lecture 12

ARKit and CoreML

cs198-001 : fall 2017

Attendances

Check that we've accounted for all of your attendances!

See Sarah's pinned post on Piazza

Make a **private piazza post** if there is an error

Remember - we allow 3 unexcused absences

Lab this week

Final custom app check in with your TA

You should be *at least* 60% done with your final app

Use this time to work on your project, ask for design advice, work with teammates, etc.

Final Project Submission

Final project due Wednesday of RRR week

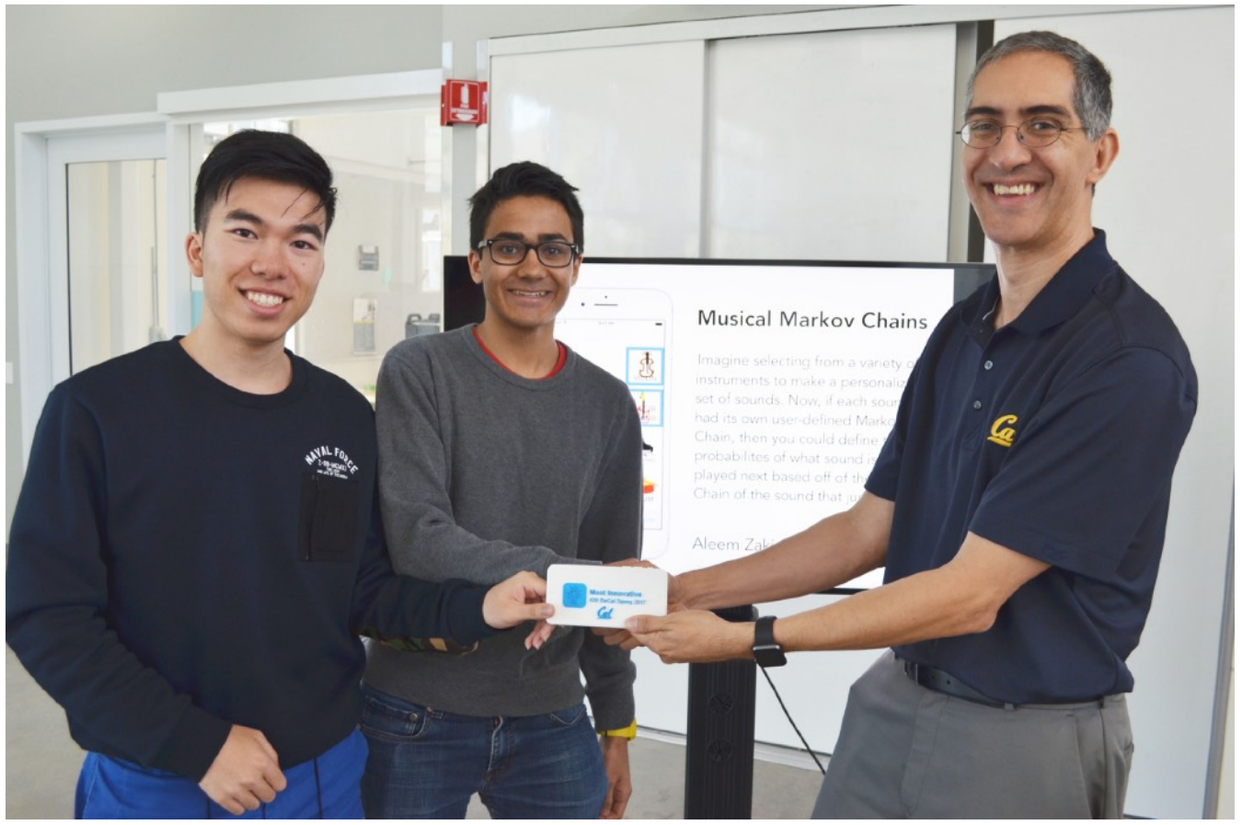
Submit via google form (on website)

- github repo link
- video walkthrough of your app
- app logo
- app description

Graded on implementation of app (35% of total grade)

If chosen to present, we will notify you on Wednesday (by the end of the day)

