Abstract

Hong Kong- one of the world’s fastest economies today, provides endless opportunities for retailers all around the globe. However, this vertical city has plummeting rates that sometimes create an insurmountable barrier for several retailers and hawkers to establish a legitimate shopping environment, thus driving them to other cities with lesser costs of rental and infrastructure. With the applications of Augmented Reality being on the rise, especially in the retail world, the vision to perceive products before buying them can be actualised. On one hand, while online shopping provides convenience, minimal setup costs and limitless inventory for retailers, physical stores provide an interactive and intuitive environment for customers to view products. This report explores the opportunity present in the two markets and proposes an all new hybrid shopping experience enabled by Augmented Reality technology and Photogrammetry. The vision would be manifested as an iOS based application that scans objects in 3D and allows convenient viewing of the products in AR. The project aims to create a working e-commerce platform within the time frame of ten months that serves both customers and retailers thereby allowing retailers to upload various images of the product onto the platform which would be spawned as 3D models for the customers to help enhance the shopping experience.
Acknowledgement

I would like to thank our project supervisor Dr HF Ting for his constant support and guidance through the course of this project. His valuable feedback and advice has guided us in the right direction. I would also like to thank the Faculty of Computer Science Engineering for providing us with the resources (iPad and the Mac Laboratory) needed for this project.
Abbreviations

- AR: Augmented Reality
- SDKs: Software Development Kits
- UI: User Interface
- VIO: Visual Inertial Odometry
- API: Application Programming Interface
# Table of Contents

List of Figures ............................................................................................................. 5  
List of Tables .................................................................................................................. 6  

1 Introduction .................................................................................................................. 7  
  1.1 Outline ..................................................................................................................... 7  
  1.2 What is Augmented Reality? .................................................................................... 7  
  1.3 Background .............................................................................................................. 8  
  1.4 Scope ......................................................................................................................... 8  
  1.5 Deliverables ............................................................................................................. 9  

2 Related Works .............................................................................................................. 9  
  2.1 Ikea ......................................................................................................................... 10  
  2.2 Scann3D ................................................................................................................... 11  

3 Project Features and Development Objectives .......................................................... 11  
  3.1 Model Creation ........................................................................................................ 11  
  3.2 Product Demos ....................................................................................................... 11  
  3.3 User Interface ......................................................................................................... 11  

4 Methodology ............................................................................................................... 12  
  4.1 Technologies used ................................................................................................... 12  
     4.1.1 ARKit SDK ......................................................................................................... 12  
     4.1.2 XCode 10 Integrated Development Environment ......................................... 13  
     4.1.3 ReCap API by Autodesk .................................................................................... 14  
  4.2 Application Design ................................................................................................ 15  
  4.3 System Design ....................................................................................................... 15  
     4.3.1 Customer Journey ............................................................................................. 15  
     4.3.2 Retailer Journey ............................................................................................... 16  
  4.4 Technical Implementation ...................................................................................... 17  
     4.4.1 Object Spawning ............................................................................................... 17  
     4.4.2 Creating the E-commerce Application ............................................................ 22  
     4.4.3 Photo to 3D Model Creation .......................................................................... 29  

5 Schedule ...................................................................................................................... 35  

6 Results and Discussion .............................................................................................. 35  
  6.1 Switching from Unity to Xcode .............................................................................. 35  
  6.2 Photo Processing Problem ...................................................................................... 36  
  6.3 User Testing: AR Spawning ..................................................................................... 37  

7 Limitations ................................................................................................................. 37  

8 Conclusion ................................................................................................................. 37  

9 References ................................................................................................................. 39
List of Figures

1 Figure 1: Difference between VR and AR............................................................ 8
2 Figure 2: Impact of AR on Retail........................................................................ 10
3 Figure 3: Ikea Place App.................................................................................... 10
4 Figure 4: Reality Capture API............................................................................. 14
5 Figure 5: Add New Product Form....................................................................... 17
6 Figure 6: Dice Spawned in midair and on a Detected Plane............................... 20
7 Figure 7: Tab Bar Controller of BazaAR Application........................................ 23
8 Figure 8: Navigation Controller of the BazaAR Application............................... 24
9 Figure 9: Collection View Controller................................................................. 24
10 Figure 10: Table View Controller and Page View Controller......................... 25
11 Figure 11: Entity Relationship Diagram......................................................... 27
12 Figure 12: Poor 3D Model Rendering............................................................... 36
13 Figure 13: Good Quality Rendering................................................................ 37
List of Tables

1 Table 1 Schedule ................................................................. 39
1. **Introduction**

Augmented Reality today is a rather new technology with several implications in the modern world. However, as it is still very fresh, not much is known about its potential and implications. This section aims to implore several aspects of the technology with respect to the project at hand.

