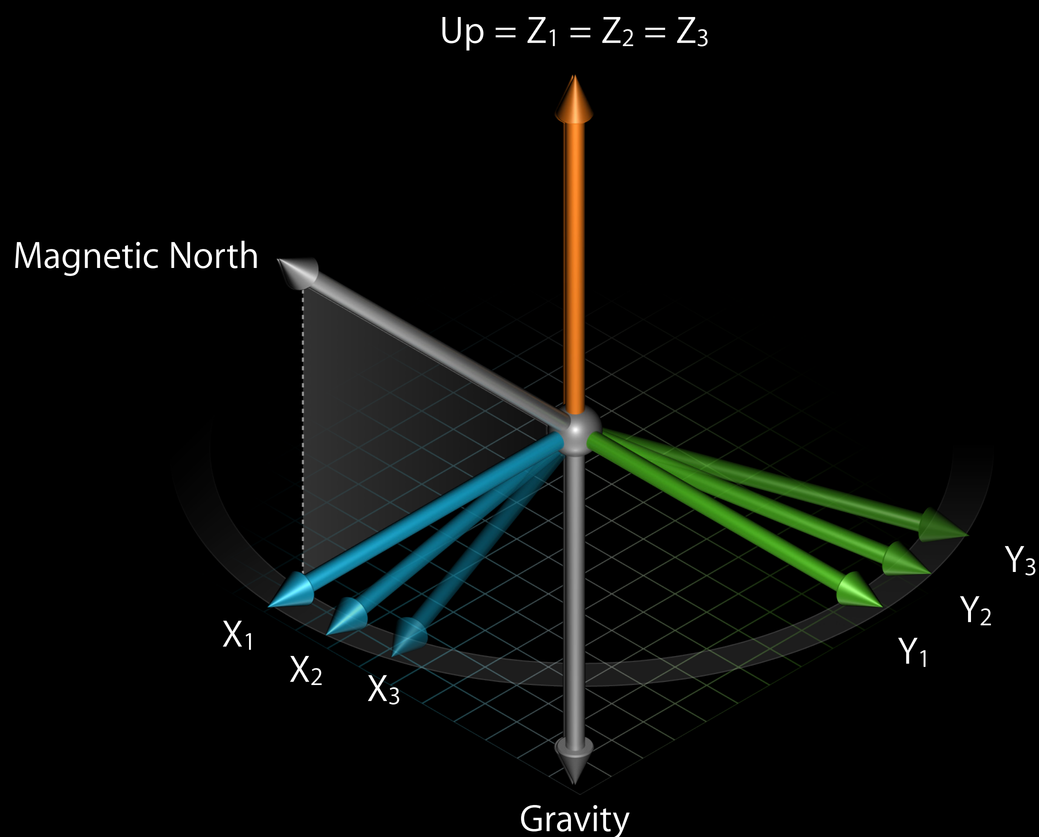
```
typedef enum {
  CMAttitudeReferenceFrameXArbitraryZVertical = 1 << 0,
  CMAttitudeReferenceFrameXArbitraryCorrectedZVertical = 1 << 1,
  CMAttitudeReferenceFrameXMagneticNorthZVertical = 1 << 2,
  CMAttitudeReferenceFrameXTrueNorthZVertical = 1 << 3
} CMAttitudeReferenceFrame;
```

```
+ (NSUInteger)availableAttitudeReferenceFrames
```

```
- (void)startDeviceMotionUpdatesUsingReferenceFrame:
```

