Spring 2017

iPhone Swift 3 Development for "What's Good?"

Cameron Reilly
The University of Akron, cjr61@zips.uakron.edu

The University of Akron

Computer Science

iPhone Swift 3 Development for “What’s Good?”

Author:
Cameron Reilly

Advisor:
Dr. Michael L. Collard

Introduction

The iPhone is the most widely used piece of personal technology used today. According to Craig Smith at expandedramblings.com, over 1 billion have been sold as of July 2016. That averages to just under a staggering 400 iPhones sold every minute. This rapid expansion of personal technology has created a massive market for iOS development with 40% of smartphone owners in the US being iPhone users (Elmer-DeWitt). With the incredible success of the game "Pokémon Go!" the market for widely used Augmented Reality phone applications has been created, and holds many opportunities for research and development.

The overall popularity of the iPhone, and the possibilities for development were the inspiration for the project "What's Good?" This project aims to use an augmented reality system to show reviews of restaurant locations above them in real time. Users will be able to hold their phone up to a restaurant while looking through a camera view and see review ratings, average costs, and the name of a dining location hovering above the restaurant on their screen. The information is drawn from an API, parsed, and placed into a field that renders on screen when certain location specific conditions are met.

In order for a system such as this to function properly there are many hardware components that must first be accessed by the application. The device orientation is taken from the gyroscope and accelerometer sensors. Additionally, the user's precise location and their directional heading relative to north are acquired from the internal compass and location services contained within the CoreLocation library. To create the background effect of augmented reality, the rear-facing camera is accessed and used as the background of the application. The system also obtains data from an API that possesses location-based data about restaurants and reviews of them. The information about the restaurant is then presented

to the user while they are facing the establishment. It should be noted that the camera is purely for aesthetic effect and does not employ any sort of image recognition algorithm to identify a logo or sign. Inconsistencies such as poor lighting, lack of logo and sign data, etc. at any non-franchise restaurants would create many problems that would be more difficult to solve than the scope of this project allows based on the time available. As there is no image recognition, the application will work even if the camera lens is completely covered.

Camera Access

Finding information on how to access the camera as a background element is proving to be much more difficult than previously anticipated. About 90% of the resources online are how to access the camera application from within the app, not how to use the image as a background. Further research into using a constantly updating image for the background may prove to be more fruitful than using the camera directly. The first attempt to access the live camera feed can be seen in figure 1:

```
class ViewController: UIViewController, UIImagePickerControllerDelegate, UINavigationControllerDelegate {

    var captureSession : AVCaptureSession?
    var cameraLayer : AVCaptureVideoPreviewLayer?

    @IBOutlet weak var cameraView: UIView!
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

    override func viewWillAppear(animated: Bool) {
        super.viewWillAppear(animated)

        captureSession = AVCaptureSession()
        captureSession?.sessionPreset = AVCaptureSessionPreset1920x1080         //POSSIBLE CHANGES TO
RESOULTION HERE

        var backCamera = AVCaptureDevice.defaultDeviceWithMediaType(AVMediaTypeVideo)
    }
}
```

Figure 1: First attempt to access camera feed.

In addition to this one month after the start of the project, the IDE Xcode updated to version 8.0. This update included an update to the swift language from version 2 to version 3. Luckily Xcode was able to update most of the syntax to its correct new form as seen in figure 2.

```
-    override func viewWillAppear(animated: Bool) {
+    override func viewWillAppear(_ animated: Bool) {
         super.viewWillAppear(animated)

         captureSession = AVCaptureSession()
         captureSession?.sessionPreset = AVCaptureSessionPreset1920x1080         //POSSIBLE CHANGES TO
 RESOULTION HERE

-        var backCamera = AVCaptureDevice.defaultDeviceWithMediaType(AVMediaTypeVideo)
+        var backCamera = AVCaptureDevice.defaultDevice(withMediaType: AVMediaTypeVideo)
     }
 }
```

Figure 2: Example of Xcode auto updated syntax.

