ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΜΕΤΑΔΟΣΗΣ
ΠΛΗΡΟΦΟΡΙΑΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΛΙΚΩΝ

Εξέλιξη και Ενίσχυση ιδιοτήτων Πλατφόρμας Επαυξημένης Πραγματικότητας για Ευφυείς Εφαρμογές Κινητού Υπολογισμού

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Λεωνίδας Μαυρωτάς

Επιβλέπων : Ιάκωβος Βενιέρης
Καθηγητής Ε.Μ.Π

Αθήνα, Σεπτέμβριος 2021

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΜΕΤΑΔΟΣΗΣ
ΠΛΗΡΟΦΟΡΙΑΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΛΙΚΩΝ

Εξέλιξη και Ενίσχυση ιδιοτήτων Πλατφόρμας Επαυξημένης Πραγματικότητας για Ευφυείς Εφαρμογές Κινητού Υπολογισμού

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Λεωνίδας Μαυρωτάς

Επιβλέπων : Ιάκωβος Βενιέρης
Καθηγητής Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 22η Σεπτεμβρίου 2021.

. . . . .	. . . . .	. . . . .
Βενιέρης Ιάκωβος Καθηγητής Ε.Μ.Π.	Κακλαμάνη Ι. Δήμητρα-Θεοδώρα Καθηγήτρια Ε.Μ.Π.	Ματσόπουλος Γεώργιος Καθηγητής Ε.Μ.Π.

Αθήνα, Σεπτέμβριος 2021

.....

Λεωνίδας Μαυρωτάς

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π

συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

Περίληψη

Οι έξυπνες συσκευές και συγκεκριμένα τα έξυπνα τηλέφωνα (smartphones), αποτελούν πλέον αναπόσπαστο κομμάτι της καθημερινότητας μας. Σε αυτά, προσφέρεται μια πληθώρα εφαρμογών, που μας βοηθούν σε πολλές πτυχές της ζωής μας. Η επαυξημένη πραγματικότητα δεν είναι μία καινούργια ιδέα, παρόλα αυτά δεν ήταν προσιτή για καθημερινή χρήση μέχρι πρόσφατα. Την τελευταία δεκαετία η επεξεργαστική ισχύς των έξυπνων συσκευών αυξήθηκε ραγδαία, με αποτέλεσμα η δημιουργία εφαρμογών επαυξημένης πραγματικότητας να είναι πλέον εφικτή.

Σκοπός της διπλωματικής εργασίας είναι αρχικά η εξοικείωση και η διερεύνηση των δυνατοτήτων του εργαλείου ARKit της Apple και στη συνέχεια η σχεδίαση και η ανάπτυξη μίας εφαρμογής με χρήση αυτού του εργαλείου. Η εφαρμογή αυτή θα αφορά την τρισδιάστατη αναπαράσταση μίας άσκησης φυσικής, συγκεκριμένα του νόμου του Coulomb, ώστε να αναδείξουμε το ρόλο που μπορούν να έχουν τέτοιες εφαρμογές στην εκπαίδευση. Ο χρήστης έχει τη δυνατότητα να δημιουργεί τοπολογίες ασκήσεων από το μηδέν, καθώς και να τις επεξεργάζεται, να τις αποθηκεύει ή να τις διαγράφει. Η αλληλεπίδραση με την τοπολογία πραγματοποιείται με απλές χειρονομίες του χρήστη, καθώς η εφαρμογή τον καθοδηγεί στις διάφορες ενέργειες που μπορεί να εκτελέσει. Η αναπαράσταση των φορτίων, των δυνάμεων και άλλων μεγεθών γίνεται με τρισδιάστατα μοντέλα, τα οποία δημιουργήσαμε μέσω της διεπαφής που προσφέρει το ARKit.

Η εφαρμογή προορίζεται για τις συσκευές που χρησιμοποιούν το λειτουργικό σύστημα iOS, και συγκεκριμένα οι συσκευές iPhone και iPad. Η ανάπτυξη της εφαρμογής έγινε με τη βοήθεια ενός υπολογιστή της Apple Macbook Pro, σε περιβάλλον Mac OS X 11.2.1. Για τη δημιουργία της ήταν απαραίτητη η συγγραφή του κώδικα σε γλώσσα Swift, καθώς και η δημιουργία των τρισδιάστατων μοντέλων που θα χρησιμοποιηθούν. Τα μοντέλα αυτά είναι τα φορτία, ως σφαιρικά μοντέλα, τα διανύσματα δυνάμεων των φορτίων ως βέλη, οι αποστάσεις μεταξύ των φορτίων ως διακεκομμένα ευθύγραμμα τμήματα και τέλος, οι επιγραφές των τιμών των διαφόρων μεγεθών στην τοπολογία. Επίσης για τη δοκιμή της εφαρμογής χρησιμοποιήθηκαν δύο iOS συσκευές, ένα iPhone 8 Plus και ένα iPhone SE 2020, τα οποία χρησιμοποιούν την 14η έκδοση του λειτουργικού συστήματος, το iOS 14. Τέλος, για την ανάπτυξη της εφαρμογής, ήταν απαραίτητη η εγγραφή ως μέλος στο πρόγραμμα ανάπτυξης εφαρμογών για iOS της Apple (iOS Developer Program).

Λέξεις κλειδιά

επαυξημένη πραγματικότητα, έξυπνες συσκευές, ARKit, RealityKit, ανίχνευση επιφανειών, ιχνηλάτηση, αισθητήρες, εφαρμογή, τρισδιάστατα μοντέλα

Abstract

Smart devices, and in particular smartphones, are now an integral part of our daily lives. In them, a variety of applications are offered that help us in many aspects of our lives. Augmented reality is not a new concept, however it was not available for everyday use until recently. The ever-increasing processing power of smart devices now allows the creation of augmented reality applications that users can use for their everyday activities.

The purpose of this thesis is first to familiarize and explore the capabilities of Apple's ARKit framework and then to design and develop an application using it. This application will concern the three-dimensional representation of a physics exercise, specifically Coulomb's law, in order to highlight the role that such applications can play in education. The user is able to create topologies from scratch, as well as edit, save or delete them. The interaction with the topology is done with simple gestures of the user, as the application guides him in the various actions he can perform. The representation of loads, forces and other quantities is done with three-dimensional models, which we created through the interface offered by ARKit.

The application is intended for devices using the iOS operating system, the iPhone and the iPad. It was developed with the help of an Apple Macbook Pro computer, using the Mac OS X 11.2.1 environment. To create the application, he had to write our own code, written in Swift language, as well as to create the 3D models that will be used. These models are the points of charge, represented by spherical models, the force vectors of these points, represented by arrows models, the distances between the points as dashed straight lines and finally, the inscriptions of the values that all the previous have. Two iOS devices were also used to test the application, an iPhone 8 Plus and an iPhone SE 2020, which both use the 14th version of the operating system, iOS 14. Finally, in order to develop the application, it was necessary to register as a member in Apple's iOS Developer Program.

Keywords

augmented reality, smart devices, ARKit, RealityKit, plane detection, tracking, sensors, application, 3D models

Ευχαριστίες

Με την εκπόνηση της διπλωματικής μου εργασίας καθώς και την ολοκλήρωση των σπουδών μου στο τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Ηλεκτρονικών Υπολογιστών του Ε.Μ.Π. θα ήθελα να ευχαριστήσω όλους τους ανθρώπους που με βοήθησαν να φτάσω τους στόχους μου.

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή αυτής της διπλωματικής εργασίας, κύριο Βενιέρη Ιάκωβο, για την ευκαιρία που μου έδωσε να ασχοληθώ με ένα αντικείμενο το οποίο λαμβάνει όλο και περισσότερο ενδιαφέρον στην παγκόσμια κοινότητα καθώς και την αμέριστη εμπιστοσύνη που μου έδειξε.

Ένα ακόμα ευχαριστώ πηγαίνει στον Υποψήφιο Διδάκτορα Πέτρο-Φλώριο Μπάκαλο που με καθοδήγησε στην ανάπτυξη της εφαρμογής της διπλωματικής εργασίας, καθώς και στους καθηγητές που είχα ανά τα χρόνια. Επιπλέον, ευχαριστώ όλους τους προπονητές μου, που σεβάστηκαν τους στόχους μου, μου επέτρεψαν να είμαι συνεπής στη σχολή και με βοήθησαν να διατηρήσω μία υγιή φοιτητική πορεία.

Ακόμα, θα ήθελα να ευχαριστήσω τους φίλους μου, ορισμένοι συμφοιτητές, άλλοι πρώην συμμαθητές και άλλοι συμπαίκτες μου, για την κατανόηση που έδειχναν αλλά και τη στήριξη που μου παρείχαν κατά τη διάρκεια των σπουδών μου. Επιπλέον ευχαριστώ ορισμένα ιδιαίτερα άτομα της ζωής μου που στάθηκαν δίπλα μου σε δύσκολες και απαιτητικές στιγμές κατά τη φοίτησή μου θυσιάζοντας το χρόνο τους, γνωρίζουν τα ίδια ποια είναι. Ιδιαιτέρως θα ήθελα να ευχαριστήσω τους συμφοιτητές και πολύ καλούς μου πλέον φίλους, Σταύρο, Χάρη και Αποστόλη, τους οποίους έμελλε να γνωρίσω στα αμφιθέατρα της σχολής και δώσανε μία πιο ευχάριστη νότα στα φοιτητικά μου χρόνια. Τους ευχαριστώ για όσα περάσαμε μαζί και για όσα θα θυμόμαστε.

Τέλος, ένα μεγάλο ευχαριστώ πηγαίνει στους γονείς μου, Φανή και Γιώρο, οι οποίοι αποτελούσαν την έμπνευση μου κατά τη διάρκεια των σπουδών μου, παροτρύνοντάς με ήσυχα με το παράδειγμα τους να κυνηγάω τους στόχους μου αλλά και να δέχομαι τα αποτελέσματα των κόπων μου, όποια κι αν είναι αυτά. Ευχαριστώ επίσης και την υπόλοιπη οικογένεια μου που με στήριξε στη φοιτητική μου σταδιοδρομία και ιδιαίτερα τον αδερφό μου, Άρη, ο οποίος αν και μικρότερος και χωρίς ίσως να το ξέρει αποτελεί πρότυπο στη ζωή μου.

Πίνακας περιεχομένων

Κατάλογος σχημάτων

Κατάλογος εικόνων

1. Εισαγωγή

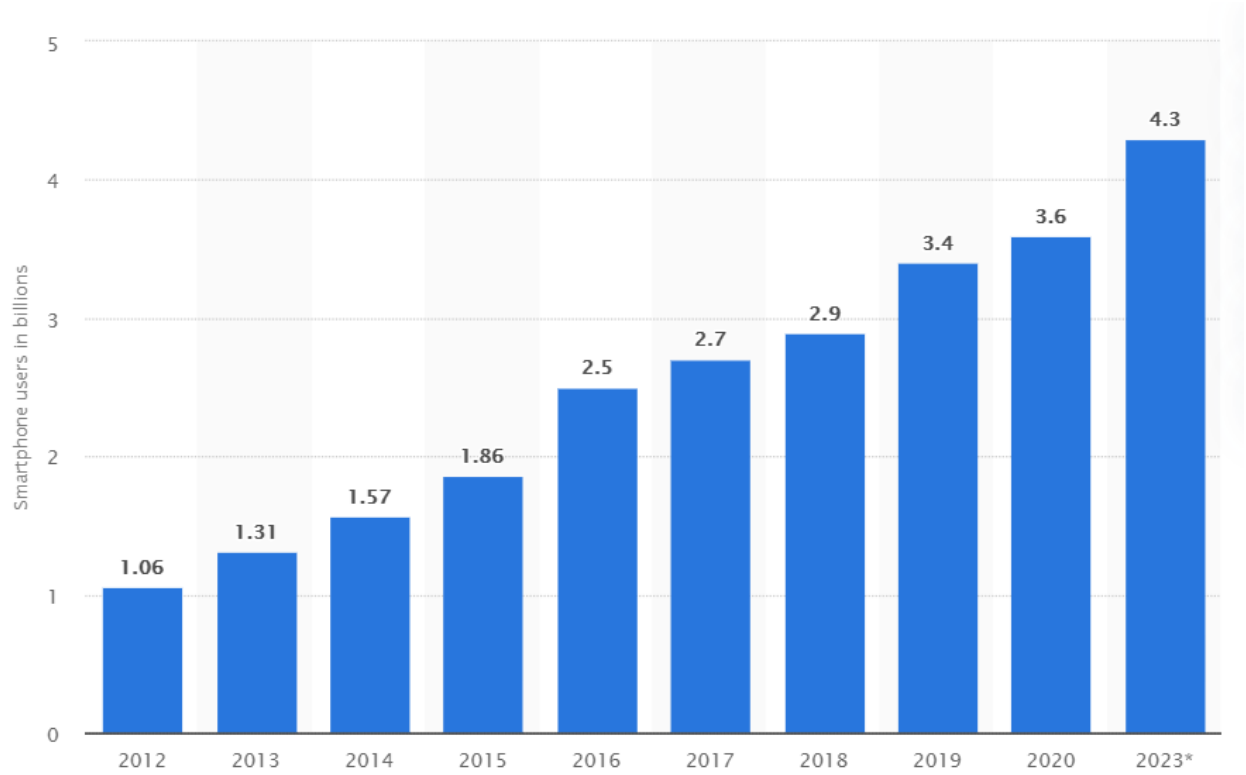
1.1 Οι έξυπνες συσκευές και μία σύντομη αναδρομή στην ιστορία τους

Ο κλάδος της κινητής τηλεφωνίας έχει πραγματοποιήσει μεγάλη πρόοδο τα τελευταία 30 χρόνια, ειδικότερα με την κυκλοφόρηση των πρώτων κινητών συσκευών με δυνατότητα σύνδεσης στο διαδίκτυο και τις πρώτες έξυπνες κινητές συσκευές. Το 1994 έγινε η πρώτη απόπειρα έξυπνης συσκευής από την IBM με την κυκλοφόρηση του Simon Personal Communicator (SPC) [1][2][3]. Η έκδοση αυτή περιείχε πολλές λειτουργικότητες οι οποίες είναι πλέον βασικές για τα κινητά σήμερα. Για παράδειγμα, η συσκευή είχε οθόνη αφής, καθώς και την ικανότητα να στείλει και να λάβει μηνύματα ηλεκτρονικού ταχυδρομείου. Ακόμα, είχε ημερολόγιο και βιβλίο επαφών, και συνοδευόταν από ένα στυλό οθόνης.