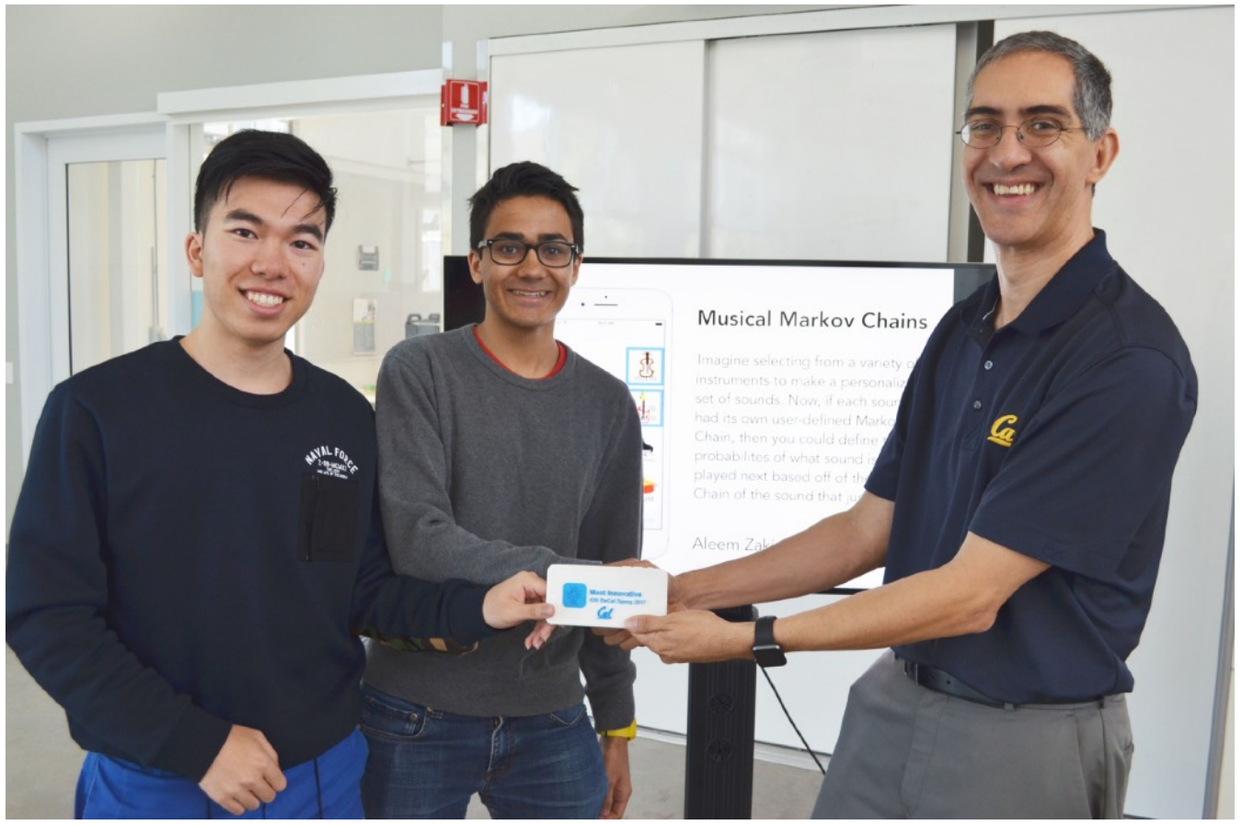
Final Project Showcase



Friday, Dec 8th at 3:30-5pm in the HP Auditorium

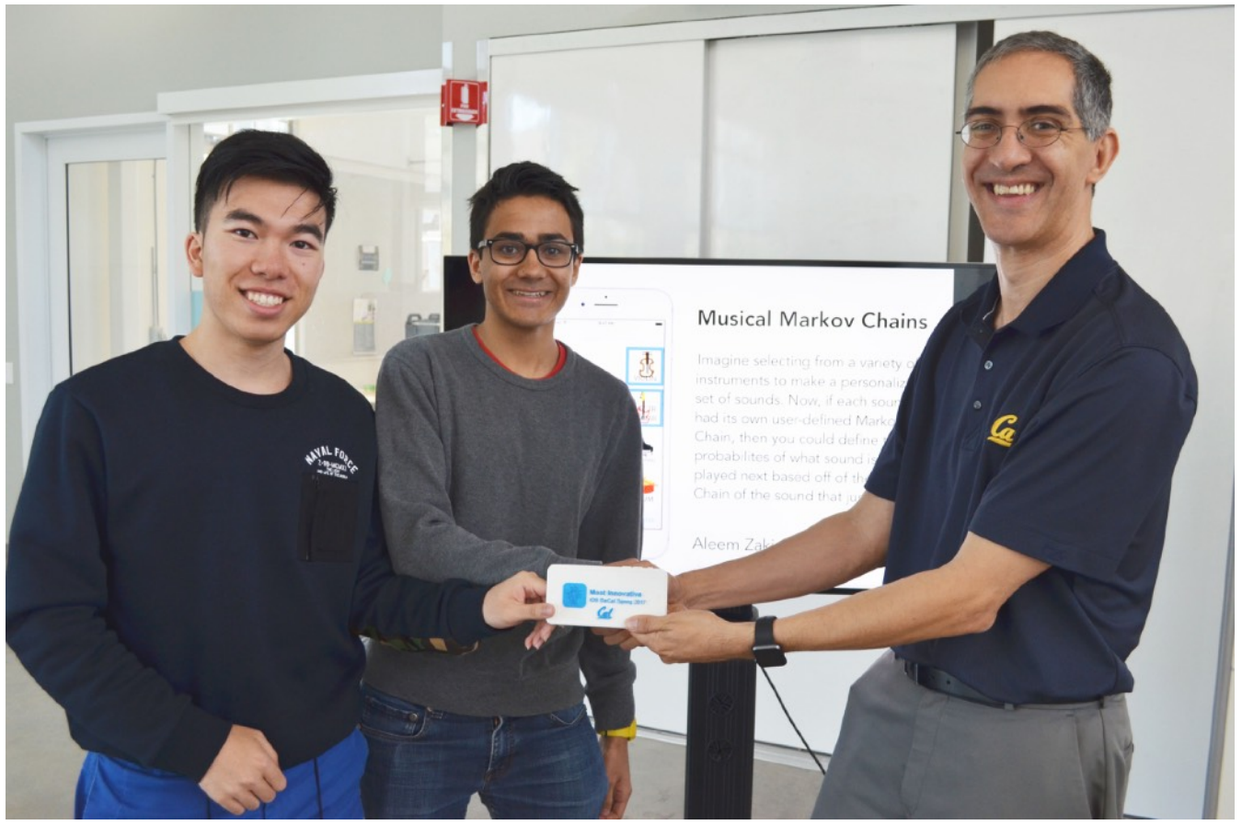
attendance mandatory for Pass

Final Project Showcase