Upon further research, the initial idea of using a constantly updating still image turned out to be the correct solution to getting camera feed without using the camera application. Instead a camera view is added to the model view controller. An AVCaptureSession is used in tandem with an AVCaptureStillImageOutput and an AVCaptureVideoPreviewLayer that was already in place as seen in figure 3. This allows a still image to be constantly updated with whatever the camera is currently looking at. CameraLayer was renamed to more properly reflect its purpose.

```
    var captureSession : AVCaptureSession?
-   var cameraLayer : AVCaptureVideoPreviewLayer?
+   var stillImageOutput : AVCaptureStillImageOutput?
+   var previewLayer : AVCaptureVideoPreviewLayer?
```

Figure 3: The components needed for an auto updating live camera feed.

The variables in figure 3 are used together with an @IBOutlet weak var cameraView: UIView! object that links to the camera view in the model view controller to set up the

camera image. These are the final functions needed to create the camera background. The code seen in figure 4 successfully implements the live camera feed as a background to the application if the camera is available.

```
        var backCamera = AVCaptureDevice.defaultDevice(withMediaType: AVMediaTypeVideo)

        var error : NSError?
-       var input = AVCaptureDeviceInput (device: backCamera, error: &error)
-
-       if error == nil && captureSession?.canAddInput(input) {
+       var input = AVCaptureDeviceInput()
+       do {
+           input = try AVCaptureDeviceInput(device: backCamera)
+       } catch {
+           //error
+       }
+       if error == nil && (captureSession?.canAddInput(input))! {
            captureSession?.addInput(input)
            stillImageOutput = AVCaptureStillImageOutput()
            stillImageOutput?.outputSettings = [AVVideoCodecKey : AVVideoCodecJPEG]

-           if captureSession?.canAddOutput(stillImageOutput) {
+           if (captureSession?.canAddOutput(stillImageOutput))! {
                captureSession?.addOutput(stillImageOutput)

                previewLayer = AVCaptureVideoPreviewLayer(session: captureSession)
                previewLayer?.videoGravity = AVLayerVideoGravityResizeAspect
                previewLayer?.connection.videoOrientation = AVCaptureVideoOrientation.portrait

-               cameraView.layer.addSublayer(previewLayer)
+               cameraView.layer.addSublayer(previewLayer!)
                captureSession?.startRunning()
            }
        }
```

Figure 4: This code within the ViewWillAppear function creates the live camera feed.

The corrections visible in figure 4 fixed compile time errors, however there were bad execution errors crashing the application at run time in the simulator. A hunch implied it was because of a lack of a hardware camera for the application to access when the code calls for it, causing a crash. Obtaining an iPod from the department to develop the app confirmed this

idea when the application was installed and ran flawlessly on the first try. The next step is to access the user's location so it can be used later to get data about local restaurants.

User Location

Looking into getting the device's current location was not nearly as complicated as setting the background as the camera. There were two libraries that needed to be imported into the project in order to access the GPS and device location coordinates. These were MapKit and CoreLocation. Additionally the ViewController class needed to be passed the CLLocationManagerDelegate to handle any calls to the location manager. Once those are in place, the locationManager function can be set up as seen in figure 5.

```
+ class ViewController: UIViewController, UIImagePickerControllerDelegate, UINavigationControllerDelegate,
  CLLocationManagerDelegate {
+
+     // Device Location services
+     let manager = CLLocationManager()
+
+     func locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {
+         let location = locations[0]
+
+         let myLocation:CLLocationCoordinate2D = CLLocationCoordinate2DMake(location.coordinate.latitude,
  location.coordinate.longitude)
+
+
+     }
```

Figure 5: Accessing location services.

After this function is defined, the lines depicted in figure 6 need to be added to the viewDidLoad function:

```
    override func viewDidLoad() {
        super.viewDidLoad()
-       // Do any additional setup after loading the view, typically from a nib.
+
+       manager.delegate = self
+       manager.desiredAccuracy = kCLLocationAccuracyBest
+       manager.requestWhenInUseAuthorization()
+       manager.startUpdatingLocation()
+
    }
```

Figure 6: These lines begin the process of getting and updating the user's location.

These work with the locationManager function defined in figure 5 to obtain and begin updating the user's latitude and longitude location. These coordinates can later be used to determine what restaurants are near the user.