Reference Frame Choices

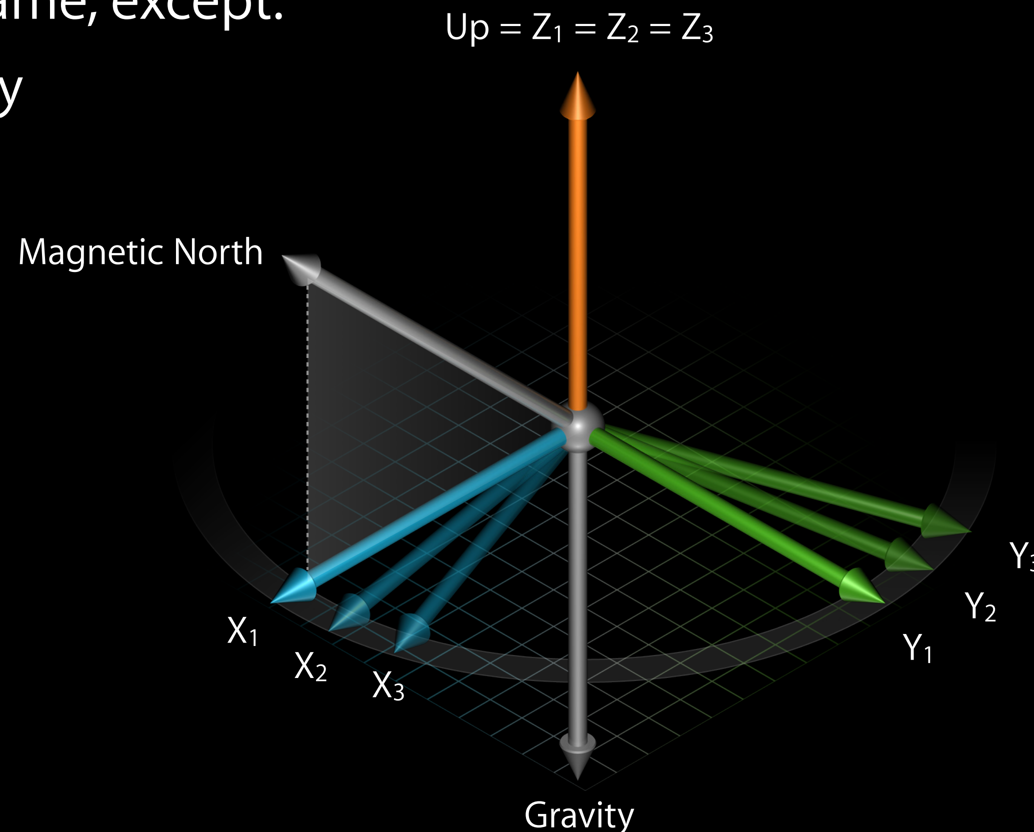
CMAttitudeReferenceFrameXArbitraryZVertical



Reference Frame Choices

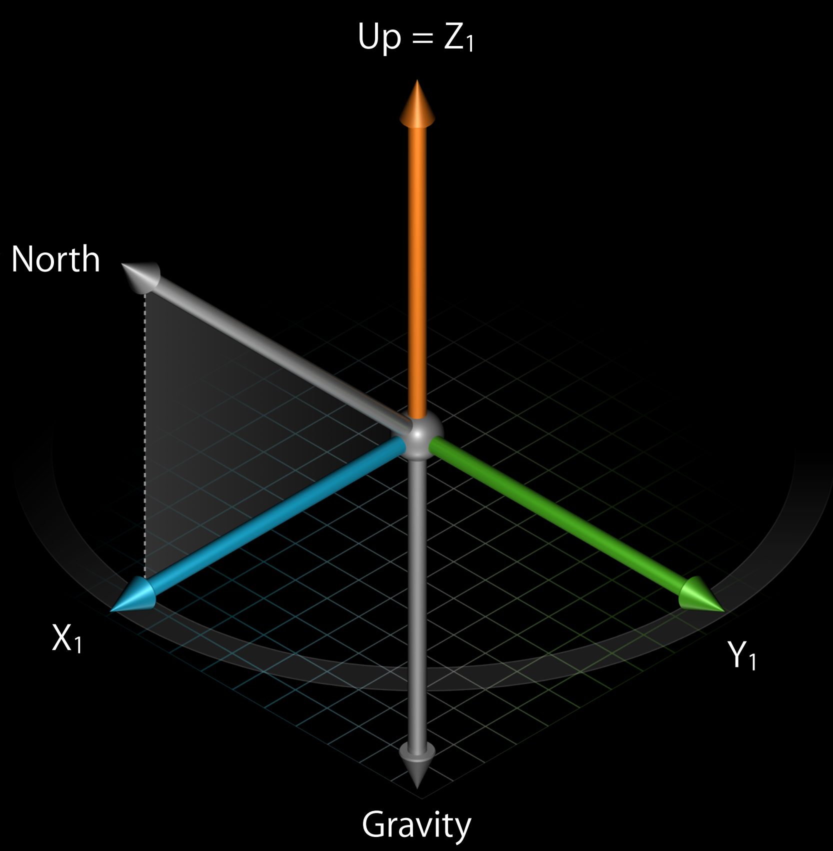
CMAttitudeReferenceFrameXArbitraryCorrectedZVertical