Student app presentations, feedback / Q&A, and awards

Final Project Showcase



Awards

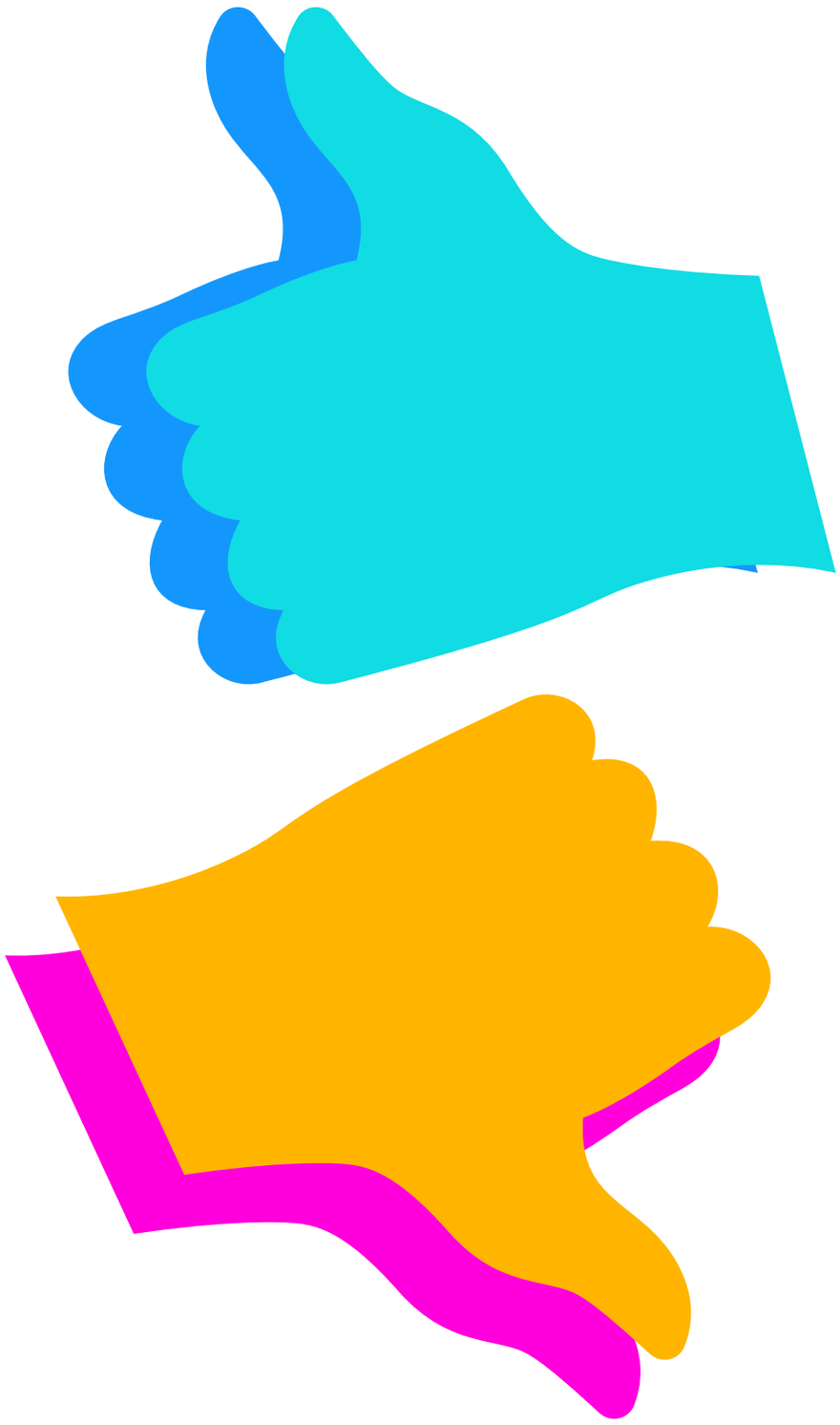
People’s Choice • Most Innovative • Social Impact • Best Design • Most Technical

Apply to TA!