1.1 **Outline**

The report discusses the current trends in Augmented Reality and points out our motivation to engage in this project. This is followed by a list of objectives we aim to achieve and a detailed methodology for executing the project. The report then explores the work we have done till now and the results obtained in the process. It will also emphasize on the results of the several rounds of iterations and testing and will also highlight the project’s limitations. The next steps to the project are discussed along with a project schedule in the final section.

1.2 **What is Augmented Reality?**

Back in 2016 when Pokemon Go was one of the most downloaded game in the Appstore, Augmented Reality garnered the attention of various technology enthusiasts. Augmented Reality is best described as a technology “which superimposes a computer-generated image on the user’s view of the real world”. Augmented Reality is often confused with Virtual Reality as the latter has been around for much longer whereas AR is relatively new in the technology world. The primary factor that sets the two apart (refer to figure 1) is that unlike Virtual Reality, Augmented Reality is not a fully immersive computer stimulated technology- instead it simply provides a composite view over the user’s real view. As seen in the image below, when using VR, the user’s environment is transformed into a new stimulated world by shutting out the real physical world while in AR the digital content is just superimposed onto the user’s view. This makes augmented reality more lifelike and intuitive.
1.3 **Background**

The current retail landscape can be broadly divided into two categories - offline shops and online stores. Online stores have become a norm in the past few years due to minimal setup costs and no inventory costs. However, the product viewing experience is not intuitive enough for buyers which often leads them to prefer Offline stores to make purchase decisions. Unfortunately for retailers, Offline stores still have infrastructure costs associated with them which makes them expensive to set up. Our project aims to bring the advantages of physical product viewing to online shopping using Augmented Reality and Photogrammetry. This hybrid shopping experience would enable users to interact with the products in virtually created environments that they can view from their phones.

1.4 **Scope**

The development of the retail application would be done on Apple’s ARKit2 platform thus the scope would be limited to iOS devices. The reason for choosing ARKit as our development platform has been discussed in the section 4.1.1. The application would be ready to deploy on the following ARKit Compatible devices.
- iPhone 6s and 6s Plus.
- iPhone 7 and 7 Plus.
- iPhone SE.
- iPad Pro (9.7, 10.5 or 12.9) – both first-gen and 2nd-gen.
- iPad (2017)
- iPhone 8 and 8 Plus.
- iPhone X, Xs and Xr

1.5 **Deliverables**

Our end deliverable would be an iOS based application that would be ready to deploy on the devices mentioned above. We would be using the ReCap API for photo to 3D model creation.

2. **Related Works**

Volkswagen was the first major retail company to incorporate AR in its application called “MARTA”. This application used augmented reality to help the company’s technicians with step by step automobile repair assistance by viewing various car parts in real time. Several retail giants today maintain that using AR in their applications has the potential to give them a competitive edge in the market. The data in figure 2 (as seen below) gives light to the impact and acceptance of AR in the retail market today. AR has been used in mobile shopping applications by various firms like IKEA and Shopify to add competitive edge to their consumer-experience (Sheehan, 2018). Photogammetry has been used in architecture, industrial design and game development industries. This technique of modelling uses photos to make measurements between objects and create a subsequent geometric representation of the same. Recently companies like Shopify have shown interest in this space by launching services for retailers to convert their products to 3D scans.

The following section discusses a few of these use-cases and also furthers our case for making an using the 2 technologies in our application.
2.1 Ikea

Ikea Place is an application that helps the user visualize the furniture in their home space before making a purchase decision. (Chang, 2018).” This is particularly important in the home decor retail space as customers often spend countless hours trying to decide whether something will look good in their homes or not. The “try before you buy” concept by Ikea has been very well received by its customers and serves as a motivation for us to build our application for at-home product viewing.

Figure 2: Impact of AR on Retail
Image retrieved from https://sensortower.com/blog/top-arkit-apps

Figure 3: IKEA’s Application - Place
2.2 Scann3D
This application requires the user to take approximately 20 to 30 photos of the object that they wish to 3D scan. This application available on iOS makes the device a standalone tool to turn images into 3D models for storing, sharing and editing in Sketchfab.