- Same as previous reference frame, except:
  - Better longterm yaw accuracy
  - Higher CPU usage



Reference Frame Choices

CMAttitudeReferenceFrameXMagneticNorthZVertical

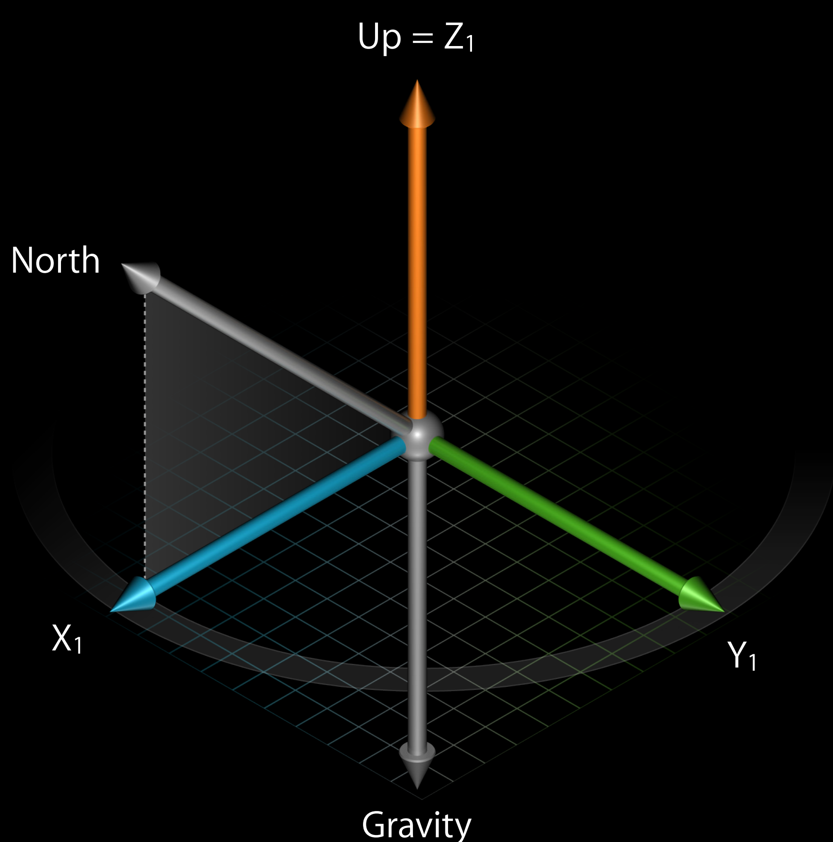


Calibration Requirements

- When using CMAttitudeReferenceFrameXMagneticNorthZVertical
  - Magnetometer calibration may be required
  - New CMMotionManager property to control display of calibration HUD
    - `@property(assign, nonatomic) BOOL showsDeviceMovementDisplay`