CS 399 - 1 unit

apply by Dec 9

tinyurl.com/iosdecalTA



Course Evaluation

Loved the decal? Hated the decal? Let us know!

tinyurl.com/iosdecalfeedback

ARKit and CoreML

Gokul Swamy・iOS DeCal

Background

- ARKit and CoreML released at WWDC 2017
- Both represent a shift in traditional developer model
  - Artist/Researcher creates model
  - Apple takes care of hard part of implementation
  - You just have to connect the above

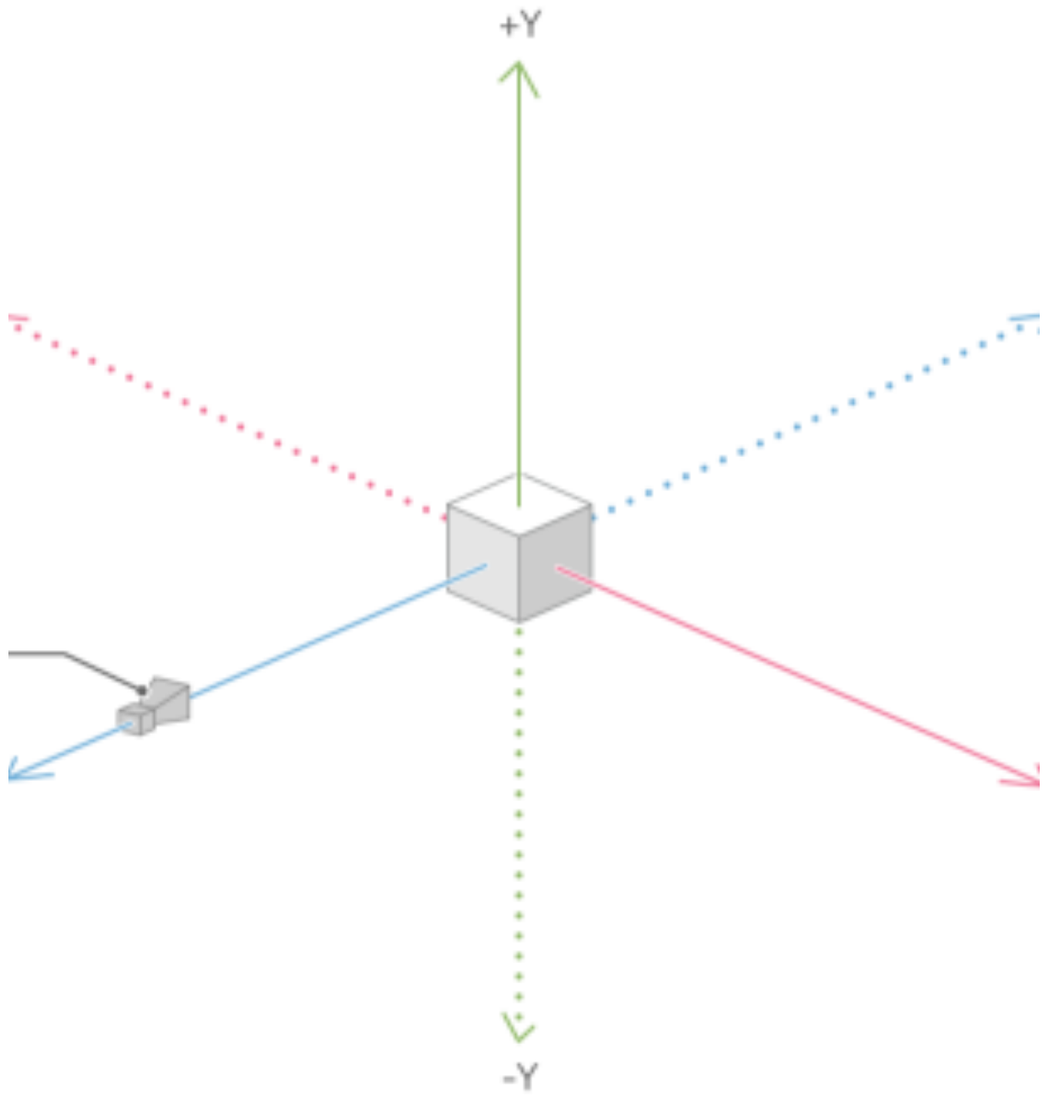
ARKit

- Hard Parts (taken care of):
    - Visual Inertial Odometry (fusion of cameras and motion sensing) to estimate how much device has moved
    - Finding planes and surfaces in surrounding area as well as estimating lighting for simulated objects
    - Detecting faces and matching with 3D characters
- Remember to import!