Gyroscope Access

Another critical hardware component that needed to be accessed is the gyroscope in order to tell how the device is oriented. The first step to achieve this is to include the CoreMotion library. Next a motion manager of type CMMotionManager needs to be declared to handle calls to the motion hardware as seen in figure 7. At this point the CLLocationManager in figure 5 above was renamed to locationManager to more accurately represent its function and avoid confusion with the newly declared motionManager.

```
//gyroscope motion manager
var motionManager = CMMotionManager()



// Device Location services
let manager = CLLocationManager()
let locationManager = CLLocationManager()
```

Figure 7: Declaration of motion manager and renaming of location manager.

Once the motion manager is declared it is possible to access the gyroscopic data. To access the gyroscope data use the statement shown in figure 8 within the viewDidAppear function:

```
override func viewDidAppear(_ animated: Bool) {
    //begin collecting gyro data
    motionManager.startGyroUpdates(to: OperationQueue.current!) { (data, error) in
        if let mydata = data{
            //print(mydata.rotationRate)
        }
    }

    //camera setup
    super.viewDidAppear(animated)
    previewLayer?.frame = cameraView.bounds
}
```

Figure 8: Accessing the gyroscope data.

The commented line inside figure 8 allowed the verification that data was in fact being collected. As the project is continued this area will be used to access the data collected from the gyroscope. The aim is eventually to be able to tell if the device is being held at an upright angle.

Compass Access

In order to tell which direction the device is facing relative to any dining locations, the onboard magnetic compass can be accessed to give a directional heading. Accessing it is based off the location manager declared earlier. Initially there were many errors occurring that prevented the application from running after implementing the code that accessed the magnetic heading. After conducting additional research, it was revealed that an additional function needed to check if the magnetic heading was available before being able to begin updating it. This revelation resulted in the code seen in figure 9.

```
        if (CLLocationManager.headingAvailable()) {
            locationManager.headingFilter = 1 //compass
            locationManager.startUpdatingHeading()
        }
        else{
            print("no compass")
        }
```

Figure 9: Error catching for devices without a magnetic compass

Once this code was in place, a very serious problem presented itself. The iPod that I was issued does not possess a magnetic compass. This makes getting the application to accurately detect the user's directional heading much more difficult. Further research into the possibilities for this will have to be conducted after the final components are in place.

API Data Access

The most crucial part of this project is the ability to access data based on the location of the user and pull information about nearby restaurants from it. The initial idea was to use data from either Google maps, Apple maps, or Yelp in order to create the augmented reality images that will present the information to the user. During the process an API called Zomato was discovered that offered information about restaurants based off a latitude and longitude coordinate. Additionally it also offered an average rating and average cost, all of which is useful to present to the users within this application. Having never accessed an API before, research in this field took a significant amount of time. The information about the user's location is placed into variables declared outside the scope of the location manager seen in figure 10 so they can be accessed in other functions such as the API call in figure 11. In it's current form the code for the API call is occurring before the user's location is updated.

```
func locationManager(_ locationManager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {
    let location = locations[0]

    let myLocation:CLLocationCoordinate2D = CLLocationCoordinate2DMake(location.coordinate.latitude, location.coordinate.longitude)
    userLat = myLocation.latitude
    userLon = myLocation.longitude

    //location data print
    print (myLocation)


}
```

Figure 10: Attempting to collect data from the location manager

```
        let url = "https://developers.zomato.com/api/v2.1/geocode?
apikey=21a22086fa4c05e648be29aece327aea&lat=\(userLat)&lon=\(userLon)"
        let urlRequest = URL(string: url)

        URLSession.shared.dataTask(with: urlRequest!, completionHandler: {
            (data, response, error) in
            if (error != nil){
                print(error.debugDescription)
            }
            else{
                //handle api data
            }
```