Εικόνα 1.1: Το Simon Personal Communicator (1994) (1.).

Στις αρχές του 2000, δόθηκε για πρώτη φορά στις κινητές συσκευές η δυνατότητα σύνδεσης σε δίκτυο 3G, δηλαδή οι συσκευές τηλεφωνίας πλέον υποστήριζαν σύνδεση με το διαδίκτυο. Η χρονιά σταθμός στην ιστορία των έξυπνων συσκευών ήταν το 2007, όπου ο Steve Jobs και η ομάδα της Apple κυκλοφόρησαν το πρώτο iPhone. Η συσκευή αυτή, εκτός από το απλό ύφος που διέθετε, έδινε στο χρήστη τη δυνατότητα να περιηγηθεί στο διαδίκτυο όπως θα κάνανε και από έναν υπολογιστή.

Σήμερα πλέον ο αριθμός των χρηστών με κινητές συσκευές ξεπερνάει τα 5 δισεκατομμύρια, ενώ οι χρήστες με έξυπνες συσκευές αγγίζουν τα 3 δισεκατομμύρια [3]. Στο διάγραμμα παρακάτω φαίνεται η ραγδαία αύξηση των έξυπνων κινητών συσκευών από το 2012 έως το 2020, καθώς επίσης και μία πρόβλεψη του αριθμού αυτού που αγγίζει τα 4.3 δισεκατομμύρια το 2023.



Σχήμα 1.1: Η σταδιακή αύξηση των έξυπνων συσκευών την τελευταία δεκαετία (2.).

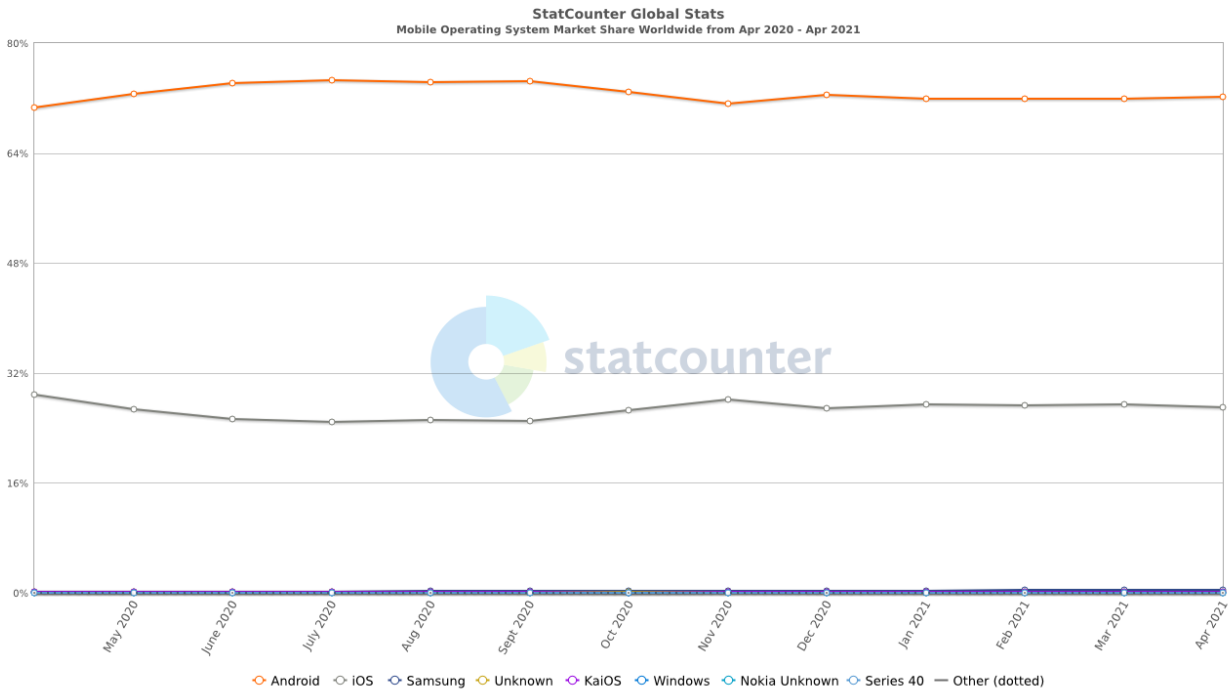
Είναι ξεκάθαρο πλέον ότι οι έξυπνες συσκευές αποτελούν αναπόσπαστο κομμάτι της ζωής μας και επηρεάζουν τον τρόπο με τον οποίο πραγματοποιούμε ποικίλες δραστηριότητες, όπως η παραγγελιοληψία προϊόντων και γευμάτων, η ανάγνωση ειδήσεων και άλλες. Ο νεαρότερος πληθυσμός, όπως οι μαθητές και οι φοιτητές, παρατηρείται ότι είναι πολύ εξοικειωμένοι με τις συσκευές τους και τις χρησιμοποιούν περισσότερες ώρες τη μέρα από ότι άλλες ηλικιακές ομάδες.

Πολλά σχολεία και ιδρύματα προσπάθησαν κατά τα πρώτα χρόνια κυκλοφορίας των έξυπνων συσκευών να απαγορεύσουν τη χρήση τους μέσα στους χώρους τους, καθώς θεωρούσαν πως αποσπούν τους μαθητές από το καθορισμένο μονοπάτι εκπαίδευσης. Με τα χρόνια όμως όλο και περισσότεροι εκπαιδευτικοί κατανοούν την ανάγκη το μονοπάτι αυτό να επαναπροσδιοριστεί χρησιμοποιώντας τα τις έξυπνες συσκευές ως εργαλείο στην εκπαίδευση. Πολλοί χρησιμοποιούν τις συσκευές και τις βασικές τους λειτουργίες για απλές δραστηριότητες όπως χρήση μηχανών αναζήτησης, υπολογισμό πράξεων, ομαδική συνομιλία της τάξης για εύκολη επικοινωνία με τους μαθητές. Εκτός από τη χρήση των βασικών λειτουργιών των συσκευών, έχουν κάνει ήδη την

εμφάνισή τους πολλές εφαρμογές εκπαιδευτικού χαρακτήρα. Μερικές από αυτές είναι η Duolingo, Instructable, Evernote.

1.2 Λειτουργικά συστήματα έξυπνων συσκευών

Υπάρχουν πολλά λειτουργικά συστήματα για τις έξυπνες κινητές συσκευές, παρόλα αυτά δύο εξ αυτών τα συναντάμε πιο συχνά, το iOS της Apple και το Android της Google. Το μεγαλύτερο κομμάτι της αγοράς ανήκει στο Android με ποσοστό λίγο πάνω από 70%, ενώ το iOS εξυπηρετεί ένα ποσοστό κοντά στο 25% [4]. Η διαφορά αυτή οφείλεται κυρίως στο ότι οι συσκευές iOS είναι πιο διαδεδομένες στις Ηνωμένες Πολιτείες και όχι τόσο στον υπόλοιπο κόσμο.



Σχήμα 1.2: Το ποσοστό της αγοράς για τα λειτουργικά συστήματα των έξυπνων συσκευών (3.).

Τα λειτουργικά αυτά συστήματα προσφέρουν ένα μεγάλο φάσμα εφαρμογών, οι οποίες καλύπτουν πολλές ανάγκες αλλα δημιουργούν και νέες. Οι πιο βασικές είναι εφαρμογές τηλεφωνικών κλήσεων, μηνυμάτων, βιντεοκλήσεων, ηλεκτρονικού ταχυδρομείου, περιηγητή ιστού, λήψης και συλλογής φωτογραφιών και βίντεο, αριθμομηχανή και πολλές άλλες. Πέρα από τις βασικές εφαρμογές, ο προγραμματιστής έχει τη δυνατότητα να δημιουργήσει δικές του εφαρμογές βασισμένες στο λειτουργικό σύστημα που επιθυμεί. Η εφαρμογή που υλοποιήσαμε στην παρούσα διπλωματική εργασία βασίζεται στο σύστημα iOS. Μετά τη δημιουργία, ο προγραμματιστής μπορεί να δημοσιοποιήσει τις εφαρμογές του στα ηλεκτρονικά καταστήματα εφαρμογών που παρέχονται στις έξυπνες συσκευές, στην τιμή που εκείνος επιθυμεί. Επιπλέον,

συσκευές που μοιράζονται την ίδια οικογένεια λειτουργικού συστήματος μπορούν να επικοινωνούν πιο εύκολα και να διατηρούν κοινούς λογαριασμούς χρήστη. Πρωτοπόρος σε αυτή τη μεθοδολογία είναι η Apple, όπου η σύνδεση μεταξύ των συσκευών της είναι τόσο εύκολη που προτρέπει τους χρήστες να αποκτήσουν περισσότερες Apple συσκευές ώστε να απολαμβάνουν ένα πολύ φιλικό οικοσύστημα έξυπνων συσκευών. Στην υλοποίηση πιο αποτελεσματικών λειτουργικών συστημάτων συμβάλει και η πρόοδος του μεγέθους των επεξεργαστών των συσκευών [5]. Με το πέρασμα των χρόνων, οι μονάδες επεξεργασίας μικραίνουν σε όγκο και γίνονται πιο ισχυρές υπολογιστικά, με αποτέλεσμα τα λειτουργικά συστήματα να είναι ακόμα πιο γρήγορα και να προσφέρουν τη δυνατότητα δημιουργίας πολύπλοκων εφαρμογών, τη χρήση αισθητήρων και προχωρημένων τεχνολογιών κάμερας.

Η εξέλιξη αυτή έχει οδηγήσει τα τελευταία χρόνια στη δημιουργία εφαρμογών επαυξημένης πραγματικότητας, οι οποίες απαιτούν υψηλή κατανάλωση της μπαταρίας της συσκευής, αλλά και την ύπαρξη αισθητήρων κίνησης της συσκευής και αισθητήρων βάθους της κάμερας. Με αυτά ως εργαλεία, ο χρήστης είναι πλέον σε θέση να βιώσει μία νέα πραγματικότητα μέσα στο χώρο όπου βρίσκεται, καθώς η εφαρμογή “ενισχύει” τον πραγματικό χώρο γύρω του με ψηφιακά δεδομένα και αντικείμενα, όπως ένα δράκο στην αυλή του. Εκτός από τον τομέα της φαντασίας και των παιχνιδιών, οι εφαρμογές επαυξημένης πραγματικότητας μπορούν να είναι χρήσιμες για καθημερινές εργασίες, όπως την μέτρηση αποστάσεων στο σπίτι, την μελέτη των σκιών κατά τη διάρκεια της μέρας για τα συνεργεία κινηματογράφου, την πλοήγηση του χρήστη σε ένα μεγάλο χώρο όπως μία βιβλιοθήκη, και πολλές άλλες. Παρακάτω αναλύουμε τον τρόπο με τον οποίο λειτουργεί αυτή η τεχνολογία στις συσκευές και με ποια εργαλεία υλοποιούνται.

1.3 Επαυξημένη πραγματικότητα

Η επαυξημένη πραγματικότητα αποτελεί την τεχνολογία με την οποία εισάγουμε ψηφιακά αντικείμενα στον πραγματικό κόσμο σε πραγματικό χρόνο. Η διαφορά με την εικονική πραγματικότητα είναι ότι η τελευταία προσφέρει στο χρήστη μία μετάβαση σε έναν εντελώς ψηφιακό κόσμο, αδιαφορώντας για τον πραγματικό χώρο γύρω του [6]. Η έρευνα και χρήση της ξεκίνησε από το 1968 στο Χάρβαρντ, ενώ το 2008 έγινε η πρώτη επίσημη χρήση της τεχνολογίας αυτής για καθημερινό σκοπό και συγκεκριμένα για τη διαφήμιση του αυτοκινήτου BMW Mini. Πλέον η τεχνολογία αυτή είναι διαθέσιμη στις έξυπνες συσκευές των χρηστών στη μορφή εφαρμογών, διαφόρων ειδών, όπως παιχνίδια, χάρτες, ξεναγήσεις κ.α. [7].

Χρησιμοποιούνται τεχνολογίες όπως η μηχανική όραση (Computer Vision), η συνεχής παρακολούθηση του βάθους (Depth Tracking) και ο ταυτόχρονος εντοπισμός και χαρτογράφηση ώστε πραγματοποιηθεί η εξερεύνηση του χώρου. Οι τεχνολογίες αυτές επιτρέπουν στην κάμερα της συσκευής να συγκεντρώνει, να στέλνει και να επεξεργάζεται πληροφορίες ώστε να προβάλει ψηφιακά μοντέλα στο επιθυμητό σημείο του χώρου. Προφανώς, ανάλογα με τις επεξεργαστικές δυνατότητες και την ισχύ της συσκευής του χρήστη, το αποτέλεσμα μπορεί να είναι λιγότερο ή περισσότερο ρεαλιστικό. Επειδή η εξερεύνηση στοιχείων του περιβάλλοντος χώρου συμβαίνει σε πραγματικό χρόνο, χρησιμοποιούνται στιγμιότυπα που λαμβάνονται από την κάμερα και

εκτελούνται οι απαραίτητες διεργασίες στο καθένα χωριστά και με σειρά. Πιο συγκεκριμένα, η διαδικασία αυτή μπορεί να αναλυθεί σε 4 στάδια [8].

1. Η εφαρμογή αναγνωρίζει τον κόσμο γύρω από το χρήστη με τη χρήση μηχανικής όρασης μέσω των στιγμιοτύπων που λαμβάνει από την κάμερα της συσκευής.

2. Κάθε στιγμιότυπο της κάμερας αναλύεται ώστε να αναγνωριστούν σημεία, στα οποία θα γίνει απεικόνιση των ψηφιακών δεδομένων, όπως για παράδειγμα η επιφάνεια ενός τραπεζιού, μία εικόνα σε ένα φυλλάδιο, το χέρι ενός χρήστη κ.α. Η διαδικασία αυτή μπορεί να πραγματοποιηθεί με διαφορετικές προσεγγίσεις (trackers, sensors, lasers, GPS), ανάλογα με τη φύση της εφαρμογής. Οι προσεγγίσεις αυτές θα αναλυθούν στη συνέχεια.