Demo

Coordinates

- Coordinates may not be what you'd expect (x is left-right, y is up-down, z is forward-backwards)
- Camera is at (0, 0, 0)
  - So don't place object there!
- 1 float = 1 meter
- Angles should be specified in radians



Position, Scale, Rotation

- Position: SCNVector3(x, y, z)
    - z = 0.5 means place object half a meter in front of user (remember coordinate axes)
- Scale: SCNVector3(x, y, z)
    - z = 0.5 means compress in outwards direction to half
- Rotation: SCNVector4(x, y, z, w)
    - x, y, and z correspond to whether the object should be rotated around an axis, w is the angle in radians

SceneKit

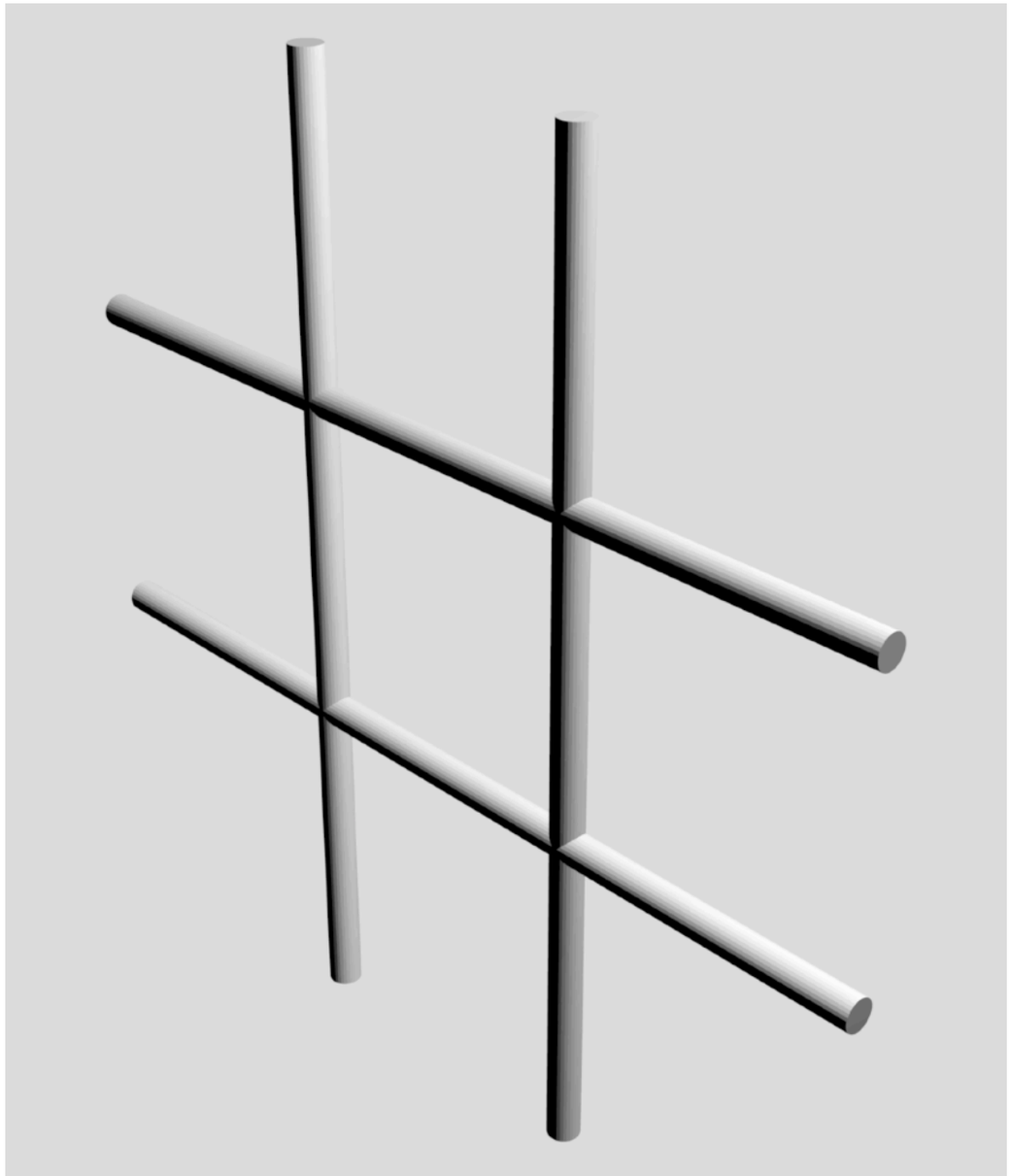
- Used to handle 3D objects and handles interfacing with ARKit

- Can use SpriteKit to handle 2D objects (see **here** for details)

- Can create simple shapes (cubes, spheres, …) as well as importing more complicated ones

Getting Objects

- Download objects made by artists
    - https://www.turbosquid.com/
    - https://sketchfab.com/
- Create your own
    - https://www.blender.org/
- **.dae** or **.scn** files



Setting the Scene

- Add a **ARSceneView** and add constraints to make it fill the whole screen (recommended by HIG)
- Ask user for permission to use camera
    - Open up Info.plist and add a “Privacy - Camera Usage Description” key
    - Set the value for what text you want to prompt the user with.