Figure 11: Trying to make a call to the API using the user's current location

Further research determined that in order to obtain the data in time for the API call, the data had to be pulled directly from the location manager delegate immediately before the API call that used it as seen in figure 12:

```
    let userLat = locationManager.location?.coordinate.latitude
    let userLon = locationManager.location?.coordinate.longitude
    //API setup
    print(userLon, userLat)
    let url = "https://developers.zomato.com/api/v2.1/geocode?apikey=21a22086fa4c05e648be29aece327aea&lat=\(userLat)&lon=\(userLon)"
    let urlRequest = URL(string: url)

    URLSession.shared.dataTask(with: urlRequest!, completionHandler: {
        (data, response, error) in
        if (error != nil){
            print(error.debugDescription)
        }
        else{
            //handle api data
        }
    })
}
```

Figure 12: Accessing the user's location from the location manager delegate for the API call

The code in Figure 12 did not fulfill the necessary function that was required. Either the API call failed or the data was not being parsed. The "else" statement was never reached while at the same time no error was given. The code simply stopped in place and never continued past

that point. Research on information presented by an advisor helped to result in the following code seen in figure 13. This code correctly accesses the API and pulls location-based data to be parsed.

```
let userLat = locationManager.location?.coordinate.latitude
let userLon = locationManager.location?.coordinate.longitude

//API setup
guard let url = URL(string: "https://developers.zomato.com/api/v2.1/geocode?apikey=21a22086fa4c05e648be29aece327aea&lat=\
    (userLat)&lon=\(userLon)") else{
    print("ERROR: Invalid URL")
    return
}

let task = URLSession.shared.dataTask(with: url) {

    (data, response, error) -> Void in

    // URL request is complete
    guard let data = data else {
        print("ERROR: Unable to access content")
        return
    }


    do{
        guard let parsedData = try JSONSerialization.jsonObject(with: data, options: .allowFragments) as? NSDictionary else{
            print("ERROR: Unable to deserialize")
            return
        }

        print (userLat)
        print (userLon)
        print (parsedData)


    } catch{
        print("ERROR: unable to convert download")
        print(error)
        return
    }
}

task.resume()
```

Figure 13: Mostly functioning API call. User location is still not passed correctly.

The code present in figure 13 has one major flaw however. The way it is called is passing the incorrect latitude and longitude into the URL. They are being updated to the accurate user location after the formation of the URL string. Subsequently the API is being given the wrong information and is often returning data centered near Southeast Asia.

Researching into past information from the iOS class, a small detail that was previously overlooked came to light. The data for the user location was an optional type and had to be unwrapped while it was passed into the URL string seen in figure 14. By not unwrapping it, the latitude and longitude variables were passing in incorrect data.

```
        guard let url = URL(string: "https://developers.zomato.com/api/v2.1/geocode?
apikey=21a22086fa4c05e648be29aece327aea&lat=\(userLat)&lon=\(userLon)") else{
        guard let url = URL(string: "https://developers.zomato.com/api/v2.1/geocode?
apikey=21a22086fa4c05e648be29aece327aea&lat=\(userLat!)&lon=\(userLon!)") else{
```

Figure 14: Adding "!" to unwrap optional variables.

With this addition, the application successfully calls the API and gathers data about restaurants near the device's current location.

Future Research

Future research into properly rendering and scaling image size based on distance will be needed. The user's location will determine if the restaurant is close enough to render its information or just leave it out. This should be updating constantly as the user moves around. If a review is registered as close enough to appear, then the application will determine when it should appear on screen based off the gyroscope and magnetic heading. These components will tell if the device is facing the correct direction and being held at the correct angle. Figure 15 is a conceptual model of what the rendered review bubble may look like. Once the main view is completed, other features can be added to improve the application. If a bubble is tapped it can give more detailed information including the website etc. This information is already contained within the data retrieved from the API so no further calls to it would be needed. Eventually a system for users to leave their own ratings and reviews could be implemented as well.