3. Για κάθε σημείο ενδιαφέροντος που αναγνωρίζεται, η εφαρμογή συγκεντρώνει τα ψηφιακά δεδομένα που θα προβληθούν σε αυτό, ώστε τελικά να επικαλύψει τον πραγματικό χώρο του σημείου με την πρόσθετο ψηφιακό υλικό.

4. Μόλις λάβει τα δεδομένα για το συγκεκριμένο σημείο, δημιουργεί μία νέα εικόνα του συγκεκριμένου στιγμιοτύπου, η οποία εκτός από τον πραγματικό χώρο και τα πραγματικά αντικείμενα απεικονίζει πλέον και τα ψηφιακά δεδομένα στο σημείο αυτό (rendering).

Για την αναγνώριση των σημείων υπάρχουν διαφορετικές μεθοδολογίες από τις οποίες ο σχεδιαστής της εφαρμογής καλείται να επιλέξει. Η πρώτη από αυτές είναι η τεχνολογία ταυτόχρονου εντοπισμού και χαρτογράφησης (Simultaneous Localization And Mapping technology, SLAM) [9]. Βασίζεται σε ένα σύνολο πολύπλοκων αλγορίθμων και σε δεδομένα που λαμβάνονται από αισθητήρες. Επιτρέπει στους υπολογιστές να κατανοήσουν το χώρο γύρω τους με τη βοήθεια συστήματος εντοπισμού σημείων των αντικειμένων ενδιαφέροντος που υπάρχουν στο χώρο αυτό. Ουσιαστικά κατασκευάζεται ή ανανεώνεται ένας χάρτης ενός άγνωστου περιβάλλοντος ενώ ταυτόχρονα καταγράφει την τοποθεσία της κύριας διεργασίας (agent) του προγράμματος. Πρόκειται για ένα πρόβλημα που θυμίζει το παράδοξο της κότας και του αυγού, παρόλα αυτά οι πολύπλοκοι αλγόριθμοι που χρησιμοποιούνται το λύνουν προσεγγιστικά σε ικανοποιητικό χρόνο. Η πολυπλοκότητα που εισάγει η τεχνολογία αυτή δημιουργεί υψηλές απαιτήσεις υλισμικού, αυξάνοντας το κόστος της χρήσης της. Επιπλέον, εφαρμογές που βασίζονται στη τεχνολογία SLAM απαιτούν έναν προκαθορισμένο χάρτη ώστε να τοποθετηθούν σωστά τα ψηφιακά τρισδιάστατα μοντέλα. Παρά τα μειονεκτήματα αυτά, η τεχνολογία αυτή χρησιμοποιείται ευρύτατα πλέον σε εφαρμογές αυτόνομων οχημάτων, ρομποτικών εξοπλισμών και drones.

Η πιο διαδεδομένη προσέγγιση για την ανίχνευση σημείων ενδιαφέροντος παρόλα αυτά είναι η μεθοδολογία αναγνώρισης με τη χρήση σημαδιών (markers) [9]. Τα σημάδια αυτά μπορεί να είναι AR-codes, QR-codes, φυσικά αντικείμενα, τυπωμένες εικόνες και άλλα. Χρησιμοποιούνται από τις εφαρμογές επαυξημένης πραγματικότητας πάνω σε πραγματικά αντικείμενα ώστε αυτά να τεθούν ως διακόπτες ενεργοποίησης κάποιου γεγονότος (event trigger). Μόλις ανιχνευθεί ένα σημάδι σε ένα αντικείμενο ενδιαφέροντος στη σκηνή, το λογισμικό της εφαρμογής υπολογίζει τη θέση του σημαδιού καθώς και του ψηφιακού μοντέλου που θα μπει στη θέση του. Για αυτό και κάθε αλλαγή

στη θέση του πραγματικού αντικειμένου θα επηρεάσει τη θέση του ψηφιακού μοντέλου που τοποθετείται σε αυτό. Όσο καλύτερα αντιλαμβάνεται η εφαρμογή πραγματικά αντικείμενα, τόσο πιο ακριβής θα είναι η τοποθέτηση και απόδοση των ψηφιακών μοντέλων της.

Η τρίτη και τελευταία μεθοδολογία αφορά εφαρμογές επαυξημένης πραγματικότητας που βασίζονται στην τοποθεσία (Position), οι οποίες χρησιμοποιούν τα δεδομένα που παράγονται από το GPS, την ενσωματωμένη πυξίδα, το επιταχυνσιόμετρο και το γυροσκόπιο της συσκευής. Για να μπορεί μία τέτοια εφαρμογή να αποδώσει ψηφιακά μοντέλα στη σωστή γεωγραφική θέση, πρέπει να είναι εφικτή η ενεργοποίηση γεγονότων βάσει της τοποθεσίας και της όψης της κάμερας της συσκευής [8]. Για παράδειγμα, μόλις η κάμερα της συσκευής στραφεί προς ένα εστιατόριο σε ένα συγκεκριμένο δρόμο, θα αναγνωριστεί η τοποθεσία η οποία βρίσκεται στη σκηνή της κάμερας και η εφαρμογή θα τοποθετήσει ένα ψηφιακό μοντέλο επιγραφής από πάνω του με το όνομά του, το μενού και οτιδήποτε άλλο προσφέρει η εφαρμογή στο χρήστη.

1.4 Μία σύντομη εισαγωγή στα εργαλεία ARKit και RealityKit

Το 2017 η Apple δημοσίευσε ένα εργαλείο (framework) με όνομα ARKit [13][14], το οποίο μέσα από τη διεπαφή του δίνει στους προγραμματιστές τη δυνατότητα να δημιουργήσουν εφαρμογές επαυξημένης πραγματικότητας για έξυπνες συσκευές με λειτουργικό σύστημα iOS. Προσφέρει πολλές λειτουργικότητες όπως η ανίχνευση επιφανειών, εικόνων, αντικειμένων, σωμάτων και προσώπων, καθώς επίσης και συνεργατικές συνεδρίες ώστε πολλοί χρήστες να μοιράζονται ταυτόχρονα την ίδια εμπειρία. Ακόμα, η προσθήκη ψηφιακών αντικειμένων στο χώρο μπορεί να γίνει ακόμα πιο ρεαλιστική μέσα από διάφορες επιλογές υφής και σκιών των αντικειμένων αυτών ώστε να ταιριάζουν στον περίγυρο, καθώς και την δυνατότητα απόκρυψης των αντικειμένων αν περάσει άνθρωπος ανάμεσα σε αυτά και την κάμερα. Επιπλέον, δημοσιεύτηκε ένα ακόμα εργαλείο με όνομα RealityKit, το οποίο εκμεταλλεύεται πολλές από τις πληροφορίες που προσφέρει το ARKit ώστε να εισάγει τρισδιάστατα μοντέλα στη σκηνή.

Εκτός από τα δύο εργαλεία που αναφέραμε, η Apple δημοσίευσε την εφαρμογή Reality Composer [16], με την οποία μπορεί ο προγραμματιστής να δημιουργήσει τρισδιάστατα μοντέλα κάθε μορφής, τα οποία μπορούν να περιέχουν κίνηση και ήχους. Τα μοντέλα αυτά δύναται να χρησιμοποιηθούν σε μία εφαρμογή επαυξημένης πραγματικότητας η οποία χρησιμοποιεί το εργαλείο RealityKit και κατ' επέκταση το ARKit, αρκεί να τα εισάγει ο προγραμματιστής στο περιβάλλον ανάπτυξής της. Παρά την πληθώρα επιλογών που προσφέρει το ARKit σε συνδυασμό με το Reality Composer, τα μοντέλα αυτά δεν έχουν τη δυνατότητα ενημέρωσης της μορφής τους κατά τη διάρκεια της λειτουργίας της εφαρμογής, πέρα από τις προκαθορισμένες κινήσεις που έχουν προγραμματιστεί να κάνουν. Για παράδειγμα, ένα μοντέλο μπάλας μπορεί να έχει προγραμματιστεί να αναπηδάει τρεις φορές όταν ο χρήστης πατάει πάνω της, όμως δεν είναι δυνατό η διάμετρός της να αλλάξει σύμφωνα με μία τιμή που θα της δώσουμε κατά τη διάρκεια λειτουργίας της εφαρμογής.

1.5 Αντικείμενο διπλωματικής εργασίας

Το αντικείμενο της διπλωματικής εργασίας αποτελεί ο σχεδιασμός και η υλοποίηση μίας εφαρμογής επαυξημένης πραγματικότητας εκπαιδευτικού χαρακτήρα, η οποία δεν υπάρχει στην αγορά και με την οποία οι μαθητές και καθηγητές θα είναι σε θέση για πρώτη φορά να αναπαραστήσουν μία άσκηση φυσικής και να πειραματιστούν με διαφορετικές παραμέτρους της. Συγκεκριμένα, η άσκηση φυσικής αφορά το νόμο του Coulomb και ο χρήστης της εφαρμογής θα είναι σε θέση να εισάγει πάνω σε μία επιφάνεια τρισδιάστατα μοντέλα φορτίων, τα οποία θα συνοδεύονται από τις αποστάσεις και τις δυνάμεις τους. Οι τιμές των φορτίων θα αλλάζουν μέσα από κατάλληλο μενού επεξεργασίας, ενώ τα μοντέλα αποστάσεων και δυνάμεων θα ενημερώνουν συνεχώς τη μορφή τους ανάλογα με την τιμή τους.

Στην προηγούμενη ενότητα είδαμε ότι το ARKit δε δίνει τη δυνατότητα ενημέρωσης των τιμών των διαστάσεων των τρισδιάστατων μοντέλων που δημιουργούνται μέσω του εργαλείου Reality Composer. Επομένως στα πλαίσια αυτής της εργασίας γίνεται μία πρώτη απόπειρα δημιουργίας μίας εφαρμογής που χρησιμοποιεί τρισδιάστατα μοντέλα τα οποία θα ενημερώνονται ζωντανά κατά τη λειτουργία της. Επίσης, στην προσπάθεια αυτή εξασφαλίζουμε ότι η απόδοση και η ποιότητα της εφαρμογής λόγω του φόρτου των γραφικών δε θα μειωθεί. Θα μπορούσαμε να συνοψίσουμε λοιπόν τη συνεισφορά της παρούσας εργασίας στα παρακάτω σημεία:

- Η μελέτη των διαφορετικών λειτουργικοτήτων που προσφέρει το ARKit καθώς και οι τρόποι με τους οποίους τις υλοποιούμε στην εφαρμογή μας.
- Η επιλογή των απαραίτητων παραμέτρων για την ανίχνευση κατάλληλων επιφανειών στο χώρο.
- Η συγγραφή κώδικα με τον οποίο ξεπερνάμε το πρόβλημα αλλαγής των διαστάσεων των μοντέλων και καταφέρνουμε τα μοντέλα να ενημερώνουν τη μορφή τους συνεχώς κατά τη διάρκεια λειτουργίας της εφαρμογής, χωρίς να χαλάει η ποιότητά της.
- Τα συμπεράσματα που προκύπτουν από τη διαδικασία ανάλυσης και υλοποίησης μίας εφαρμογής επαυξημένης πραγματικότητας με το ARKit, χάρις τα οποία μπορεί κανείς να καθοδηγηθεί για την πορεία δημιουργίας μίας εφαρμογής με μοντέλα που αλλάζουν μορφή δυναμικά.
- Οι εκπαιδευτικές επεκτάσεις της εφαρμογής που προτείνονται στο τέλος της εργασίας, καθώς και διαφορετικές μέθοδοι παρουσίασης των τρισδιάστατων μοντέλων ώστε να αντιμετωπίζεται το ζήτημα της ζωντανής ενημέρωσής τους κατά τη λειτουργία της εφαρμογής.

1.6 Διάρθρωση της εργασίας

Στην παρούσα διπλωματική εργασία παρουσιάζουμε αρχικά στο δεύτερο κεφάλαιο τις λειτουργικότητες που χρησιμοποιήσαμε για την υλοποίηση της εφαρμογής μας, ξεκινώντας από το θεωρητικό υπόβαθρο των πλαισίων ARKit και RealityKit και καταλήγοντας σε μία πιο λεπτομερή ανάλυση των χαρακτηριστικών τους. Τα επόμενα τρία κεφάλαια αφορούν τη διαδικασία δημιουργίας της εφαρμογής.

Στο τρίτο κεφάλαιο παρουσιάζεται η εφαρμογή που αναπτύξαμε στα πλαίσια της παρούσας διπλωματικής και πιο συγκεκριμένα στην ανάλυση και σχεδιασμό και τελικά στην υλοποίησή της σε δύο διαδοχικές υποενότητες. Στην πρώτη υποενότητα γίνεται αναφορά στις απαιτήσεις της εφαρμογής, δηλαδή τα απαραίτητα χαρακτηριστικά που πρέπει να έχει για την ομαλή αλληλεπίδρασή της με το χρήστη. Στη συνέχεια παρατίθενται οι τρόποι με τους οποίους η εφαρμογή αλληλεπιδρά με το χρήστη ενώ στη δεύτερη υποενότητα παρουσιάζονται όλα τα σενάρια χρήσης όπου η εφαρμογή θα υποστηρίζει. Στην τρίτη υποενότητα περιγράφουμε τις οθόνες της εφαρμογής καθώς και μία λίστα από τους ορισμούς των κλάσεων, των μεταβλητών και των μεθόδων που χρησιμοποιήθηκαν.

Στη τέταρτη υποενότητα του τρίτου κεφαλαίου περιγράφουμε την υλοποίηση της εφαρμογής μας, ξεκινώντας με μία ανάλυση σε βάθος των χαρακτηριστικών των πλαισίων ARKit και RealityKit που χρησιμοποιήσαμε, ενώ γίνεται και μία εισαγωγή στις έννοιες, τον κύριο λόγο και τη λύση του προβλήματος ενημέρωσης μοντέλων. Στην πέμπτη και τελευταία υποενότητα, εμβαθύνουμε ακόμα περισσότερο στην αντιμετώπιση του προβλήματος ανανέωσης των διαστάσεων των τρισδιάστατων μοντέλων. Συγκεκριμένα, επεξηγούμε με ποιο τρόπο καταφέρνουμε την μεταβολή του μήκους των μοντέλων των δυνάμεων και των αποστάσεων μεταξύ των φορτίων κατά τη διάρκεια λειτουργίας της εφαρμογής, βάσει των θέσεων που τα τοποθετεί ο χρήστης. Στο μέρος αυτό επιλέξαμε να μην παρουσιάσουμε όλα τα κομμάτια κώδικα της εφαρμογής εκτός από τα πιο κύρια στα οποία βασίζεται η εργασία. Αντ' αυτού, επιλέξαμε να μεταφέρουμε τα κομμάτια αυτά στο παράρτημα Β.