3. Features and Objectives
This section outlines the different features of the application that we aim on building in the project

3.1 Model creation
The retailer will be guided through a process to take photos of the object they want to sell. After the photosession is complete, our application would convert these photos to 3D models using PGM. These 3D objects can be used by the retailer to display the items he has for sale along with the pictures he/she took.

3.2 Product Demos
These items for sale inside the storefront would form the basis of our AR experience. The user would be able to interact with the objects in AR in a manner similar to physical products before making a purchase. Our project focuses on four types of products that can be viewed at homes - Jewellery, Furniture, Handicrafts and Artwork.

3.3 User Interface
To guide the user through different modes of the application, an easy to use and intuitive UI will be designed. While the overall layout of the application is based on existing e-commerce platforms. Several unique design features are introduced to emphasise the hybrid experience of the product.
4. Methodology

This section discusses the technologies used by the team to implement the project. It lays out the application design as well as the system design in detail.

4.1 Technologies used

4.1.1 ARKit SDK

In early 2017, Apple and Google announced their Augmented Reality SDKs - ARKit and ARCore respectively. These two software development kits act as the main platform for creating AR applications. For our final year project, the choice between the two was a challenging one. Both of them are quite alike on various important fronts like motion tracking, light estimation and environmental understanding. However, ARKit has some significant advantages-

- ARKit is found to be more accurate to distinguish between horizontal and vertical surfaces. As our application is triggered by tabletop detection, this serves as a quintessential factor in the SDK selection.
- ARKit has a higher exposure and reach owing to Apple’s economic structure. Having been around for longer, ARKit has already been used for developing a much wider range of applications than ARCore [4].
- ARkit based applications have been installed more than three million times globally since the launch of iOS 11 (Sensor Tower, 2017). The number of applications using ARKit help with the development process of our project by providing vital documentation.
- ARKit2 announced its latest feature of “shared experiences” which supports a collaborative environment that is much needed for a retail application.

ARKit is a Visual Inertial Odometry (VIO) system that tracks the user’s position by matching a point in the real world to a pixel on the camera sensor at each frame refresh. The Inertial Measurement Unit (IMU) also measures the user’s position using two inertial sensors - Gyroscope and accelerometer. The time measurements and accelerations are integrated backwards to calculate velocity and distance travelled between two frames. While error in such a system is upwards of 30% in
most systems, ARKit’s error correction reduces the error to single digit. Every tiny muscle movement can be detected by the inertial system providing unparalleled accuracy. The output of these systems is combined using a Kaiman Filter to provide the best estimate of the actual position which is published to the ARKit SDK.

4.1.2 XCode 10 Integrated Development Environment

Xcode is a very productive IDE and a major development tool for creating iOS applications. It is compatible with ARKit and acts as a very good framework for creating AR based applications. The E-Commerce application for this project is built using Xcode 10 which uses Swift version 4.2.1. The object spawning and AR integration is also done using Xcode. It is tightly integrated with cocoapods, an application level dependency manager for Swift, with which we installed several libraries like SwiftyJSON, Alamofire, BSIImagePicker etc. to help with our app development.

- **SwiftyJSON** is an open source library that provides a user friendly way to deal with the JSON data. Swift uses a very tedious method for JSON serialization which is performed this way-

```swift
if let jsonObject = try JSONSerialization.jsonObject(with: data, options: .allowFragments) as? [[String: Any]],
    let username = (jsonObject?["user"] as? [String: Any])?["name"] as? String
```

In our application, we have to rely heavily on JSON objects which are received from the Recap API HTTP server. Using the standard approach in swift makes the code very messy and tedious. The SwiftyJSON library abstracts a large part of this code and makes it possible to declare JSON objects implicitly and allows access in a manner similar to an ArrayObject, in this way-

```swift
let json = JSON(data: dataFromNetworking)
if let userName = json?["user"]?["name"].string {
    //Now you got your value
}
```

- **Alamofire**

In standard Swift, NSURLConnection and a Foundational URL Loading system are used to manage request-response cycles to HTTP Servers. Alamofire provides an easy to use interface that performs HTTP
Networking asynchronously without having to write large chunks of the protocol manually.

- **BSImagePicker**

  As Swift’s UIImagePickerController Class only enables the selection of one image from the user’s photo library. To create the 3D models, multiple images have to be uploaded by the shopkeeper. The BSImagePicker framework provides functions for creating an Image Picker and converts the selected assets to JPEG objects ready for upload.