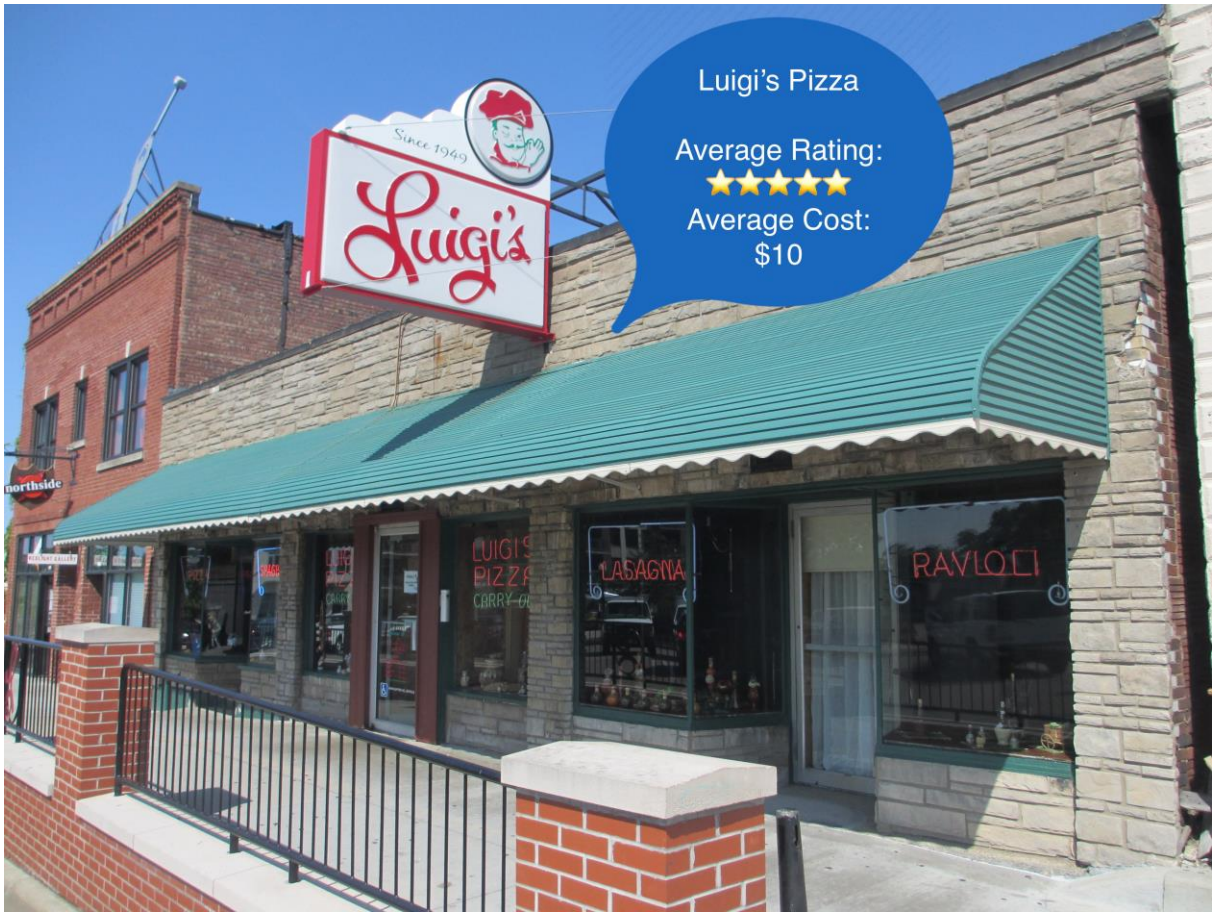

Figure 15: Conceptual model of Augmented Reality view.

Conclusion

The project "What's Good?" in its current form, possesses all the necessary information to create the output accurately. Using CLLocationManager the application has information about the user's location. Using this location information, it is able to generate a call to the Zomato API, which returns information about restaurants located near the user. Additionally the project has access to the device's magnetic compass when available, and is able to use the built in gyroscope and accelerometer to tell precisely how it is being held. These are all essential components that will give information in order to create the augmented reality graphics on the live camera feed that is set as the background of the application.

Works Cited

Elmer-DeWitt, P. (2016, February 11). About Apple's 40% Share of the U.S. Smartphone Market. Retrieved from http://fortune.com/2016/02/11/apple-iphone-ios-share/

Smith, C. (2017, January 21). 50 Amazing iPhone Statistics. Retrieved from http://expandedramblings.com/index.php/iphone-statistics/