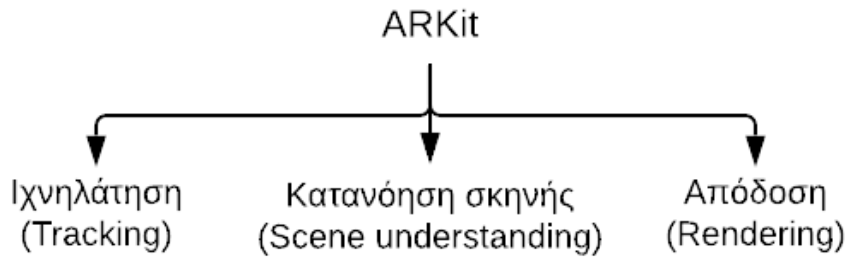
Τέλος, στο τέταρτο κεφάλαιο αναφέρουμε τα συμπεράσματα που εξάγαμε από τη διαδικασία ανάπτυξης αυτής της εφαρμογής, ειδικά όσον αφορά τη χρήση μεταβαλλόμενων τρισδιάστατων μοντέλων σε εφαρμογές που χτίζονται με τα εργαλεία ARKit και RealityKit. Επίσης, παραθέτουμε ορισμένες επεκτάσεις τις οποίες μπορεί να λάβει η εφαρμογή μας για την καλύτερη αξιοποίηση των πλαισίων αυτών, καθώς επίσης και ορισμένους τρόπους με τους οποίους θα μπορούσαμε να αποφύγουμε το πρόβλημα ενημέρωσης των τρισδιάστατων μοντέλων.

2. Το προγραμματιστικό περιβάλλον ARKit

2.1 Το θεωρητικό υπόβαθρο του εργαλείου ARKit

Το ARKit επιτρέπει στον προγραμματιστή να δημιουργήσει πιο εύκολα και γρήγορα εφαρμογές επαυξημένης πραγματικότητας μέσω της διεπαφής (API) που παρέχει. Χρησιμοποιεί τις κάμερες, τους επεξεργαστές και τους ανιχνευτές κίνησης της iOS συσκευής του χρήστη ώστε να επιτρέπει ρεαλιστικές αλληλεπιδράσεις των ψηφιακών αντικειμένων με το πραγματικό περιβάλλον γύρω από το χρήστη. Για να αναγνωρίσει τη θέση και την κίνηση της συσκευής του χρήστη στο χώρο, το ARKit βασίζεται σε μία τεχνολογία με όνομα Οπτική Αδρανειακή Οδομετρία (Virtual Inertial Odometry - VIO) [15]. Χρησιμοποιεί τα δεδομένα που συγκεντρώνει, όχι μόνο για να αναλύσει τη διάταξη του χώρου, αλλά και για να ανιχνεύσει επιφάνειες (οριζόντιες ή/και κάθετες) όπως τοίχους και τραπέζια και άλλα σημεία ενδιαφέροντος.

Μπορούμε να αναλύσουμε τη λειτουργία του ARKit σε τρεις βασικούς πυλώνες, την ιχνηλάτηση (Tracking), την κατανόηση του χώρου (Scene Understanding) και την απόδοση των ψηφιακών δεδομένων και τρισδιάστατων μοντέλων (Rendering) [28] [29].

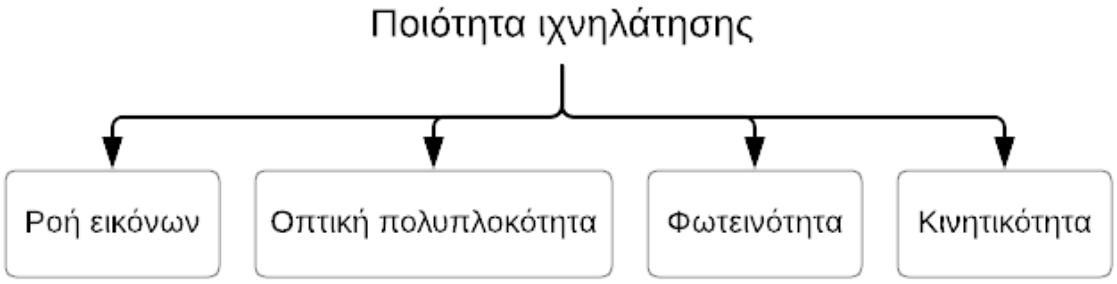


Σχήμα 2.1: Οι τρεις πυλώνες του ARKit

Η ιχνηλάτηση δίνει τη δυνατότητα στην εφαρμογή να παρακολουθεί συνεχώς τη θέση και κίνηση της συσκευής σε σχέση με το πραγματικό περιβάλλον γύρω της. Επίσης, καθιστά εφικτή και την αναγνώριση μίας φυσικής τοποθεσίας στο χώρο σε πραγματικό χρόνο. Για να γίνει κατανοητό πόσο σημαντικό χαρακτηριστικό είναι η διαδικασία της ιχνηλάτησης για μία εφαρμογή επαυξημένης πραγματικότητας, μπορεί κανείς να φανταστεί μία ψηφιακή καρέκλα τοποθετημένη στην επιφάνεια του πατώματος. Προφανώς, όσο και αν ο χρήστης κινεί ή περιστρέφει τη συσκευή του στο δωμάτιο, έχουμε την απαίτηση από την εφαρμογή μας να διατηρεί την καρέκλα σταθερή στην αρχική της θέση, χωρίς να αλλάζει τον προσανατολισμό της. Αυτό επιτυγχάνεται μέσω της ιχνηλάτησης του χώρου.

Η τεχνολογία VIO [15], στην οποία αναφερθήκαμε προηγουμένως, δέχεται συνεχώς στιγμιότυπα από την κάμερα καθώς και δεδομένα κίνησης από τους αισθητήρες της συσκευής ώστε να χτίσει μια εικόνα για τη θέση της συσκευής στο χώρο τη δεδομένη στιγμή και για τον προσανατολισμό της. Οι αισθητήρες αυτοί βρίσκονται σε κάθε συσκευή που πληροί τις απαραίτητες προδιαγραφές, και δεν απαιτείται επιπλέον εξοπλισμός. Σημαντικό πλεονέκτημα αυτής της μεθοδολογίας είναι ότι δεν απαιτείται επίσης καμία απολύτως πρώιμη γνώση του χώρου στον οποίο θα χρησιμοποιηθεί η εφαρμογή επαυξημένης πραγματικότητας.

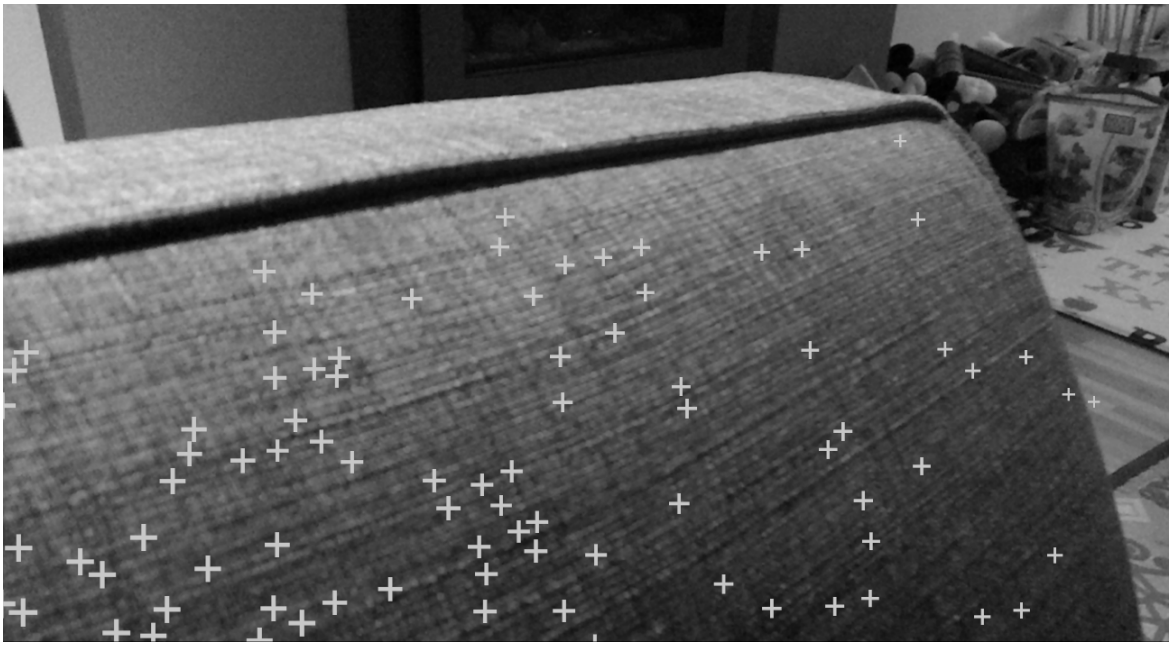
Για να πετύχει η ιχνηλάτηση, είναι απαραίτητο να διατηρούμε την μήτρα προβολής (project matrix) που παρέχει το εργαλείο ARKit ενημερωμένη. Ο πίνακας αυτός περιέχει τα γεωμετρικά στοιχεία της κάμερας που βοηθούν στην αποτελεσματική μετάφραση του τρισδιάστατου κόσμου στη δισδιάστατη οθόνη. Φυσικά, δεν είναι εφικτό η ιχνηλάτηση να είναι πάντα ακριβής και η ακρίβειά της εξαρτάται από τους παρακάτω παράγοντες.



Σχήμα 2.2: Οι κύριοι παράγοντες που επηρεάζουν την ποιότητα ιχνηλάτησης

Η συνεχής ροή πληροφοριών και εικόνων που καταφθάνουν από το υλισμικό της συσκευής αποτελούν τον πρώτο παράγοντα της ποιότητας της ιχνηλάτησης. Μόλις η συσκευή σταματήσει να παρέχει τα παραπάνω δεδομένα, σταματάει αυτομάτως κι η ιχνηλάτηση του χώρου. Ένας ακόμα παράγοντας είναι η υφή του περιβάλλοντος χώρου, δηλαδή η οπτική πολυπλοκότητα που παρέχεται στην κάμερα ώστε να βρεθούν σημεία ενδιαφέροντος (feature points) στο χώρο αυτό. Για παράδειγμα, αν ο χρήστης κοιτάει με την κάμερά του έναν άσπρο τοίχο ή βρίσκεται σε χώρο με λιγοστό φωτισμό, η κάμερα θα δυσκολευτεί πολύ να βρει σημεία ενδιαφέροντος, και η ποιότητα της ιχνηλάτησης θα περιοριστεί αρκετά. Επίσης σημαντικό ρόλο παίζει η κινητικότητα που υπάρχει στη σκηνή, καθώς όταν υπάρχει αρκετή κίνηση σε όσα καταγράφει η κάμερα οι εικόνες δε θα αντιστοιχούν στα δεδομένα κίνησης, κάτι το οποίο προκαλεί και πάλι περιορισμό στην ποιότητα ιχνηλάτησης.

Η κατανόηση σκηνής αφορά την ικανότητα της συσκευής να αναγνωρίσει χαρακτηριστικά του περιβάλλοντος γύρω της. Κατά τη διάρκεια λειτουργίας της εφαρμογής, το ARKit δημιουργεί ένα νέφος από σημεία ενδιαφέροντος (feature points) τα οποία αντιπροσωπεύουν πληροφορίες που συλλέγονται κατά την διαδικασία κατανόησης του χώρου γύρω από το χρήστη. Το κάθε ένα από αυτά τα σημεία αντιπροσωπεύει ένα μέρος της σκηνής που είναι πολύ πιθανό να ανήκει σε μία επιφάνεια του πραγματικού κόσμου.



Εικόνα 2.1: Τα σημεία ενδιαφέροντος (feature points) που δημιουργεί η εφαρμογή για την κατανόηση της σκηνής και συγκεκριμένα για τον καναπέ (4.).

Το ARKit επωφελείται από τις πολλές δυνατότητες που του παρέχει η κατανόηση σκηνής. Μία από τις πιο σημαντικές είναι η αναγνώριση επιφανειών (Plane Detection), όπως το πάτωμα ή ένα τραπέζι. Η συγκεκριμένη λειτουργία χρησιμοποιείται πολύ συχνά στις εφαρμογές επαυξημένης πραγματικότητας, όπως και στη δική μας, καθώς οι διάφορες επιφάνειες χρησιμοποιούνται ως βάσεις για τα ψηφιακά αντικείμενα.

Μία ακόμα σημαντική λειτουργικότητα που χρησιμοποιήσαμε και στην εφαρμογή μας είναι η εκτίμηση του φωτισμού (Light Estimation). Τα ψηφιακά αντικείμενα βασίζονται σε αυτή για να αποδίδονται με τον κατάλληλο φωτισμό, αντανακλάσεις και σκιές ώστε να ταιριάζουν με τον περιβάλλοντα χώρο. Το ARKit χρησιμοποιεί τις πληροφορίες φίλτρου φωτισμού της κάμερας (Camera Exposure) στις εικόνες που λαμβάνει από αυτή, ώστε να υπολογίσει τη σχετική τους φωτεινότητα. Για μια πολύ καλά φωτισμένη εικόνα, η τιμή αυτή αντιστοιχεί σε 1000 lumen. Όσο πιο φωτεινό είναι το περιβάλλον, τόσο μεγαλύτερη είναι η τιμή της σχετικής φωτεινότητας.