4.1.3 **ReCAP API by Autodesk**

ReCap Photo is an Autodesk 360 service that creates 3D Scans from photos such that users can view and share the 3D data. Users upload images on Autodesk 360 and the Cloud-based service creates a 3D mesh model reading for importing. The API works on REST Calls and allows developers to integrate this service for into their applications and products. The figure below gives a brief overview about how the service works.

![Reality Capture API](image)

Figure 4: Reality Capture API
4.2 **Application Design**

Our application, BazAR, is mainly designed to cater two users - Shopkeepers and customers. The application has different use cases for the two end users.

When the application is loaded, the various shopfronts and featured products are displayed on the home screen. There is an add button for the shopkeepers to upload their products and create a new product/shopfront category. When the shopkeeper clicks on the button, he/she can select images from their device’s library and upload multiple images. For better 3D model generation, it is advised to upload around 50 pictures of the product. Once the shopkeeper uploads the pictures, a 3D model of the product will be generated and featured on the application for the customers to view.

The Customer would be able to use the application just as he/she would use existing e-commerce applications like Amazon, Taobao etc. On the application’s home screen, the customer would be able to view the shopfronts and select a shop to visit. Once they select the browse option, they can navigate through a detailed product list and choose to view the particular product in AR. When the user views a specific item, a button will pop up on the screen to view the model of the object. On clicking, the user will be able to point the camera at a surface and the object will appear in 3D for interactive viewing.

4.3 **System Design**

4.3.1 **Customer Journey**

Step 1: The user starts the application
Step 2: User browses through the list of shops which appear from the data entries
Step 3: User selects and enters the particular shopfront, and views the product list
Step 4: User selects a product from the list
Step 5: The respective 3D model is called from the database and the camera is started again
Step 6: Repeat steps 1,2,3,4,5 till user exits.
4.3.2 **Retailer Journey**

When the retailer starts the application, the home screen has a button called add new product. This pulls up a form (as seen in the figure below) for the retailer to enter the product details and select images to upload to the server for them to be converted into a 3D model of the retailer’s product. When the retailer clicks on the create 3D model button, the following calls are made to the ReCap API for processing the mesh:

*Part of PhotoToMesh()*

- **POST photoscene**
  - This creates a Photoscene that will perform calibration and meshing.

- **POST file**
  - The reference for each image file is provided in this call for uploading to ReCAP360.

- **POST photoscene/:photosceneid**
  - Initiates processing of the Photoscene created above.

*Part of GetModelStatus()*

- **GET photoscene/:photosceneid/progress**
  - This returns the current processing progress percentage and status of the Photoscene.

- **GET photoscene/:photosceneid**
  - Returns a time-limited HTTPS link to the .obj output file that can be downloaded for display as 3D object.
4.4 Technical Implementation

4.4.1 Object Spawning

The basis for our augmented reality is object spawning. This was achieved using the ARKit2 Software Development Kit in the Xcode IDE. ARKit2 provides a way of implementing augmented reality by creating and tracking the correspondence between the real-world space and the virtual space where you can model/spawn the visual content. The SDK makes use of the A9-A10 chip which are available in all Apple devices that support iOS 11 and above.

When the user selects a product from the product list, the selected product is spawned as a game object on the AR Scene. ARKit’s WorldTrackingConfiguration provides the basis for this as it helps to map and track the user’s real-world space. It allows you to create (i.e spawn) the game objects as a virtual content on the user’s rear camera view. The WorldTracking Scene is where the virtual object is spawned. This configuration makes the object very lifelike by increasing its size as you move towards it with the device in hand.
The first step for spawning was to create a new 3D SceneKit in Xcode. Scene Kit combines a high-performance rendering engine with a descriptive API for import, manipulation and rendering of 3D assets. It requires only descriptions of the scene’s contents and actions or the animations that are desired by the user to perform. It autogenerates an ARSCNView in the main storyboard of the Xcode project which is where the object is spawned. The scene in the ARSCNView consists of various scene nodes called SCNNode which act like a structural element of the scene graph. To this node various elements like geometry, cameras, lights etc. can be attached.