Managing a Session

- Usually use **ARWorldTrackingConfiguration** unless you need something more specific
  - Tracks planes, feature points, and 6 degrees of freedom of device

```
override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    let configuration = ARWorldTrackingConfiguration()
    sceneView.session.run(configuration)
}
override func viewWillDisappear(_ animated: Bool) {
    super.viewWillDisappear(animated)
    sceneView.session.pause()
}
```

Tracking

- Feature points (notable points in the environment that make it easer for us to keep a simulated object stationary) detected automatically

- To detect planes add the following:

```
let configuration = ARWorldTrackingConfiguration()
configuration.planeDetection = .horizontal
sceneView.session.run(configuration)
```

Adding Simple Objects

- Create **SCNNode**
- Create Geometry (**SCN**____) and add to node
- Set position (rotation/scale if necessary) of node
- Add created node as child node of sceneView's root node

```
func addSphere(x: Double, y: Double, z: Double) {
    let sphere = SCNSphere(radius: 0.05)
    let sphereNode = SCNNode()
    sphereNode.geometry = sphere
    sphereNode.position = SCNVector3(x, y, z)
    sceneView.scene.rootNode.addChildNode(sphereNode)

}
```

Adding Complex Objects

- Safely load scene from file
- Create new node
- Add each of the children nodes of the root node of the loaded scene as children nodes of the new node
- Set position (and scale/rotation if necessary) of new node

Adding Complex Objects

- In Code:

```
func addBoard() {
    guard let boardScene = SCNScene(named: "board.dae") else {return}
    let boardNode = SCNNode()
    for childNode in boardScene.rootNode.childNodes {
        boardNode.addChildNode(childNode)
    }
    boardNode.position = SCNVector3(0, 0, -1)
    boardNode.scale = SCNVector3(0.1, 0.1, 0.1)
    boardNode.rotation = SCNVector4(1, 0, 0, Float.pi / 2)
    sceneView.scene.rootNode.addChildNode(boardNode)
}
```

Deleting Objects

- Just remove as child node of root node of sceneView

```
for node in self.sceneView.scene.rootNode.childNodes {
    node.removeFromParentNode()
}
```

Enabling Interaction

- Add Gesture Recognizer to sceneView

```
let tapGestureRecognizer = UITapGestureRecognizer(target: self, action:
    #selector(ViewController.didTap(withGestureRecognizer:)))
sceneView.addGestureRecognizer(tapGestureRecognizer)
```

- And extension to transformation matrix class

```
extension float4x4 {
    var translation: float3 {
        let translation = self.columns.3
        return float3(translation.x, translation.y, translation.z)
    }
}
```

Enabling Interaction

- One can check if we tapped on a feature point to add a node

```
@objc func didTap(withGestureRecognizer recognizer: UIGestureRecognizer) {
    let tapLocation = recognizer.location(in: sceneView)
    let hitTestResultsWithFeaturePoints = sceneView.hitTest(tapLocation, types: .featurePoint)
    if let hitTestResultWithFeaturePoints = hitTestResultsWithFeaturePoints.first {
        let translation = hitTestResultWithFeaturePoints.worldTransform.translation
        addBox(x: translation.x, y: translation.y, z: translation.z)
    }
}
```

- Or can check if we tapped a node that already exists and remove it

```
@objc func didTap(withGestureRecognizer recognizer: UIGestureRecognizer) {
    let tapLocation = recognizer.location(in: sceneView)
    let hitTestResults = sceneView.hitTest(tapLocation)
    guard let node = hitTestResults.first?.node else { return }
    node.removeFromParentNode()
}
```

Adding in Lighting

- Default lighting is probably sufficient unless you want to have some kind of effect (for example, a spotlight on a certain character)

```
func configureLighting() {
    sceneView.autoenablesDefaultLighting = true
    sceneView.automaticallyUpdatesLighting = true
}
```