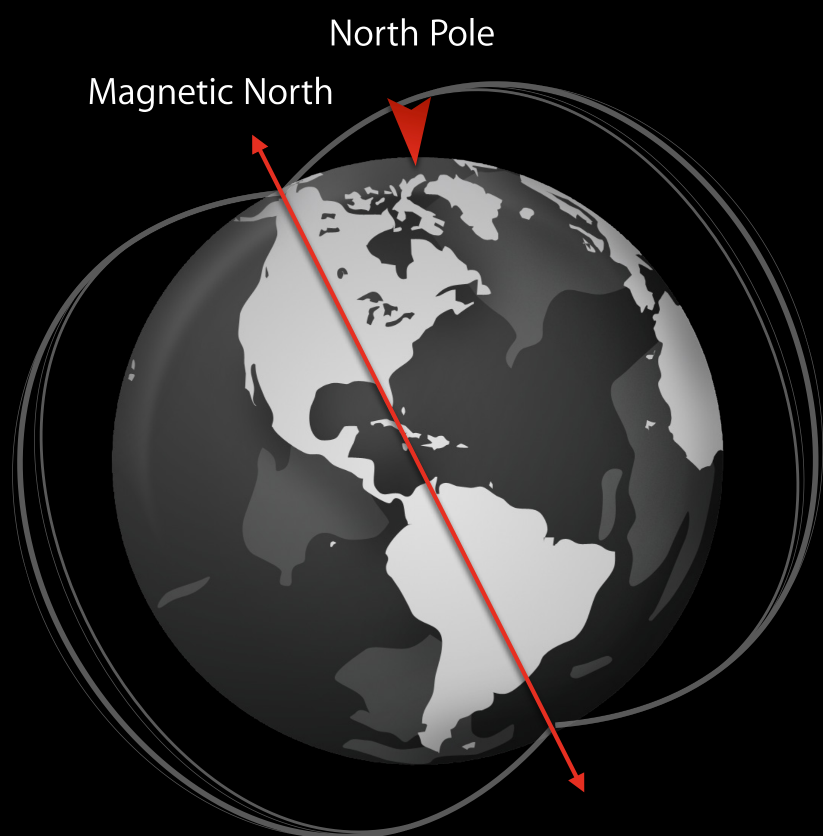
Reference Frame Choices

CMAttitudeReferenceFrameXTrueNorthZVertical



Magnetic North

Direction of Earth's Magnetic Field

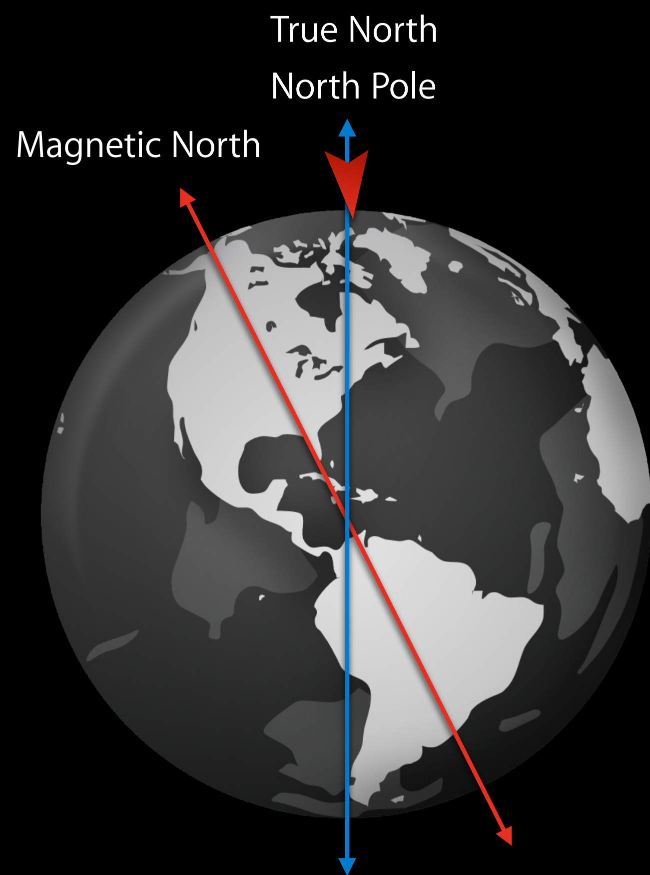


True North

Direction of the Earth's Geographic Poles

- Direction we are most familiar with
- Marked in the sky by the North Star, Polaris
- Reference point used when creating maps