The other major component of the SceneKit that is used during spawning is the SCN Scene Renderer. This component is a renderer that manages the Scene Kit’s rendering and animation of the scene’s contents. The code snippet below automatically adds light to the scene which is ambient and adjusting to the camera’s environment.

```swift
super.viewDidLoad()
self.sceneView.debugOptions = [ARSCNDebugOptions.showFeaturePoints]
sceneView.delegate = self

sceneView.autoEnablesDefaultLighting = true
```

The spawning of the object is triggered on user touch. To do so the “touches” component of the scene kit is used. These touches are instances of the UITouch class which represent the starting phase of the user touch event. The touches began function, as shown in the code snippet below, is called by the Scene Kit when a new user touch is detected on the iPad screen. For the purpose of spawning, this function has been overridden to customize its on touch event. As per the code snippet, first the touch location is detected and saved locally. This is in the form of a 2D space with only x and y components. As the object is to be spawned on a 3D plane, the hitTest function converts the touch location coordinates in a way that it hits a point in the 3D space.

```swift
override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
    if let touch = touches.first {
        let touchLocation = touch.location(in: sceneView)
        let results = sceneView.hitTest(touchLocation, types: .existingPlaneUsingExtent) //converting 2d to 3d touch has x and y component, so its a 2D location, hit a point in 3D space
    }
```
The result of the hitTest function can be understood more clearly through the figure below. On touch detection the coordinates of the touch component were - x: -0.026806; z: -0.087583. This shows that the touch was in a 2D plane and hence was missing a Y component. However, after the function was called, the 2D coordinates were mapped to a 3D plane as shown in the highlighted blue section of the snippet below.

The last component of the scene kit that plays a major role in object spawning is plane detection. Initially, the object was spawned in thin air at a random location. However, plane detection was added to make the user able to visualize the object on a table object for better understanding and interaction. The two images show the difference in the spawning of a dice before and after the addition of plane detection.
To achieve this the plane detection component of the world tracking configuration was set as horizontal. By doing so, the session added ARPlane Anchor objects and notified the ARSCNView if the camera captured real world video images detected a horizontal flat surface or not. These flat surfaces are detected by the device’s rear camera and in our application’s, case are defined by a blue grid to prompt the user to an ideal horizontal surface to spawn the object on. The plane anchors give information about the estimated position and shape of the horizontal surface. Feature points (the yellow dots in the figure above) were added in the debugging options to help give the virtual object a place to anchor on and helping them get a sense of where to remain in case the user moves their device around. The function below was added to first check if the anchors that were added were used for plane detection or not and then places the plane on the scene view added later by the renderer mentioned above.

Figure 6: Dice spawned in midair and on a detected plane
After all these components are rightly set up, the object is finally spawned onto a plane for user interaction by adding the .usdz file as a scene and appending it to the main scene view. The coordinates of the object are obtained from the UI touch coordinates which successfully spawns the object exactly where the user desired it to be placed (as long as it is on a horizontal surface). The function below shows the appending of the “bagNode” as a scene in the scene view using the hit result’s coordinates.

```swift
func renderer(.renderer: SCNSceneRenderer, didAdd node: SCNNode, for anchor: ARAnchor) {
    if anchor is ARPlaneAnchor {
        let planeAnchor = anchor as! ARPlaneAnchor
        let plane = SCNPlane(width: CGFloat(planeAnchor.extent.x), height: CGFloat(planeAnchor.extent.z))

        let planeNode = SCNNode()
        planeNode.position = SCNVector3(x: planeAnchor.center.x, y: 0, z: planeAnchor.center.z)

        planeNode.transform = SCNMatrix4MakeRotation(-Float.pi/2, 1, 0, 0)

        let gridMaterial = SCNMaterial()
        gridMaterial.diffuse.contents = UIImage(named: "art.scnassets/grid.png")
        plane.materials = [gridMaterial]
        planeNode.geometry = plane
        node.addChildNode(planeNode)
    } else {

        if let bagNode=scene.rootNode.childNode(withName: '"Dice', recursively: true) {
            bagNode.position=SCNVector3(x: hitResult.worldTransform.columns.3.x, y: hitResult.worldTransform.columns.3.y + bagNode.boundingSphere.radius , z: hitResult.worldTransform.columns.3.z)
            sceneView.scene.rootNode.addChildNode(bagNode)
        }
    }
}
```
4.4.2 Creating the E-commerce application

The application, BazaAR, is designed to be user friendly like most other e-commerce applications available today. The application serves as a platform between shopkeepers and customers, allowing shopkeepers to upload new products, and being viewed/purchased by the customers. The development of this application was done using the Xcode 10 IDE.

To do the development we followed the Model View Controller architectural pattern. This pattern was selected to separate the application’s various sections and concerns. It helps in code organization as each section of the code has a different purpose. The MVC model also helps with the use of various dependency managers like cocoapods as they are also built using the MVC pattern.