2.2 Απαραίτητα εργαλεία ανάπτυξης

Όπως κάθε iOS εφαρμογή, έτσι και η εφαρμογή επαυξημένης πραγματικότητας που σχεδιάσαμε με τα πλαίσια ARKit και RealityKit δημιουργήθηκε στο περιβάλλον ανάπτυξης Xcode [11]. Το συγκεκριμένο περιβάλλον διανέμεται δωρεάν στο ηλεκτρονικό κατάστημα App Store και παρέχει τη δυνατότητα σύνδεσης της εφαρμογής με μία πραγματική συσκευή ή έναν προσομοιωτή συσκευής, ώστε να μπορεί ο προγραμματιστής να δοκιμάζει τον κώδικά του. Στην περίπτωση εφαρμογών επαυξημένης πραγματικότητας δεν αρκεί ένας προσομοιωτής έξυπνης συσκευής αλλά απαιτείται η σύνδεση με μία πραγματική, καθώς η χρήση της κάμερας είναι απαραίτητη. Η γλώσσα προγραμματισμού που χρησιμοποιήσαμε για τη συγγραφή του κώδικα της εφαρμογής μας είναι η γλώσσα Swift [12], η οποία απευθύνεται αποκλειστικά σε εφαρμογές των λειτουργικών συστημάτων iOS και macOS της Apple. Τέλος, για να εργαστεί κανείς στο εργαλείο Xcode είναι απαραίτητο ο υπολογιστής του να τρέχει το λειτουργικό σύστημα macOS, οπότε και επιλέξαμε να εργαστούμε σε ένα Macbook Pro 2018, ενώ η έξυπνη συσκευή που χρησιμοποιήσαμε για τις δοκιμές μας ήταν ένα iPhone SE 2020.

2.3 Οι λειτουργικότητες των εργαλείων ARKit και RealityKit που χρησιμοποιήσαμε στην εφαρμογή AR Coulomb

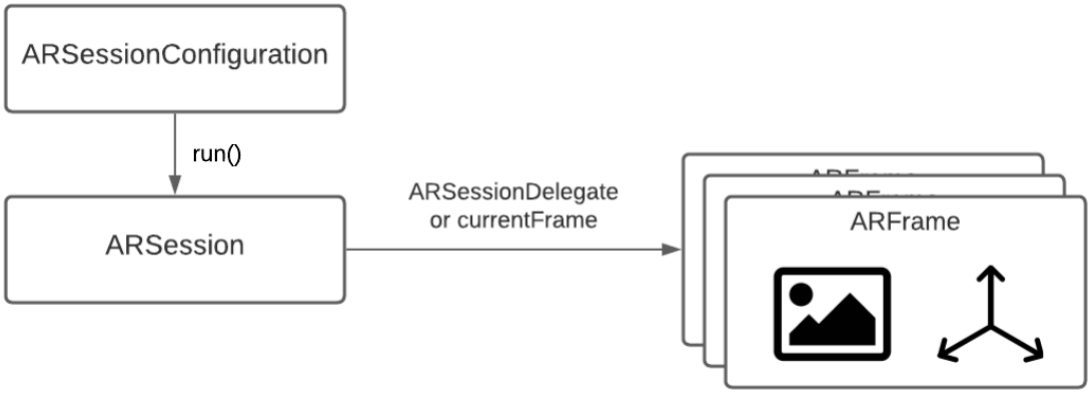
Στην εφαρμογή AR Coulomb χρησιμοποιήσαμε ορισμένες από τις βασικές λειτουργικότητες που προσφέρουν τα εργαλεία ARKit και RealityKit. Στο κεφάλαιο αυτό πραγματοποιείται μία ανάλυση αυτών και των τρόπων με τους οποίους τις αξιοποιήσαμε.

2.3.1 Κύκλος ζωής - Συνεδρίες (ARSession)

Το ARKit βασίζει τη ροή λειτουργίας του σε συνεδρίες, που αντιπροσωπεύονται από την κλάση *ARSession* [20]. Ένα αντικείμενο συνεδρίας *ARSession* συντονίζει τις κύριες διεργασίες της εφαρμογής, οι οποίες περιλαμβάνουν την ανάγνωση δεδομένων από τους ανιχνευτές κίνησης της συσκευής, τον χειρισμό της κάμερας καθώς και την ανάλυση των εικόνων που καταγράφονται. Το αντικείμενο συνεδρίας συνδυάζει τα αποτελέσματα αυτά ώστε να αποδώσει μία αντιστοιχία μεταξύ του πραγματικού χώρου γύρω από τη συσκευή και του ψηφιακού χώρου, ο οποίος είναι ένας συνδυασμός του πραγματικού χώρου και των ψηφιακών δεδομένων στον οποίο θα αποδοθούν τελικά και τα τρισδιάστατα μοντέλα της εφαρμογής.

Επομένως, πρώτο στάδιο για τη λειτουργία της εφαρμογής μας είναι η δημιουργία ενός αντικειμένου συνεδρίας. Αφού δημιουργηθεί, πρέπει να λάβει μία παραμετροποίηση που περιγράφει το είδος ιχνηλάτησης που θα πραγματοποιήσει η εφαρμογή μας, καθώς υπάρχουν διαφορετικά είδη όπως είναι η ιχνηλάτηση χώρου, σώματος, προσώπου και άλλες. Υπάρχουν διαφορετικές κλάσεις παραμετροποίησης αναλόγως το είδος ιχνηλάτησης που επιδιώκουμε, όλες όμως κληρονομούν από την βασική κλάση *ARConfiguration.* Στην εφαρμογή μας στοχεύουμε στην ιχνηλάτηση του περιβάλλοντα χώρου, οπότε και χρησιμοποιήσαμε ένα αντικείμενο παραμετροποίησης *ARWorldTrackingConfiguration*. Η κλάση αυτή παρέχει τον προσανατολισμό της συσκευής και τη σχετική της θέση στο χώρο, καθώς και άλλες επιπρόσθετες λειτουργικότητες όπου η κλάση *ARConfiguration* δεν προσφέρει, όπως η λειτουργικότητα "Hit Testing" που θα αναλύσουμε σε επόμενη ενότητα.

Για να ξεκινήσει η συνεδρία, αρκεί η εκτέλεση της συνάρτησης *run(_ configuration)*, η οποία δέχεται ως όρισμα το αντικείμενο παραμετροποίησης που ορίσαμε προηγουμένως. Χωρίς να χρειαστεί κάποια περαιτέρω ενέργεια από τον προγραμματιστή, ξεκινάει αμέσως κι η διαδικασία καταγραφής, δηλαδή δημιουργείται αυτόματα μία συνεδρία τύπου *AVCaptureSession* καθώς και ένα αντικείμενο διαχειριστή κίνησης *CMMotionManager*. Μόλις ολοκληρωθεί η επεξεργασία, η συνεδρία αποδίδει στιγμιότυπα που αντιπροσωπεύονται από μία συνεχόμενη ροή *ARFrame* αντικειμένων .



Σχήμα 2.3: Από την παραμετροποίηση μίας συνεδρίας στην παραγωγή των *ARFrame* αντικειμένων.

Ανά πάσα στιγμή η συνεδρία μπορεί να παύσει καλώντας τη συνάρτηση *pause*, η οποία σταματάει προσωρινά όλες τις επεξεργαστικές λειτουργίες της συνεδρίας και είναι χρήσιμη σε περιπτώσεις όπου μία όψη της εφαρμογής δεν είναι πλέον ορατή οπότε δεν χρειάζεται να σπαταλάει ισχύ από τον επεξεργαστή της συσκευής. Επίσης, είναι εφικτή η κλήση της συνάρτησης *run* περισσότερες από μία φορά, με σκοπό κάποια αλλαγή στο αντικείμενο παραμετροποίησης. Αν για παράδειγμα επιθυμούμε να ενεργοποιήσουμε την ανίχνευση επιφανειών (Plane Detection), αρκεί να δημιουργήσουμε ένα νέο αντικείμενο παραμετροποίησης που να την ενεργοποιεί και να το θέσουμε ως όρισμα στη μέθοδο *run* του τρέχοντος αντικειμένου *ARSession*.

Στην εφαρμογή ARCoulomb, δημιουργούμε κατά την εκκίνηση της εφαρμογής ένα αντικείμενο *ARSession*, το οποίο αντιπροσωπεύει τη συνεδρία της. Έπειτα, δημιουργούμε και προσθέτουμε στο αντικείμενο συνεδρίας μία παραμετροποίηση τύπου *ARWorldTrackingConfiguration*, η οποία ενεργοποιεί την ιχνηλάτηση του χώρου και ταυτόχρονα μας επιτρέπει να ανιχνεύσουμε επιφάνειες σε αυτόν. Κάθε φορά που ο χρήστης επανεκκινεί την εμπειρία, φροντίζουμε να δημιουργούμε από την αρχή μία νέα συνεδρία με παρόμοια παραμετροποίηση.

2.3.2 Σύστημα κόμβων (ARAnchor System)

Στον πραγματικό χώρο υπάρχουν πολλά σημεία, όπως επιφάνειες και αντικείμενα, για τα οποία θέλουμε η εφαρμογή μας να είναι συνεχώς ενήμερη για τη θέση τους σε σχέση με τη συσκευή μας. Το εργαλείο ARKit χρησιμοποιεί αντικείμενα της κλάσης *ARAnchor* για να αντιπροσωπεύει τη θέση και τον προσανατολισμό αυτών των σημείων μέσα στη σκηνή επαυξημένης πραγματικότητας [21].

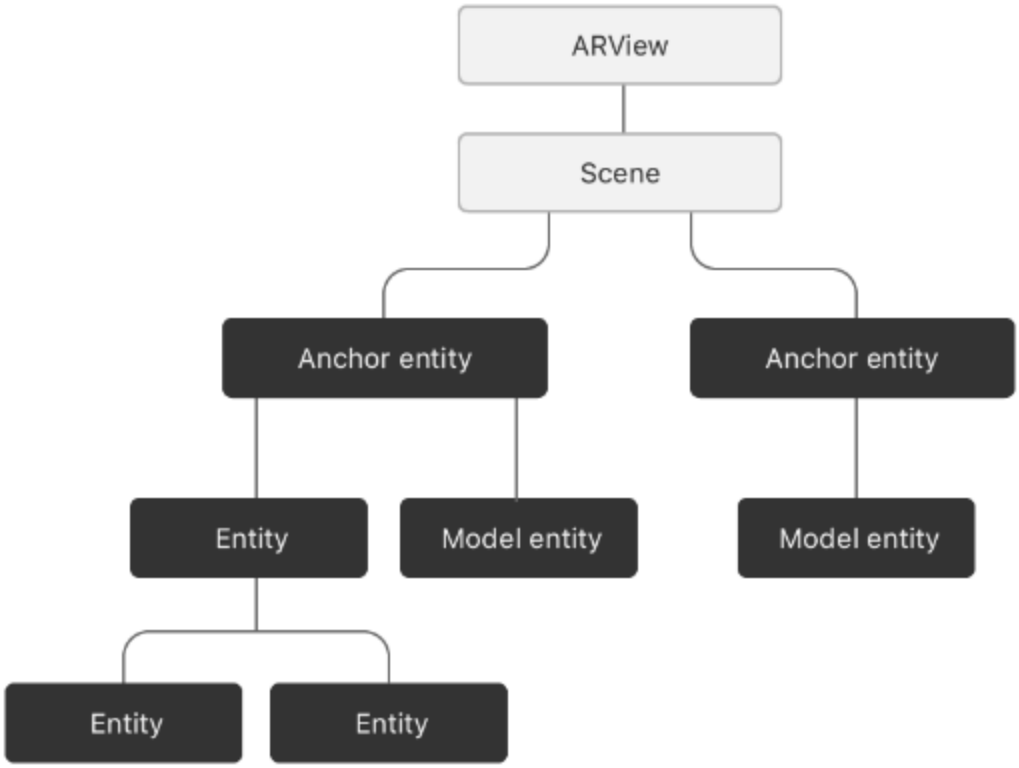
Πρόκειται για ψηφιακούς κόμβους οι οποίοι προσκολλώνται στα πραγματικά αντικείμενα που αντιπροσωπεύουν, ενώ ταυτόχρονα διατηρούν τις πληροφορίες του προσανατολισμού αυτών.

Είναι στην ευχέρεια του προγραμματιστή να προσθέτει κόμβους στη σκηνή πάνω σε αντικείμενα ενδιαφέροντος, επισυνάπτοντάς τα απλώς στο τρέχον αντικείμενο συνεδρίας *ARSession*. Όσο η συνεδρία είναι ενεργή, κάθε αντικείμενο κόμβου *ARAnchor* παραμένει επίσης ενεργό. Εκτός από τη χειροκίνητη μέθοδο, ειδικοί τύποι κόμβων προστίθενται αυτόματα στο αντικείμενο συνεδρίας μέσω της διαδικασίας ιχνηλάτησης που ορίστηκε στο αντικείμενο παραμετροποίησης [22]. Για παράδειγμα, αν η εφαρμογή μας στοχεύει στην ανίχνευση επιφανειών, θα προστεθεί αυτόματα ένα αντικείμενο κόμβου επιφάνειας *ARPlaneAnchor* σε κάθε επιφάνεια που ανιχνεύεται στο χώρο, χωρίς την παρέμβαση του προγραμματιστή [23].

Στην εφαρμογή μας χρησιμοποιούμε και τα δύο είδη κόμβων, αυτόματους και μη. Κατά την ανίχνευση επιφανειών στην αρχή της εμπειρίας, προστίθενται αυτόματα αντικείμενα κόμβων επιφανειών *ARPlaneAnchor* για κάθε ανιχνευμένη επιφάνεια. Όταν ο χρήστης επιλέξει την επιφάνεια που θέλει να χρησιμοποιήσει ως βάση της τοπολογίας, διαγράφουμε τους υπόλοιπους κόμβους επιφάνειας που δημιουργήθηκαν αυτόματα και διακόπτουμε την ανίχνευση επιφανειών, διατηρώντας το αντικείμενο κόμβου που αντιστοιχεί στην επιλεγμένη επιφάνεια. Μόλις ο χρήστης τοποθετήσει την τοπολογία σε μία θέση της επιφάνειας, σημειώνουμε τη θέση αυτή δημιουργώντας χειροκίνητα έναν κόμβο *ARAnchor* και τον προσθέτουμε στην τρέχουσα συνεδρία.

2.3.3 Σύστημα οντοτήτων (Entity System)

Ένα από τα πιο σημαντικά χαρακτηριστικά που προσφέρει το RealityKit για την αναπαράσταση τρισδιάστατων μοντέλων είναι οι οντότητες και το σύστημα οντοτήτων μέσω της κλάσης *Entity* [24]. Οι οντότητες αυτές προστίθενται στη σκηνή επαυξημένης πραγματικότητας και μπορούν να έχουν διαφορετικούς ρόλους.