- Can calculate true north given:
  - Magnetic north
  - Model of difference between true north and magnetic north around globe
  - Approximate location



Example

```
CMDeviceMotion *deviceMotion = motionManager.deviceMotion;
```

```
CMRotationMatrix R = deviceMotion.attitude.rotationMatrix;
```

```
CMAcceleration gravityReference = {0.0, 0.0, -1.0};
```

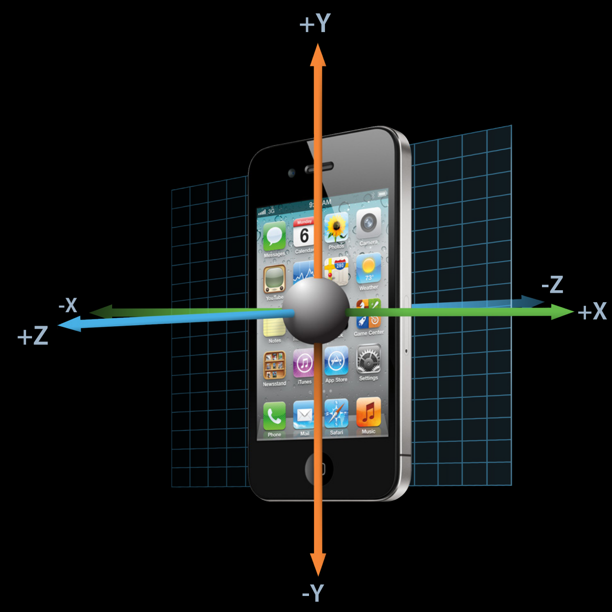
```
// gravityDevice == deviceMotion.gravity
gravityDevice = multiplyMatrixAndVector(R, gravityReference);
```

$$\texttt{deviceMotion.gravity} = R \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}$$

Gravity and User Acceleration

```
@property(readonly, nonatomic) CMAcceleration gravity;
@property(readonly, nonatomic) CMAcceleration userAcceleration;
```

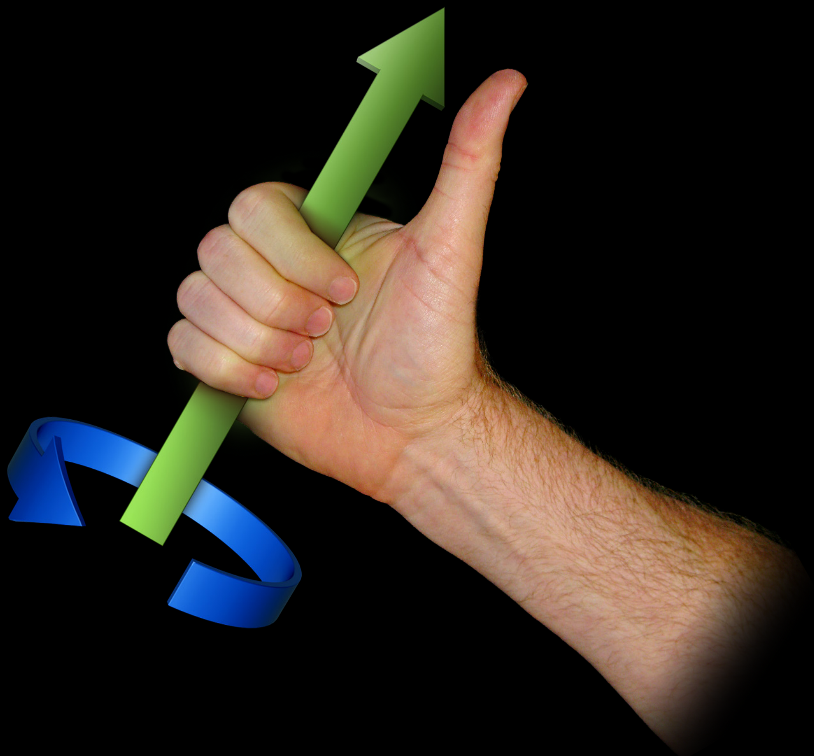
```
// Units are G's
typedef struct {
  double x;
  double y;
  double z;
} CMAcceleration;
```