The designing of the user interface is done using the Main Storyboard. The main storyboard provides a visual representation of the application’s UI by showing various screens of content and the connections between them. It is made up of a sequence of scene which each have a view controller and its respective views. The scenes in the storyboard are connected using segues which act as a transition between the view controllers.

The storyboard in our application contains view controllers for the following:

1. **Navigation Controller**: This is a container view controller that defines a stack based scheme for navigation hierarchical content. In this view interface, only one child view controller is visible at a time. It makes use of an ordered array called the navigation stack. It manages the navigation bar at the top and the toolbar at the bottom of the interface. The purpose of the navigation controller is to essentially embed the content of other view controllers inside of itself.
2. **Page View Controller:** This view controller manages the navigation between pages of content wherein each page is managed by a child view controller. Page view controllers use the user specified transition to animate the changes.

3. **Tab Bar Controller:** This view controller manages a radio-style selection interface in which on selection the display of a particular child view controller is determined. It acts as an interface that is used to display tabs at the bottom of the page window to navigate between different modes and for displaying the views for that mode. Each tab of the tab bar controller interface is linked with a particular custom view controller. Thus, when a user selects a specific tab, the tab bar controller displays the root view of the corresponding view controller and replaces any previous views.

4. **Collection View Controller:** It is a view controller that manages an ordered collection of data items and presents them using customizable layouts. It presents the contents using a cell which is an instance of the UI Collection View Cell class.

5. **Table View Controller:** It is a view controller that is required to manage a table view. In our application it helps to manage the product list in the browse product view. This is also appended with a scroll view for easy scrolling in case of multiple products spanning more than one-page view.

---

**Figure 7:** Tab Bar Controller of the BazaAR Application.

This controller has two tabs for user view- Home and Browse which is displayed on user selection.
Figure 8: Navigation Controller of the BazaAR Application.
Contains a navigation stack for the home screen which has multiple UI views and Collection Views.

Figure 9: Collection View Controller
This has multiple cells to display the shopfronts and the products. The image is rendered using UI View as a part of the collection view cell.
We will now discuss how the database was populated and parsed for the application.

1. **Data Models**

To develop the application’s model layer, Core Data was used. It provides a set of classes that corroboratively support the data models. It also persists or caches data and supports undo on the deployment device. It makes data saving easy without having to administer the database directly by abstracting the details of mapping the development objects to a store.

The next major component required for data modelling is the Core Data Stack. It comprises of one or more managed object contexts which are connected to a single persistent store coordinator which are then connected to one or more persistent
stores. This stack contains all the components of core data which are required to create, manipulate and fetch managed objects. Each core data stack has one persistent store coordinator. To use the core data stack efficiently, we make use of the NS Persistent Container Class. It handles the creation of the managed object model, persistent store coordinator and the managed object context thereby simplifying the creation and management of the core data stack.

For the purpose of our application, we created 4 main data objects(entities):

1. **Product**: This data entity stores the product details like id, name, sale price, description etc.
2. **Manufacturer**: This data entity stores the details of the manufacturer like id and name.
3. **Product Image**: This stores the images associated with a product which will be used to create the 3D models.
4. **Product Info**: This data entity stores the detailed information of a product which will be displayed on the product detail page of the application.

The class diagram below gives a clear picture of the data models and their relationship with one another.
2. **Data Parsing**

To display the product and its information on the screen, the load products function is called in the AppDelegate class. This function performs data extraction and JSON serialization to get the data from the database and display it in the respective container view cell. The Application Delegate class consists of a set of methods that are called in response to important application life cycle event. For the purpose of our application, the App Delegate class performed the following functions:

- It holds the application’s startup code.
- It instantiates and calls the core data stack.
- It stores the application’s central data objects which may not have their own view controllers.
- It responds to transitions in the applications and maintains the state of the application.
• Resets the database when necessary by using NS Batch Delete Request
• Loads products from the database, by performing necessary serializations and then displaying it in the correct container view cell as shown in the code snippet below.

```swift
private func loadProducts() {
    let managedObjectContext = CoreDataStack.persistentContext
    let url = Bundle.main.url(forResource: "products", withExtension: "json")
    if let url = url {
        let data = try? Data(contentsOf: url)
        do {
            guard let data = data else {
                return
            }
        }
    }
    let jsonResult = try? JSONSerialization.jsonObject(with: data, options: .mutableContainers) as! [String: Any]
    let jsonArray = jsonResult.value(forKey: "products") as! [String: Any]

    for json in jsonArray {
        let productData = json as! [String: Any]
        guard let productId = productData["id"] else { return }
        guard let name = productData["name"] else { return }
        guard let type = productData["type"] else { return }

        let product = Product(context: managedObjectContext)
        product.id = productId as? String
        product.name = name as? String
        product.type = type as? String

        if let regularPrice = productData["regularPrice"] {
            product.regularPrice = (regularPrice as? Double)!
        }
    }
}
```