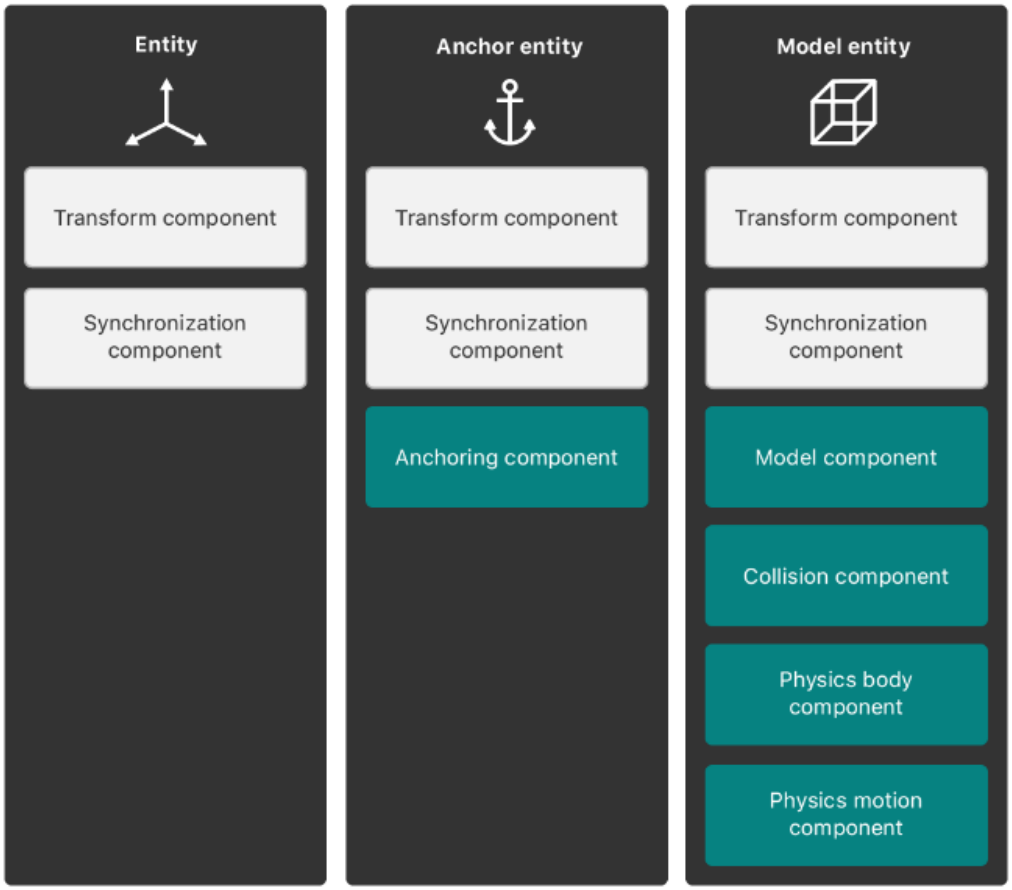


Σχήμα 2.4: Το σύστημα οντοτήτων σε μία σκηνή επαυξημένης πραγματικότητας (5.).

Αρχικά, είναι σημαντικό να δημιουργήσουμε οντότητες-κόμβους με την κλάση *AnchorEntity* [26], οι οποίες αντιπροσωπεύουν σημεία του πραγματικού χώρου. Ο ρόλος τους μοιάζει πολύ με αυτόν των κόμβων (*ARAnchor*) που είδαμε προηγουμένως, καθώς εισάγοντας οντότητες-κόμβους στη σκηνή μπορούμε να τοποθετήσουμε άλλες οντότητες ως παιδιά τους, σε θέσεις σχετικές με αυτές των οντοτήτων-κόμβων. Στην εφαρμογή μας χρησιμοποιούμε τις οντότητες-κόμβους για να αναπαραστήσουμε τη θέση στην επιφάνεια όπου θα τοποθετηθεί η τοπολογία. Από τη στιγμή που ορίσουμε τη θέση αυτή, η τοποθέτηση των φορτίων και των υπόλοιπων μοντέλων είναι πολύ εύκολη καθώς γίνεται με βάση την οντότητα-κόμβο. Για την αναπαράσταση των τρισδιάστατων μοντέλων συγκεκριμένα χρησιμοποιούμε την κλάση *ModelEntity* [25], και προσθέτουμε τις οντότητες-μοντέλα ως παιδιά στις οντότητες-κόμβους. Όλες οι οντότητες κληρονομούν ορισμένα συστατικά από την βασική κλάση *Entity*. Τα συστατικά αυτά είναι:

- Το συστατικό τοποθέτησης *TransformComponent*, το οποίο αφορά τη θέση και τον προσανατολισμό της οντότητας
- Το συστατικό συγχρονισμού *SynchronizationComponent*, το οποίο αφορά το συγχρονισμό μεταξύ των οντοτήτων της σκηνής

Οντότητες που ανήκουν σε πιο ειδικές κατηγορίες περιέχουν επιπρόσθετα συστατικά, όπως φαίνεται στο επόμενο σχήμα.

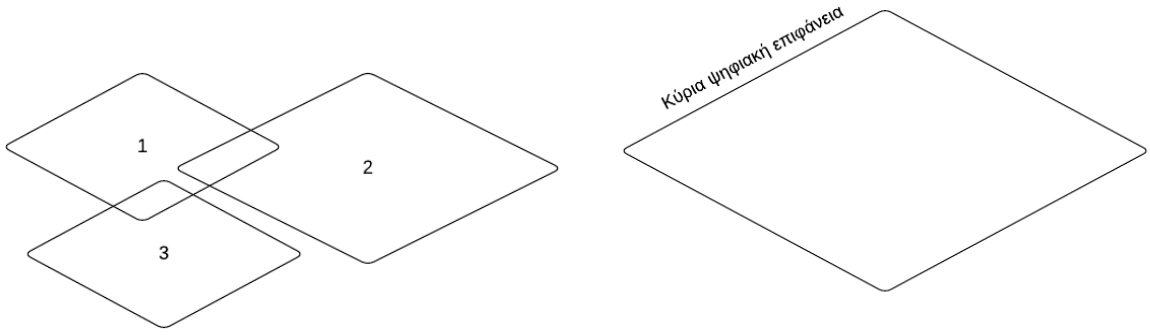


Σχήμα 2.5: Τα συστατικά που περιέχονται σε κάθε βασική κατηγορία οντοτήτων (6.).

Είναι εφικτό να κατασκευάσουμε και τα δικά μας είδη οντοτήτων, όπως και κάνουμε στην εφαρμογή μας για την αναπαράσταση των τρισδιάστατων μοντέλων των φορτίων, των δυνάμεων και των δεικτών αποστάσεων. Συνοπτικά, δημιουργούμε αρχικά μία απλή οντότητα. Έπειτα, δημιουργούμε ένα συστατικό μοντελοποίησης στη μορφή που επιθυμούμε, όπως για παράδειγμα σφαιρικό για την αναπαράσταση φορτίων, και το προσθέτουμε στην οντότητα. Θα δούμε σε βάθος την υλοποίηση αυτής της μεθόδου δημιουργίας οντοτήτων στο κεφάλαιο υλοποίησης της εφαρμογής. Σημειώνεται, ότι στην περίπτωση των οντοτήτων των δυνάμεων, επιλέγουμε το πρόγραμμα Reality Composer [16] [17] για τη δημιουργία του μοντέλου της κεφαλής του βέλους, και στη συνέχεια προσκολλάται στο μοντέλο του υπόλοιπου σώματος όπως θα δούμε και στη συνέχεια.

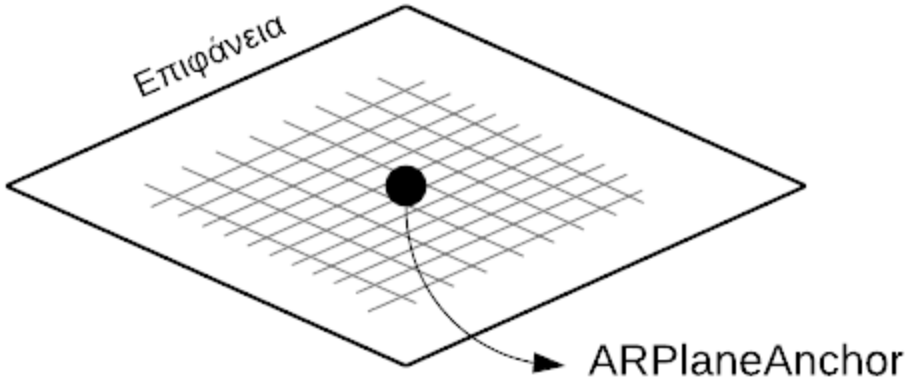
2.3.4 Ανίχνευση επιφανειών (Plane Detection)

Η ανίχνευση επιφανειών παρέχει στον προγραμματιστή τη δυνατότητα ανίχνευσης οριζόντιων ή κάθετων επιφανειών, όπως το έδαφος ή μία πόρτα [28]. Η ανίχνευση πραγματοποιείται με συνεχή ανάλυση των αλλεπάλληλων στιγμιοτύπων που αποδίδει η κάμερα της συσκευής. Συγκεκριμένα, το αντικείμενο κάμερας *ARCamera* παράγει αντικείμενα στιγμιότυπου *ARFrame*, τα οποία περιέχουν την πληροφορία της σκηνής τη δεδομένη χρονική στιγμή που ληφθήκανε. Στη συνέχεια, το ARKit τα προσθέτει σε ένα αντικείμενο ψηφιακής επιφάνειας το οποίο ενημερώνεται συνεχώς. Καθώς ο χρήστης κουνάει τη συσκευή στο χώρο, το ARKit ανακαλύπτει περισσότερες λεπτομέρειες για το χώρο και τις επιφάνειες που βρίσκονται σε αυτόν. Καθώς προσθέτει τα δεδομένα, αν ανιχθευθούν πολλές ψηφιακές επιφάνειες που παρεμβάλλονται μεταξύ τους, το ARKit τις αντιλαμβάνεται και τις ενώνει σε μία κύρια.



Σχήμα 2.6: Η συγχώνευση τριών διαφορετικών ψηφιακών επιφανειών σε μία κύρια.

Η ανιχνευμένη επιφάνεια πλέον αναπαρίσταται από μία κύρια ψηφιακή επιφάνεια, η οποία με τη σειρά της αναπαρίσταται από ένα αντικείμενο *ARPlaneAnchor* [23]. Το αντικείμενο αυτό περιέχει όλες τις πληροφορίες που αφορούν την επιφάνεια. Καθώς ο χρήστης εξακολουθεί να εξερευνεί το χώρο με την κάμερα του κινητού, η έκταση της ψηφιακής επιφάνειας συνεχίζει να ενημερώνεται.



Σχήμα 2.7: Το αντικείμενο ARPlaneAnchor που αντιστοιχεί σε μία ανιχνευμένη επιφάνεια.

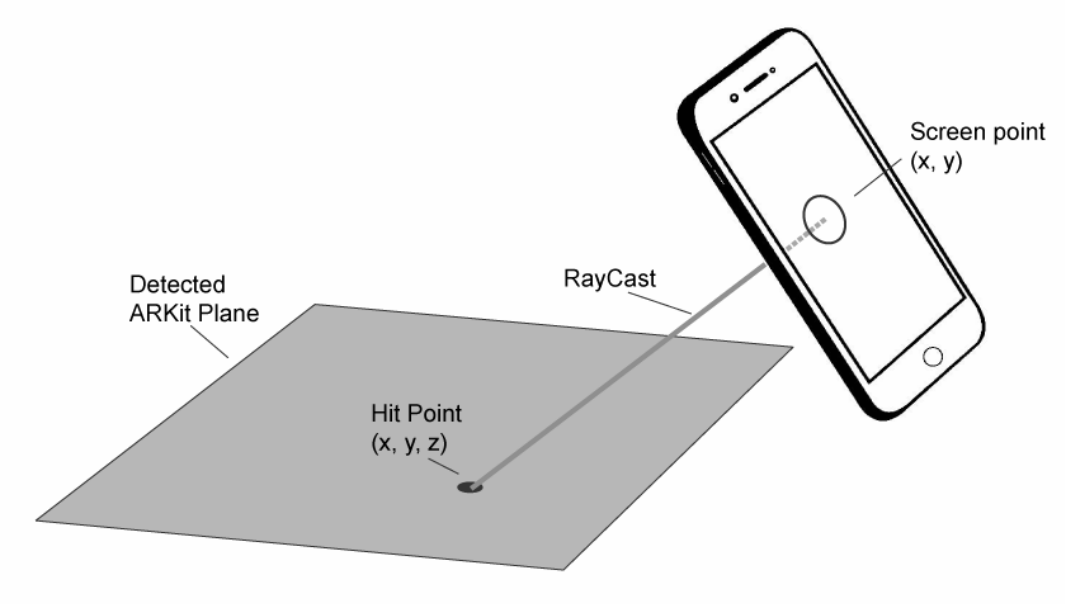
Στην εφαρμογή AR Coulomb χρησιμοποιούμε επιφάνειες για να τοποθετήσουμε μία τοπολογία μίας αναπαράστασης του νόμου Coulomb. Αρχικά, ο χρήστης καλείται να επιλέξει μεταξύ διαφορετικών επιλογών επιφανειών που ανιχνεύει η συσκευή στο χώρο. Στη συνέχεια, τα φορτία της αναπαράστασης τοποθετούνται όλα στο ίδιο επίπεδο, πάνω στην επιφάνεια που επέλεξε ο χρήστης. Επομένως, χρησιμοποιούμε την ανίχνευση επιφανειών για την εξερεύνηση επιφανειών του χρήστη κατά το πρώτο στάδιο λειτουργίας της εφαρμογής.

2.3.5 Προβολή πατήματος του χρήστη από την οθόνη στον τρισδιάστατο χώρο (Hit Testing)

Το "Hit Testing" επιτρέπει στο χρήστη να επιλέξει ένα τρισδιάστατο σημείο του χώρου μέσω ενός πατήματος σε ένα δισδιάστατο σημείο της οθόνης [31]. Κατά τη διάρκεια της διαδικασίας αυτής, το ARKit χρησιμοποιεί όλα τα δεδομένα της σκηνής που είναι διαθέσιμα, όπως ανιχνευμένες επιφάνειες και σημεία ενδιαφέροντος (feature points). Όταν ο χρήστης πατήσει πάνω στην οθόνη, η συσκευή στέλνει ακτίνες ώστε να βρει τομές με τα παραπάνω δεδομένα, και επιστρέφει έναν πίνακα που περιέχει όλα τα σημεία τομής που βρήκε ταξινομημένα με βάση την απόσταση, έτσι ώστε το πρώτο στοιχείο του πίνακα να αντιστοιχεί στο πιο κοντινό σημείο τομής από την κάμερα.