Rotation Rate

```
@property(readonly, nonatomic) CMRotationRate rotationRate;
```

```
// Units are radians/second
typedef struct {
  double x;
  double y;
  double z;
} CMRotationRate;
```



Magnetic Field

```
@property(readonly, nonatomic) CMCalibratedMagneticField magneticField;
```

```
typedef struct {
    CMMagneticField field;
    CMMagneticFieldCalibrationAccuracy accuracy;
} CMCalibratedMagneticField;
```

Magnetic Field

```
// Units are microteslas
typedef struct {
    double x;
    double y;
    double z;
} CMMagneticField;
```

```
typedef enum {
  CMMagneticFieldCalibrationAccuracyUncalibrated = -1,
  CMMagneticFieldCalibrationAccuracyLow,
  CMMagneticFieldCalibrationAccuracyMedium,
  CMMagneticFieldCalibrationAccuracyHigh
} CMMagneticFieldCalibrationAccuracy;
```

Availability Matrix

	iPhone 4	iPhone 3GS	iPad 2	iPad	iPod touch (4th Gen.)	iPod touch (3rd Gen.)	Simulator
Accelerometer Data	✓	✓	✓	✓	✓	✓	
Gyro Data	✓		✓		✓		
Magnetometer Data	✓	✓	✓	✓			
Device Motion (original)	✓		✓		✓		
Device Motion (new)	✓		✓				