3. **Data Updating**

As of now the data is stored in a JSON file which is created locally so that it can portray how the application would act when it is tied to a real database.

Every time the user adds a new product to submit, a function is called that creates a JSON object from the text input and the fetched URLs of the 3D model.
4.4.3 Photo to 3D Model Creation

When the user clicks on the create 3D model button in the pop up form, the 3D model creation process is triggered. The create model function associated with the button’s on click event runs the request response cycles from authentication to 3D model creation using the steps described below:

- Step 1: Authentication
When the application is registered with the Recap service, a client ID and a client Secret are received that are used to authenticate the application and then generate a unique access token for the next calls.

```
@IBAction func createmodel(_ sender: Any) {

    //Authorization

    let headers: HTTPHeaders = [
        "Content-Type": "application/x-www-form-urlencoded"
    ]

    let params = [  
        "client_id": "bkqNH3g7XHkU4mOQMA6rPUOvayYT",  
        "client_secret": "eUdxR281u88Snv",  
        "grant_type": "client_credentials",  
        "scope": "data:read data:write"
    ]

        if((responseData.result.value) != nil) {
            let swiftyjsonvar = JSON(responseData.result.value)

            {
                "token_type": "Bearer",  
                "expires_in": 1799,  
                "access_token": "eyJhbGciO1IJUIxI1NSIzmpZCI6Imp3dF9zeWltZXRXaWNfa2V5X2RldiJ9.eyJjYjGlbnRfaWQ0IjIjWTFqcm1rQXhpSmVtNmVoV1JSMG1JME15MTU4NzU2NSwic2NvcmV0aW9uIjoiMTBcInZlcml0ZU9TQ0FUS29tZCI6IiIsImFjZmVpYnkaIjoiZTVmNzQ1N2Y1MDZkNTUyZjA5ZmQ0ZjMwM2E1MDIyZSIsInZlcnNpb24iOiJyZWFkIn0"
            }
        }
    }

    // Step 2: Create PhotoScene
    // The next call is nested in the response of the previous call. The access token is appended in the form of a header in the new call. The new call requests creation of a unique photosceneid in which our photos will be uploaded.
    Request
```
Step 3: Upload Files

The photoscene id is sent in the form of a parameter to the new call and the user’s image gallery is opened from which the user selects the photos he/she wishes to upload using the BSImage Picker. These photos are added iteratively as a part of the POST call and then uploaded to the Recap API’s server. The progress shows the status of the upload.

Request
// create an instance
let vc = BSImagePickerController()

// display picture gallery
self.bs_presentImagePickerController(vc, animated: true,
                                      select: { (asset: PHAsset) -> Void in
                                              // User deselected an asset.
                                          },
                                      deselect: { (asset: PHAsset) -> Void in
                                                  // User cancelled. And this where the assets currently selected.
                                          },
                                      cancel: { (assets: [PHAsset]) -> Void in
                                                               // User finished with these assets.
                                                 },
                                      finish: { (assets: [PHAsset]) -> Void in
                                                               // User finished with these assets.
                                                 }
                                      }
  
  Alamofire.upload(multipartFormData: { (multipartFormData) in
    for (key, value) in params3 {
      multipartFormData.append(((value).data(using: String.Encoding.utf8)!,
                                withName: key as String)
    }
  }
  for i in 0..<assets.count
  {
   autoreleasepool {

      let manager = PHImageManager.default() 
      let option = PHImageRequestOptions() 
      var thumbnail = UIImage() 
      option.isSynchronous = true 

      manager.requestImage(for: assets[i], targetSize: 
                           PHImageManagerMaximumSize, contentMode: .aspectFill, options: option, 
                           resultHandler: { (result, info) -> Void in
                                         thumbnail = result!
                           })
  }}
Step 4: Retrieve progress of model creation

Once the upload is done successfully, a get call is iteratively made to check the progress of the model creation. The loop stops once the progress is 100%.