Η χρήση των ακτίνων είναι πολύ απλή. Όταν ακουμπήσει την οθόνη ο χρήστης ορίζεται ένα αντικείμενο *CGPoint* το οποίο αντιπροσωπεύει τις κανονικοποιημένες συντεταγμένες του σημείου αφής πάνω στην οθόνη. Για παράδειγμα, το αντικείμενο *CGPoint* που αντιστοιχεί στο πάνω

αριστερά σημείο της οθόνης έχει τιμή (0, 0) και το αντίστοιχο για το κάτω δεξιά σημείο έχει τιμή (1, 1). Επομένως αν ο χρήστης πατήσει στο κέντρο της οθόνης, θα σταλεί μία ακτίνα με αντικείμενο *CGPoint* τιμής (0.5, 0.5).



Εικόνα 2.2: Οπτική αναπαράσταση ενός hit test. Η ακτίνα ξεκινάει από ένα συγκεκριμένο σημείο της οθόνης και πέφτει πάνω σε ένα τρισδιάστατο σημείο της επιφάνειας (7.).

Υπάρχουν τέσσερις διαφορετικές μέθοδοι με τις οποίες πραγματοποιείται το Hit Testing, από τις οποίες ο προγραμματιστής μπορεί να επιλέξει ποια θα χρησιμοποιήσει στην εφαρμογή του.

1. Σε περίπτωση που θέλει να χρησιμοποιήσει την ανιχνευμένη επιφάνεια κι όχι κάποια επέκτασή της, το hit test θα επιστρέψει τομές όσων ακτινών πέσανε πάνω στην επιφάνεια και όχι έξω από τα σύνορά της.

2. Στην περίπτωση που επιζητά μεγαλύτερη ελευθερία κινήσεων ή αν θεωρήσει ότι η ανιχνευμένη επιφάνεια είναι μικρή, το ARKit πρέπει να αγνοήσει τα όρια της επιφάνειας και να θεωρήσει ότι η επιφάνεια έχει άπειρη έκταση. Τότε κατά το hit test ακόμα κι οι ακτίνες που πέφτουν στην επέκταση της επιφάνειας κι όχι μόνο στην έκτασή της, θα επιστρέψουν τομές.

3. Αν η εφαρμογή δεν χρησιμοποιεί ανίχνευση επιφανειών, ο προγραμματιστής μπορεί να δημιουργήσει την εκτίμηση μιας ψηφιακής επιφάνειας βασισμένη σε τρισδιάστατα σημεία ενδιαφέροντος που έχουν αποθηκευτεί. Το hit test τότε θα επιστρέψει σημεία τομής που προέρχονται από αυτή την επιφάνεια.

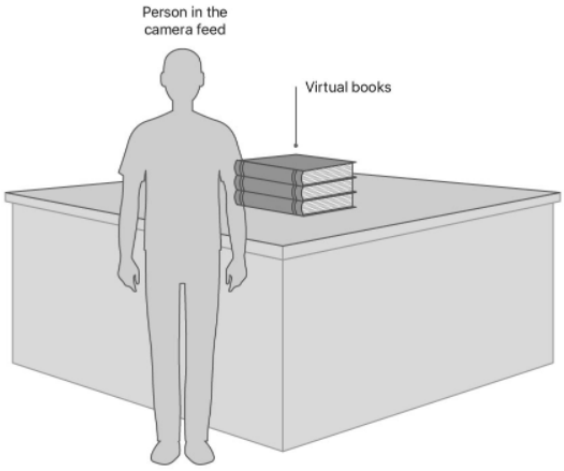
4. Τέλος, υπάρχει η περίπτωση όπου το ARKit αδυνατεί να δημιουργήσει μία ψηφιακή επιφάνεια, είτε επειδή ο χρήστης προσπαθεί να αλληλεπιδράσει με μία πολύ μικρή

επιφάνεια, είτε το περιβάλλον γύρω του είναι πολύ ακανόνιστο. Τότε, δίνεται η δυνατότητα αλληλεπίδρασης απευθείας με τα σημεία ενδιαφέροντος της σκηνής. Κατά το hit test, επιστρέφονται ως σημεία τομής τα σημεία των ακτινών που βρίσκονται πιο κοντά στα σημεία ενδιαφέροντος που θέλει ο χρήστης.

Στην εφαρμογή μας υλοποιούμε το Hit Testing σε ανιχνευμένες επιφάνειες χωρίς να χρησιμοποιούμε κάποια επέκτασή τους (μέθοδος 1) και το χρησιμοποιούμε σε δύο διαφορετικές περιπτώσεις. Η πρώτη αφορά το άγγιγμα του χρήστη στην επιφάνεια, όταν επιθυμεί να τοποθετήσει μία τοπολογία σε αυτήν. Καθώς ο χρήστης εξερευνά το χώρο, το ARKit ανιχνεύει διάφορες επιφάνειες γύρω του. Όταν θελήσει να τοποθετήσει μία τοπολογία σε μία από αυτές, πρέπει να πατήσει πάνω της και όχι σε οποιοδήποτε σημείο της οθόνης. Εκτός από την επιλογή επιφάνειας για την τοποθέτηση τοπολογίας, χρησιμοποιούμε το "hit testing" σε μετέπειτα στάδιο της εφαρμογής, όταν ο χρήστης προσπαθεί να αλληλεπιδράσει με τα φορτία της τοπολογίας. Συγκεκριμένα, θέλουμε όταν ο χρήστης πατάει ή σέρνει ένα από αυτά, να το επιβεβαιώνουμε και να το εκλέγουμε ως "επιλεγμένο φορτίο". Ο τρόπος με τον οποίο γίνεται η επαλήθευση είναι μέσω του Hit Testing στην οντότητα που άγγιξε ο χρήστης.

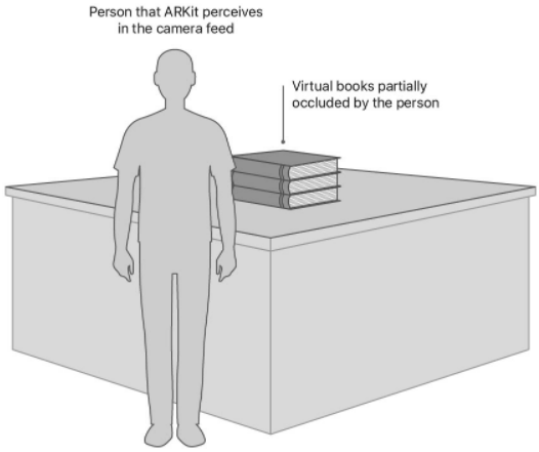
2.3.6 Ανθρώπινη φραγή (People Occlusion)

Η ανθρώπινη φραγή αφορά τον τρόπο με τον οποίο τα ψηφιακά μοντέλα μίας εφαρμογής επαυξημένης πραγματικότητας αλληλεπιδρούν με τα ανθρώπινα σώματα που επεμβαίνουν στη σκηνή [32]. Έστω ότι ο χρήστης Α τοποθετεί μία ψηφιακή στοίβα από βιβλία στο γραφείο του μέσω μίας τέτοιας εφαρμογής. Αν ένα δεύτερο άτομο Β περάσει ανάμεσα από το κινητό του χρήστη και το τραπέζι, τότε το φυσιολογικό είναι τα βιβλία να κρυφτούν πίσω από το άτομο Β. Παρόλα αυτά, η προβολή των βιβλίων δεν φράζεται από την ανθρώπινη φιγούρα, εξακολουθεί να προβάλλεται καθώς το άτομο Β περνάει από μπροστά του και χαλάει την ρεαλιστική εμπειρία του χρήστη Α. Αυτό συμβαίνει διότι το ψηφιακό περιεχόμενο αποδίδεται ένα επίπεδο πιο πάνω από αυτό στο οποίο ανήκει η εικόνα της κάμερας. Προφανώς δημιουργούνται προβλήματα όταν στην εικόνα της κάμερας ανήκει και μία ανθρώπινη φιγούρα που περνάει μπροστά από το ψηφιακό περιεχόμενο.



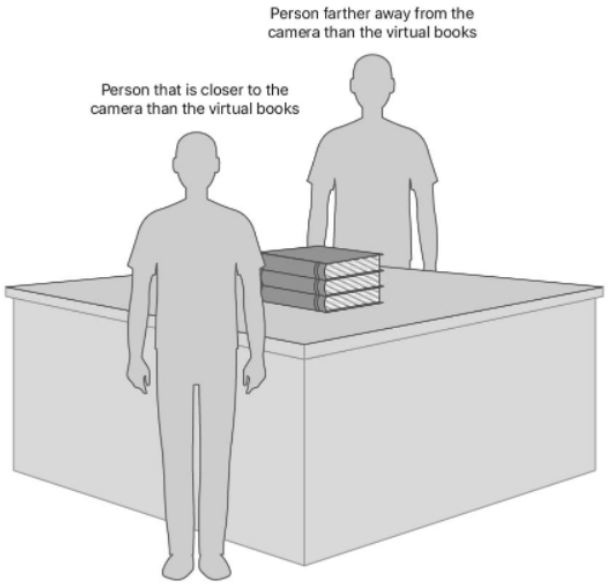
Εικόνα 2.3: Το ψηφιακό αντικείμενο αποδίδεται ένα επίπεδο πάνω από την εικόνα της κάμερας, με αποτέλεσμα να αποδίδεται λανθασμένα μπροστά από τον άνθρωπο (8.).

Η ανθρώπινη φραγή διατίθεται μέσω του ARKit στο χρήστη και λύνει ακριβώς αυτό το πρόβλημα απόδοσης ψηφιακών αντικειμένων όταν παρεμβάλλεται ανθρώπινη φιγούρα. Ο τρόπος με τον οποίο αντιμετωπίζεται η ανθρώπινη παρεμβολή είναι η αναγνώριση ανθρώπινων σωμάτων μέσω αλγορίθμων μηχανικής εκμάθησης μέσα στο πλαίσιο της εικόνας. Στη συνέχεια, δημιουργείται ένα ξεχωριστό επίπεδο το οποίο περιέχει μόνο τα pixels που περιέχουν την εικόνα του ανθρώπου. Η διαδικασία αυτή ονομάζεται κατάτμηση (segmentation). Το ξεχωριστό αυτό επίπεδο που περιέχει μόνο την ανθρώπινη φιγούρα αποδίδεται τελικά πάνω από όλα τα υπόλοιπα.



Εικόνα 2.4: Με τη χρήση της ανθρώπινης φραγής, τα ψηφιακά βιβλία αποδίδονται σωστά πίσω από την ανθρώπινη φιγούρα (9.).

Σημειώνεται ότι οι συγκεκριμένοι αλγόριθμοι μηχανικής εκμάθησης λαμβάνουν υπόψη την απόσταση από την κάμερα, τόσο των ψηφιακών αντικειμένων, όσο και των ανιχνευμένων ανθρώπων στη σκηνή. Για την γρήγορη και αποτελεσματική εκτίμηση της απόστασης, το ARKit χρησιμοποιεί τη Νευρωνική Μηχανή της Apple (Apple Neural Engine). Επιπλέον, είναι εφικτό να εντοπιστούν περισσότερες από μία ανθρώπινες φιγούρες στη σκηνή, καθώς και μέρη αυτών όπως ένα χέρι. Επομένως αν στο προηγούμενο πρόβλημα περάσει και ένα τρίτο άτομο Γ πίσω από το τραπέζι, το ARKit δε θα αποδώσει την μορφή του σε ένα ξεχωριστό επίπεδο μπροστά από τα υπόλοιπα, αλλά θα παραμείνει στο επίπεδο της εικόνας της κάμερας, κάτω από το επίπεδο του ψηφιακού μοντέλου των βιβλίων.



Εικόνα 2.5: Με τη χρήση της ανθρώπινης φραγής λαμβάνοντας υπόψη το βάθος, τα ψηφιακά βιβλία αποδίδονται σωστά ανάμεσα στις δύο ανθρώπινες φιγούρες (10.).

3. Η προτεινόμενη εφαρμογή AR Coulomb

Η τεχνολογία στην τάξη κάνει όλο και περισσότερο αισθητή την παρουσία της, καθώς συσκευές tablet αντικαθιστούν βιβλία και εγχειρίδια, ενώ ο καθένας πλέον μπορεί να αναζητήσει οποιαδήποτε πληροφορία θέλει μέσω του κινητού του τηλεφώνου. Οι μαθητές καλούνται πλέον να γνωρίζουν πως να χειρίζονται διάφορα εργαλεία στον υπολογιστή ή στο κινητό, ενώ οι εκπαιδευτικοί έχουν τη δυνατότητα να εισάγουν όλο και πιο ενδιαφέρουσες εργασίες στην τάξη μέσω έξυπνων συσκευών ώστε να κινήσουν το ενδιαφέρον των μαθητών. Επομένως παρατηρούμε ότι η τεχνολογία είναι σε θέση να βοηθήσει τον μαθητή να εξελιχθεί με το δικό του ρυθμό και να κατανοήσει καλύτερα το αντικείμενο με το οποίο ασχολείται.

Ένα τέτοιο αντικείμενο είναι το μάθημα της Φυσικής, το οποίο συνδυάζει τη θεωρία του φυσικού κόσμου και μικρόκοσμου με την άλγεβρα και τη γεωμετρία. Η ανάλυση φυσικών μεγεθών και η επίλυση ασκήσεων απαιτεί γνώσεις γεωμετρίας αλλά και σχεδιασμό αντικειμένων, τα οποία ο μαθητής ίσως να μη δει ποτέ στη ζωή του σε πραγματική κλίμακα. Ο συνδυασμός των παραπάνω απαιτήσεων μπορεί να δυσκολέψει τους μαθητές να κατανοήσουν την εφαρμογή μαθηματικών τύπων και τη συμπεριφορά των μεγεθών σε μία τοπολογία άσκησης φυσικής. Καθηγητής και μαθητής θα μπορούσαν να επωφεληθούν από μία εφαρμογή η οποία θα λειτουργεί ως εργαλείο ζωντανής τρισδιάστατης απεικόνισης των φυσικών στοιχείων και των μεγεθών τους με τη χρήση τεχνολογίας επαυξημένης πραγματικότητας.