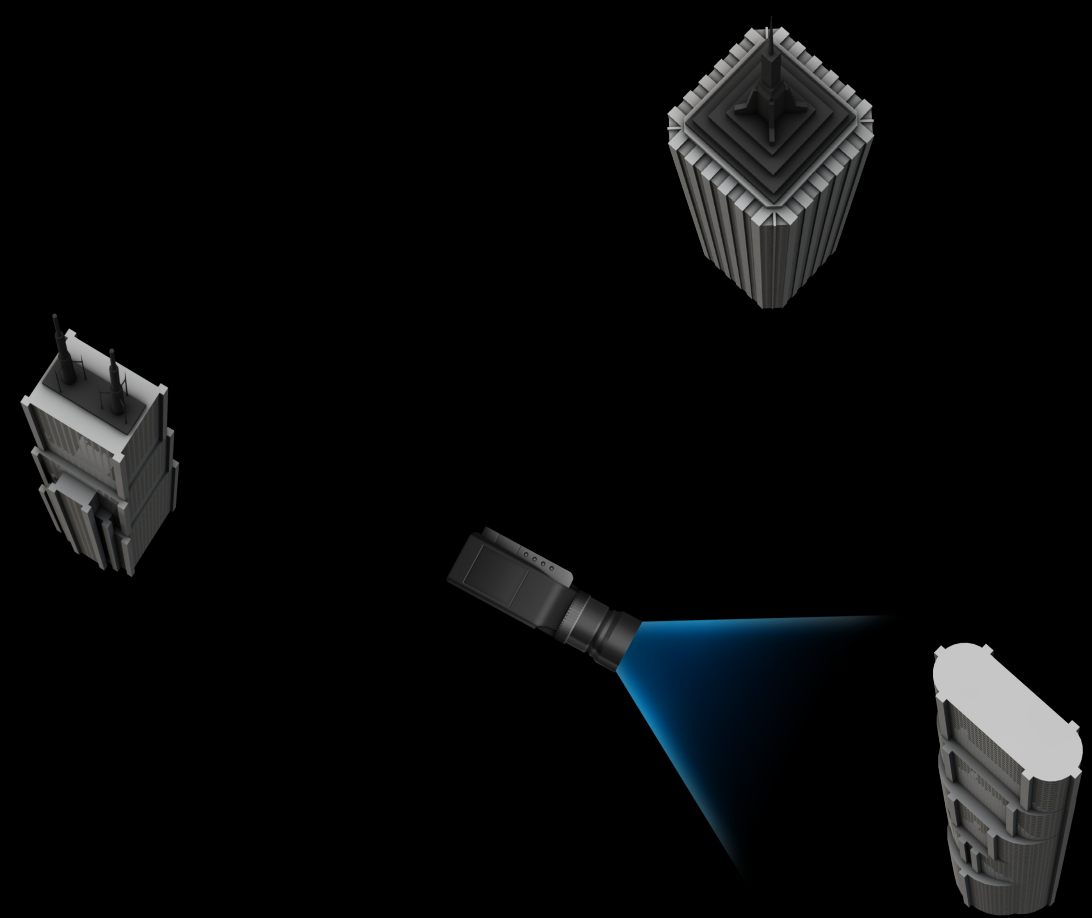
Demo

Xiaoyuan Tu
iOS Software Engineer and Scientist

Camera Control

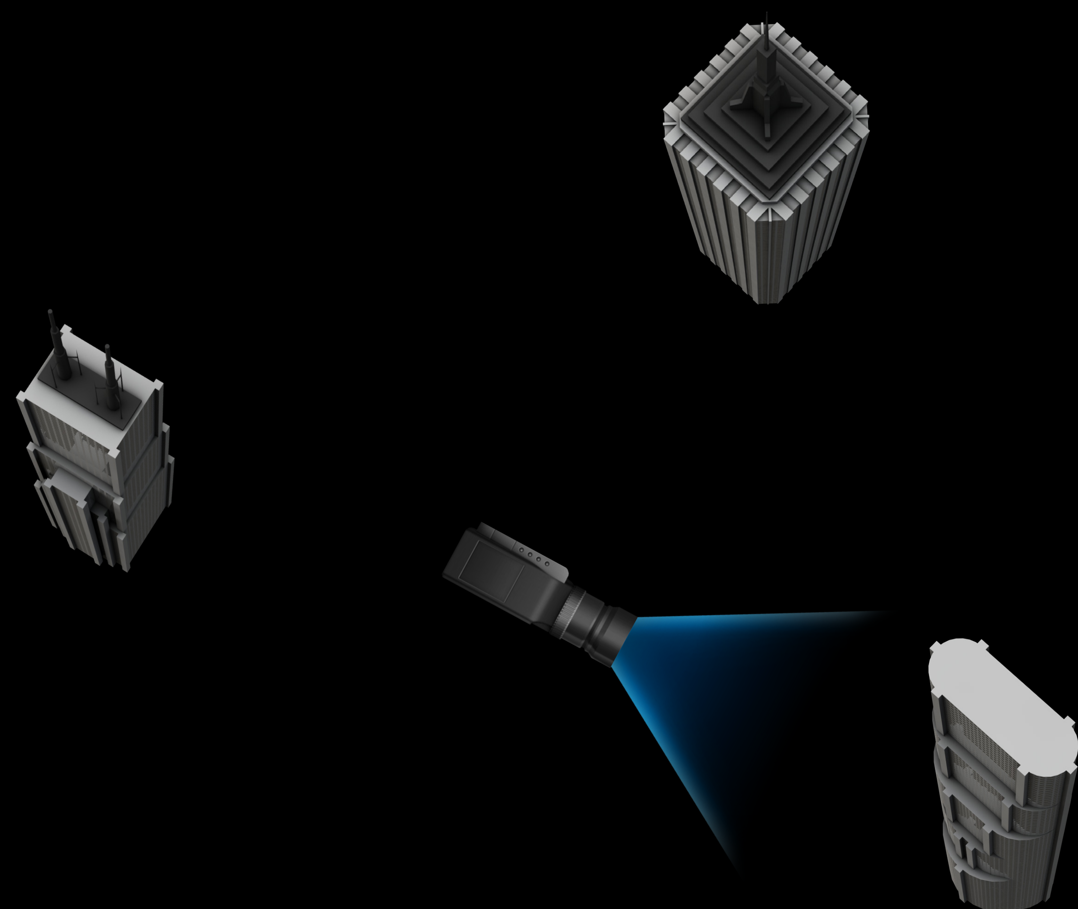
Attitude for Camera Control

Camera-centered pivot



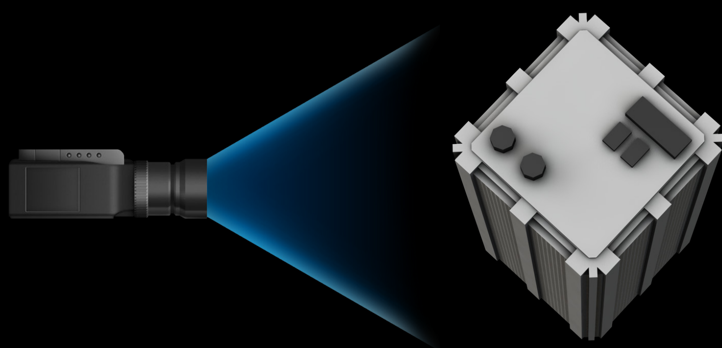
Attitude for Camera Control

Camera-centered pivot



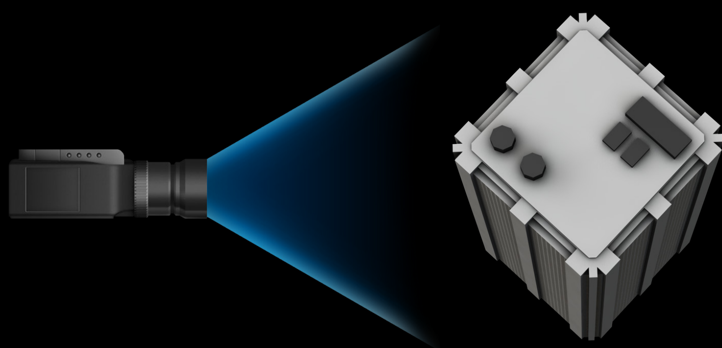
Attitude for Camera Control

Object-centered pivot



Attitude for Camera Control

Object-centered pivot



Rigid Body Transformations

$$\begin{bmatrix} R11 & R12 & R13 & Tx \\ R21 & R22 & R23 & Ty \\ R31 & R32 & R33 & Tz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Camera-centered Pivot

- pw = Camera position in world coordinates
- R = Attitude from Core Motion

```
CMRotationMatrix R = motionManager.deviceMotion.attitude.rotationMatrix;
```

- pd = Camera position in device coordinates
  - pd = R*pw = $\begin{bmatrix} pd_x \\ pd_y \\ pd_z \end{bmatrix}$

- 4x4 View matrix = $\begin{bmatrix} & R & & -pd_x \\ & & & -pd_y \\ & & & -pd_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Object-centered Pivot

- pw = Camera position in world coordinates = $\begin{bmatrix} pw_x \\ pw_y \\ pw_z \end{bmatrix}$
- R = Attitude from Core Motion

```
CMRotationMatrix R = motionManager.deviceMotion.attitude.rotationMatrix;
```

- 4x4 View matrix = $\begin{bmatrix} & R & & -pw_x \\ & & & -pw_y \\ & & & -pw_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Demo

More Information

Allan Shaffer
Graphics and Game Technologies Evangelist
aschaffer@apple.com

Documentation
Event Handling Guide for iOS
http://developer.apple.com

Apple Developer Forums
http://devforums.apple.com

Related Sessions

Essential Game Technologies for iOS Parts 1 (Repeat)	Marina Friday 10:15AM
Essential Game Technologies for iOS Part 2 (Repeat)	Marina Friday 11:30AM
Best Practices for OpenGL ES Apps in iOS	Mission Wednesday 4:30PM
Testing your Location—Aware App Without Leaving Your Chair	Mission Friday 9:00AM

Labs

Core Motion Lab	Graphics, Media, & Games Lab B Friday 11:30am-1:30pm

WWDC2011