Demo

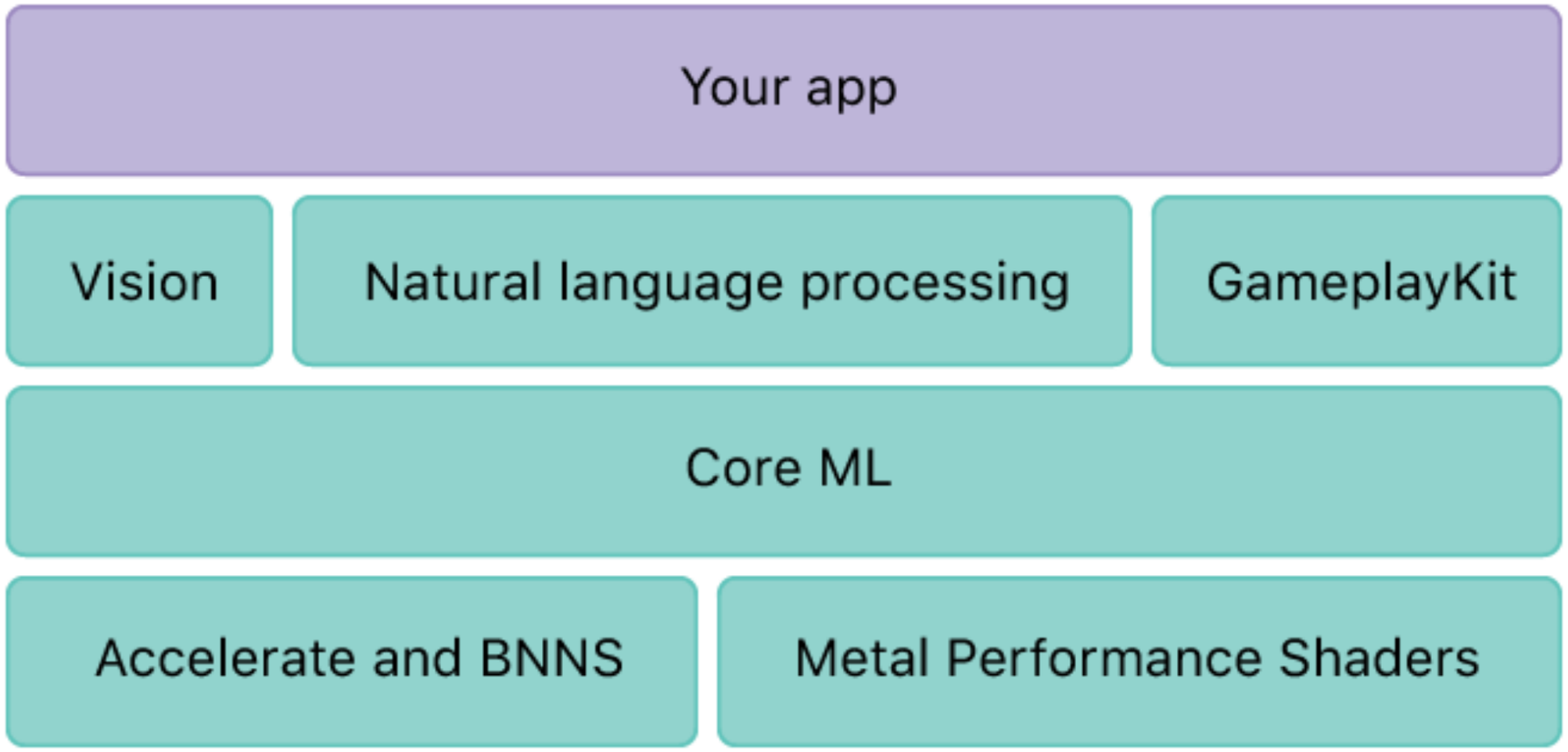
(Less Cool)

tinyurl.com/iosdecalfinalcheckin

CoreML

- An even bigger buzzword
- Lets you convert ML code written in Python into a **.mlmodel** file can be dropped directly into your app
- Hard part: optimizes code for iOS devices
- Some helpful preprocessing included
- Does not currently support on-device training

Architecture



(Apple)

(Custom*)

(Optimized Operations)

(GPU Stuff)

*** = don’t use a custom model unless Apple doesn’t cover your use case or you’re Andrew Ng**

Vision

- Allows you to do many things (detect face landmarks, OCR, scan barcodes, optical flow)

- Slower but more accurate than CoreImage and AVFoundation

- Three Steps

  - Call **VNRequestHandler**

  - Which executes a **VNRequest**

  - And returns some **VNObservation** for you to process

Vision: Code

- Be sure to add Camera Usage Description to Info.plist and import Vision

```
func detectText(image: UIImage){
    let textRequest = VNDetectTextRectanglesRequest(completionHandler: self.detectTextCompletionHandler)
    let textRequestHandler = VNImageRequestHandler(cgImage: image.cgImage!, options: [:])
    do {
        try textRequestHandler.perform([textRequest])
    } catch {
        print(error)
    }
}

func detectTextCompletionHandler(request: VNRequest, error: Error?){
    guard let results = request.results as? [VNTextObservation] else {return}
    var boxes = [VNRectangleObservation]()
    for result in results {
        if let characterBoxes = result.characterBoxes {
            for box in characterBoxes {
                boxes.append(box)
            }
        }
    }
    // do something here with boxes
}
```

Demo

NSLinguisticTagger

- Hello NS my old friend
- On device processing so no privacy worries
- Can identify language, tokenize (split up into chunks), lemmatize (give root form of word), and detect named entities (nonstandard words that may be important)
- Two Steps
  - Create **NSLinguisticTagger** with tagSchemes set to application
  - Set **tagger.string** to the text to be analyzed.

NSLinguisticTagger: Code

- For example, to detect dominant language:

```
let tagger = NSLinguisticTagger(tagSchemes: [.language], options: 0)
tagger.string = "NSLinguisticTagger provides text processing APIs."
let language = tagger.dominantLanguage
```

- For all use case sample code, check out **this link**

Custom Models

- A couple popular models already converted by Apple **here**
- In general, find code on Github or on **arXiv**
- Use python 2.7 package coremltools to convert code to .mlmodel format
- Then, instantiate instance of generated class and use **model.prediction()** method

coremltools Sources

Table 1 Models and third-party tools supported by Core ML Tools

Model type	Supported models	Supported tools
Neural networks	Feedforward, convolutional, recurrent	Caffe v1 Keras 1.2.2+
Tree ensembles	Random forests, boosted trees, decision trees	scikit-learn 0.18 XGBoost 0.6
Support vector machines	Scalar regression, multiclass classification	scikit-learn 0.18 LIBSVM 3.22
Generalized linear models	Linear regression, logistic regression	scikit-learn 0.18
Feature engineering	Sparse vectorization, dense vectorization, categorical processing	scikit-learn 0.18
Pipeline models	Sequentially chained models	scikit-learn 0.18