Request
Step 5: Download the 3D model

Once the 3D model is created successfully, a request is sent to download the image and the fetched URL is stored in the database.

```swift
var checkprog: Double = 0.0
let group = DispatchGroup()

while (checkprog < 100) {
        if (responseData.result.isSuccess) {

            let newjsonvar = JSON(response.new.result.value)

            print(newjsonvar["Photoscene"]
                ["progress"].doubleValue.description)

            group.leave()

            checkprog = newjsonvar["Photoscene"]!"progress".doubleValue

            self.viewDidLoadModel.setProgress(Float(checkprog), animated: true)

            self.progressLabelModel.text = "\(Int(checkprog)) %"
        }
    }
}
```

Response

```json
{
    "Photoscene": {
        "photosceneid": "AtAuFsedTdqWdhF9VzHepp5oM9PIITuzI4xdMbz",
        "progressmsg": "DONE",
        "progress": "100"
    }
}
```

```swift
Alamofire.request("https://developer.api.autodesk.com/photo-to-3d/v1/photoscene/" + photosceneid, method: .get, headers: headers2).responseJSON { (responseData) -> Void in
    if (responseData.result.isSuccess) {

        // Process response data
    }
}
```
5. **Schedule**

<table>
<thead>
<tr>
<th>Timeframe</th>
<th>Task</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>September-October 2018</td>
<td>Feasibility Study</td>
<td>Completed</td>
</tr>
<tr>
<td>November-December 2018</td>
<td>Functionality Testing</td>
<td>Completed</td>
</tr>
<tr>
<td>January-February 2019</td>
<td>System Design and Backend Development</td>
<td>Completed</td>
</tr>
<tr>
<td>March-April 2019</td>
<td>UI and Testing</td>
<td>Completed</td>
</tr>
</tbody>
</table>

6. **Results and Discussions**

6.1 **Switching from Unity to Xcode**

The first phase of our development was performed using the Unity game engine. However, as we proceeded, we faced various issues with the camera management and application development. As Unity is a game engine, it was highly restrictive owing to the UI class. Moreover, every game object has to be wrapped in C# scripts for further development making it very tough for us to implement the database logic and run asynchronous tasks. Therefore, we had to pivot from the Unity plan and switched to Xcode as it very compatible with the MVC pattern and supports major mobile development functions.
6.2 Photo processing problem

As we were very new to the Recap API framework, we encountered several issues like internal server errors and data processing errors. When we first finally managed to create the 3D model, it was of very poor quality as seen in the figure below.

![Poor 3D model rendering](image)

Figure 12: Poor 3D model rendering

We tried several ways to debug the issues and realized that the quality of the final 3D model depended on the number and quality of the photos uploaded to the server. We took better photos which were in focus, and from a wide range of angles. When we uploaded a total of 70 pictures, the 3D model’s quality increased significantly. However, the processing time increased too.
6.3 User Testing: AR Spawning

We ran several tests to measure quality of the spawned objects and asked for feedback from different user groups. At first the object appeared dull and could barely be seen in certain environments. To fix this, we enabled the environment’s default lighting on the spawned object so as to always make it visible to the user.

```
sceneView.autoEnablesDefaultLighting = true
```

7. Limitations

Our application has a few limitations. As the development was done using Xcode and ARKit, the deployment targets are restricted to Apple devices only. Moreover, to generate good quality 3D models, the shopkeepers have to upload a large number of photos which could be a little time consuming. The greater the number of photos, the greater is the processing speed (can take up to 40 minutes).

8. Conclusion

With companies like Apple and Google developing dedicated software bundles for AR, developers are creating world-class AR applications. Our aim to bridge the gap between physical and online stores is a humble attempt at building a strong use case for AR for mobile devices. The development of this application was motivated by the space and monetary constraints in today’s retail environment. As the world gets more and more tech savvy, the room and acceptance for the notion of an augmented/hybrid market increases significantly.
A deep study into ARKit’s developer guide and functionalities helps solidify the progress of this project. After the successful testing of sample 3D models, our aim to create an all new shopping experience which combines the advantages of online and physical retail stores was achieved. However, the scope of this application is limited to ARKit2 compatible devices only and doesn’t account for the payment gateways. The shops are currently limited to displaying only a few types of merchandises. Our project isn’t just a one-off application but a new kind of user experience that uses visually imposed images as interaction points. We believe that the application has the potential of changing the retail industry and how people perceive AR applications.
References