Python Package Setup

- Need to be in python 2.7
- pip install virtualenv
- virtualenv --python=/usr/bin/python2.7 py2.7
- source py2.7/bin/activate
- pip install numpy
- pip install scipy
- pip install sklearn
- pip install coremltools

Spam Detection

- SMS Spam Detection is a mostly solved problem that can effectively treated without using neural networks

- We’re going to train a classifier to detect spam based on a provided dataset

- We’ll be using the Naive Bayes, Support Vector Machine, and Random Forest classifiers

Loading the Data

- We'll be using data from **here**
    - In machine learning, getting good and clean data is often the hardest part
- First, we load and split our data

```
raw_data = open('./smsspamcollection/SMSSpamCollection.txt', 'r')
sms_data = []
for line in raw_data:
    split_line = line.split("\t")
    sms_data.append(split_line)

sms_data = np.array(sms_data)
X = [x.lower() for x in sms_data[:, 1]]
y = sms_data[:, 0]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=13)
```

- We train on the train data and check accuracy on test data

Testing our Models

- Use library you've selected and create models to solve your problem (each of the models hear include a preprocessing vectorizer and then a classifier)
- Train the models on the test, predict on the test set and compare to the true output for the test set to get accuracy

```
pipeline_1 = Pipeline([('vect', CountVectorizer()),('clf', MultinomialNB())])
pipeline_2 = Pipeline([('vect', CountVectorizer()),('clf', LinearSVC())])
pipeline_3 = Pipeline([('vect', CountVectorizer()),('clf', RandomForestClassifier())])
pipelines = [pipeline_1, pipeline_2, pipeline_3]

for pipeline in pipelines:
    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)
    print(classification_report(y_test, y_pred, target_names=["ham", "spam"]))
```

Converting a Model

- Train an instance of the best model on all the data (not just the training set)

- Use appropriate coremltools converter for your chosen framework (see **here** for full documentation)

- Save the model as a .mlmodel file

```
import coremltools

vectorizer = CountVectorizer()
vectorized = vectorizer.fit_transform(X)
model = LinearSVC()
model.fit(vectorized, y)

coreml_model = coremltools.converters.sklearn.convert(model, "message", "label")
coreml_model.save('SpamDetector.mlmodel')
```

Using an .mlmodel in Your App

- Drag and drop the file into your project
- Import CoreML
- Format your data as an **MLMultiArray** and pass it into the model.prediction() method
- Pick outputs from prediction (for example: most likely class, class probabilities)

Spam Detection: Code

```
func isSpam(message: String) -> Bool {
    let wordsFile = Bundle.main.path(forResource: "words_ordered", ofType: "txt")
    let message = "You have won the GRAND PRIZE reply to claim your FREE MONEY"
    do {
        let wordsFileText = try String(contentsOfFile: wordsFile!, encoding: String.Encoding.utf8)
        var wordsData = wordsFileText.components(separatedBy: .newlines)
        wordsData.removeLast() // Trailing newline.
        var wordsDict: [String: Int] = [:]
        for (idx, word) in wordsData.enumerated() {
            wordsDict[word] = idx
        }

        let posVect = vectorize(message: message, mapping: wordsDict)
        if let vect = posVect {
            let model = SpamDetector()
            let prediction = try model.prediction(message: vect)
            if prediction.label == "ham" {
                return false
            } else {
                return true
            }
        }

    }
    catch {
        print("ERROR")
    }
    return true
}
```

Vectorization: Code

```
func vectorize(message: String, mapping: [String: Int]) -> MLMultiArray? {
    var message = message
    message = message.lowercased()
    var vector = [Double](repeating: 0.0, count: mapping.count)
    for word in message.split(separator: " "){
        if let index = mapping[String(word)] {
            vector[index] += 1.0
        }
    }
    do {
        let formatted = try MLMultiArray(shape: [NSNumber(integerLiteral: vector.count)], dataType: .double)
        for (idx, elem) in vector.enumerated() {
            formatted[idx] = NSNumber(value: elem)
        }
        return formatted
    } catch {
        return nil
    }
}
```

Demo

…and checkin

Spam Detection: Closing Points

- Check out my **blog** if you're curious for how to integrate the code so far into an iMessage App

- Nowadays, this kind of problem is solved using neural networks (see **here** if curious)

- Convolutional Neural Networks work very well with text classification (see **here** and **here** if curious)

- We'd use k-fold cross validation to tune hyper-parameters as well as tf-idf vectorization if we were to do this irl

- Additionally, we'd use a much larger and more diverse dataset

Machine Learning

- If you'd like to learn more, check out the following resources

  - **ML@B Blog** by Geng and Shih
  - **STATS 385** Cheat Sheet
  - **CS 189** by Sahai
  - **CS 231n** by Karpathy
  - **Deep Learning** by Goodfellow and Bengio

Conclusion

- iOS devices are capable of some pretty awesome stuff
- Team up with people who are amazing at what they do (researchers, artists, …) to build more complex apps
- You've learned how to build awesome things that people all around the world can use
    - So start making and never stop!