Στα πλαίσια της διπλωματικής εργασίας αναπτύξαμε μία εφαρμογή επαυξημένης πραγματικότητας η οποία αποσκοπεί στην τρισδιάστατη αναπαράσταση του νόμου του Coulomb, καθώς είναι ένα από τα πολλά κεφάλαια με το οποίο έρχονται αντιμέτωποι οι μαθητές στο σχολείο. Με την εφαρμογή αυτή αναδεικνύουμε τη χρήση της τεχνολογίας επαυξημένης πραγματικότητας μέσω του εργαλείου ARKit, ενώ ταυτόχρονα αντιμετωπίζουμε το πρόβλημα της ζωντανής ενημέρωσης των τρισδιάστατων μοντέλων.

3.1 Ανάλυση απαιτήσεων της εφαρμογής

Για την αναπαράσταση μίας τοπολογίας άσκησης Coulomb δημιουργήσαμε τρισδιάστατα μοντέλα για τα φορτία, τα διανύσματα των δυνάμεων, τους δείκτες των αποστάσεων μεταξύ των φορτίων καθώς και για τις επιγραφές των διαφόρων μεγεθών. Ο χρήστης είναι ελεύθερος να αλληλεπιδράσει με τα φορτία και τις τιμές τους επομένως τα μοντέλα δυνάμεων και αποστάσεων πρέπει να είναι σε θέση να ενημερώνονται καθόλη τη διάρκεια λειτουργίας της εφαρμογής.

Κύριο μέλημά μας είναι η αναπαράσταση του νόμου Coulomb να πραγματοποιείται εύκολα, χωρίς περίπλοκες ρυθμίσεις, και να απεικονίζεται ξεκάθαρα στην οθόνη χωρίς περιττές λεπτομέρειες, ώστε να είναι όσο το δυνατόν πιο φιλική προς το χρήστη γίνεται. Επομένως η εφαρμογή μας θα πρέπει να πληροί τις παρακάτω βασικές προδιαγραφές:

- Δυνατότητα επανεκκίνησης της εμπειρίας ώστε ο χρήστης να έχει πλήρη έλεγχο του κύκλου απεικόνισης που πραγματοποιεί.
- Δυνατότητα ανάκτησης της εμπειρίας μετά από μεταφορά της εφαρμογής στο παρασκήνιο (background) λόγω κάποιας διακοπής, όπως μία τηλεφωνική κλήση.
- Δυνατότητα αποθήκευσης της τοπολογίας πάνω στην οποία εργάζεται ο χρήστης και απεικονίζεται στην οθόνη. Για την αποθήκευση θα καταχωρείται στη μνήμη της συσκευής μία φωτογραφία της τρέχουσας οθόνης όπου απεικονίζεται η τοπολογία, καθώς επίσης και ένας τίτλος, περιγραφή και τα στοιχεία της.
- Δυνατότητα ο χρήστης να επιλέξει μεταξύ αποθηκευμένων τοπολογιών, να τις επεξεργαστεί καθώς και δυνατότητα να δημιουργήσει μία από την αρχή.
- Ελευθερία κίνησης των στοιχείων, καθώς και αλλαγής τιμών των μεγεθών τους. Ο χρήστης θα πρέπει να είναι σε θέση να δοκιμάσει οποιαδήποτε διάταξη θελήσει.

3.1.1 Η αλληλεπίδραση με οδηγίες, μενού και τρισδιάστατα μοντέλα

Κατά τη διάρκεια χρήσης της εφαρμογής, ο χρήστης δέχεται διάφορα βοηθητικά μηνύματα τα οποία τον καθοδηγούν ανάλογα με την ενέργεια που προσπαθεί να εκτελέσει. Τα μηνύματα αυτά πρέπει να περιέχουν απλές εντολές, και να συνοδεύονται με κάποιο τρισδιάστατο μοντέλο όπου αυτό είναι εφικτό.

Ο χρήστης αρχικά καλείται να κουνήσει τη συσκευή με συγκεκριμένο τρόπο απέναντι από κάποια επιφάνεια ώστε η εφαρμογή να την αναγνωρίσει. Υπάρχουν δύο είδη επιφανειών που είναι σε θέση να αναγνωρίσει η εφαρμογή, οριζόντιες και κάθετες. Αφού αναγνωριστεί μία επιφάνεια, η εφαρμογή πρέπει να υποδείξει στο χρήστη ποια είναι αυτή. Αυτό γίνεται εύκολα με ένα τρισδιάστατο δείκτη τοποθέτησης. Το μοντέλο του δείκτη θα πρέπει να είναι παράλληλο με την επιφάνεια και το κέντρο του να αντιστοιχεί στο κέντρο της οθόνης του χρήστη. Όσο ο χρήστης εξακολουθεί να κουνάει τη συσκευή, ο δείκτης τοποθέτησης κινείται παράλληλα στην επιφάνεια που έχει απέναντί του.

Στη συνέχεια αρκεί ο χρήστης να πατήσει στην οθόνη ώστε να τοποθετήσει μία τοπολογία στη θέση του δείκτη τοποθέτησης. Για να μπορεί να επιλέξει ο χρήστης μία τοπολογία απαιτείται ένα μενού επιλογής το οποίο θα παρουσιάζει τις αποθηκευμένες τοπολογίες. Η αναπαράσταση των τοπολογιών στο μενού θα πρέπει να είναι σε τέτοια μορφή ώστε ο χρήστης να τις ξεχωρίζει εύκολα. Από τη στιγμή που ο χρήστης επιλέξει μία τοπολογία θα πρέπει αυτή να εμφανιστεί αμέσως στη θέση του δείκτη τοποθέτησης, ο οποίος προφανώς θα πρέπει να πάψει να υπάρχει. Μόλις η τοπολογία είναι ορατή, ο χρήστης θα πρέπει να είναι σε θέση να τη μεταβάλλει όπως επιθυμεί. Αυτό περιλαμβάνει κινήσεις των φορτίων, αλλαγές στη τιμή τους, προσθήκη νέων φορτίων αλλά και διαγραφή. Για αυτές τις ενέργειες πρέπει να δίνεται η δυνατότητα

αλληλεπίδρασης με τα τρισδιάστατα μοντέλα, με διαφορετικές μεθόδους όπως πάτημα δακτύλου (tap), παρατεταμένο πάτημα (long tap), σύρσιμο (drag) [34].

3.1.2 Αλληλεπίδραση με το σύστημα αποθήκευσης

Η διαχείριση τοπολογιών μέσα από ένα σύστημα αποθήκευσης είναι απαραίτητη για την χρησιμότητα της εφαρμογής. Με άλλα λόγια, πρέπει να δίνεται η δυνατότητα στο χρήστη να αποθηκεύει τοπολογίες σε μία μονάδα αποθήκευσης της εφαρμογής καθώς και να τις διαγράφει. Τα βήματα για την αποθήκευση θα πρέπει να συμπεριλαμβάνουν τη δημιουργία ενός στιγμιότυπου της τρέχουσας σκηνής καθώς και ενός τίτλου και μίας περιγραφής της, ώστε η τοπολογία να αντιπροσωπεύεται με ικανοποιητική λεπτομέρεια στο μενού επιλογής.

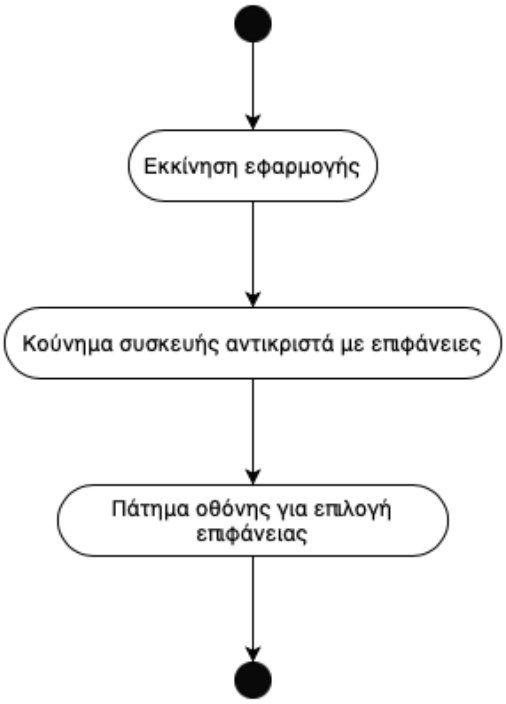
Το μενού επιλογής τοπολογίας θα πρέπει να παρουσιάζει για κάθε μία τόσο το στιγμιότυπο (φωτογραφία) όσο και την περιγραφή της. Επίσης θα πρέπει να παρέχει δύο επιλογές για την κάθε μία, επιλογή ή διαγραφή. Η αποθήκευση και η διαγραφή μίας τοπολογίας θα πραγματοποιούνται στη μνήμη της εφαρμογής, και κατ' επέκταση στη μνήμη της συσκευής. Σε μία μελλοντική υλοποίηση θα μπορούσε ο χώρος αποθήκευσης να είναι ένας απομακρυσμένος διακομιστής και η σύνδεση να γίνεται μέσω διαδικτύου.

3.2 Σενάρια χρήσης

Αναζήτηση και επιλογή επιφάνειας

Χρήστης	Εφαρμογή
Εκκίνηση εφαρμογής	
	Εκκίνηση ανίχνευσης επιφανειών
	Εκκίνηση βοηθητικών μηνυμάτων (coaching overlay)
Κούνημα συσκευής αντικριστά με επιφάνειες	
Πάτημα οθόνης για επιλογή επιφάνειας	

Πίνακας 3.1

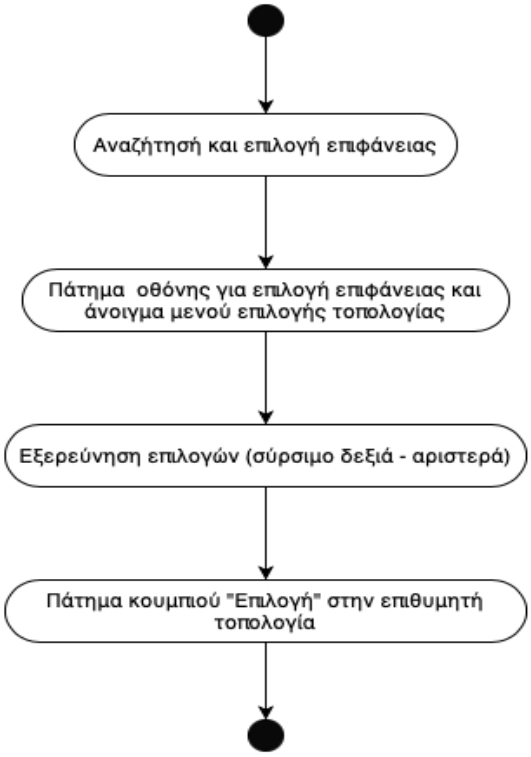


Σχήμα 3.1: Διάγραμμα ροής για επιλογή επιφάνειας.

Τοποθέτηση τοπολογίας

Χρήστης	Εφαρμογή
Εκκίνηση εφαρμογής	
	Φόρτωση δεδομένων εφαρμογής (αποθηκευμένες τοπολογίες)
	Εκκίνηση ανίχνευσης επιφανειών
	Εκκίνηση βοηθητικών μηνυμάτων (coaching overlay)
Κούνημα συσκευής αντικριστά με επιφάνειες	
Πάτημα οθόνης για επιλογή επιφάνειας	
	Φόρτωση μενού επιλογής τοπολογίας
Εξερεύνηση επιλογών του μενού	
Πάτημα κουμπιού “Επιλογή” για την επιλογή της τρέχουσας προβεβλημένης τοπολογίας του μενού	
	Δημιουργία φορτίων, αποστάσεων και μεγεθών της επιλεγμένης τοπολογίας
	Τοποθέτηση της τοπολογίας στην επιλεγμένη επιφάνεια

Πίνακας 3.2



Σχήμα 3.2: Διάγραμμα ροής για την τοποθέτηση τοπολογίας

Προσθήκη φορτίου

Χρήστης	**Εφαρμογή**
Αναζήτηση και επιλογή επιφάνειας (σενάριο 1)	
Επιλογή και τοποθέτηση τοπολογίας (σενάριο 2)	
Πάτημα κουμπιού “Προσθήκη”	
	Επιλογή μίας τυχαίας τοποθεσίας στην τρέχουσα τοπολογία
	Δημιουργία ενός νέου φορτίου
	Τοποθέτηση του νέου φορτίου στην επιλεγμένη τυχαία τοποθεσία
	Άνοιγμα μενού διαχείρισης τιμής του νέου

	φορτίου

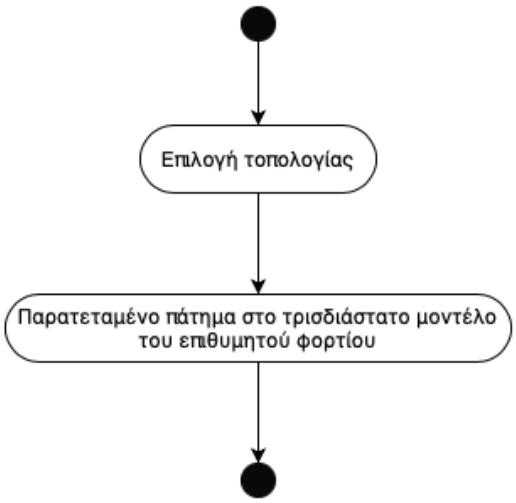
Πίνακας 3.3

Σχήμα 3.3: Διάγραμμα ροής για την προσθήκη ενός φορτίου.

Επιλογή φορτίου

Χρήστης	Εφαρμογή
Επιλογή και τοποθέτηση τοπολογίας (σενάριο 2)	
Παρατεταμένο πάτημα πάνω στο μοντέλο ενός φορτίου	
	Η εφαρμογή το θέτει ως τρέχον επιλεγμένο φορτίο
	Άνοιγμα μενού διαχείρισης τιμής του επιλεγμένου φορτίου

Πίνακας 3.4

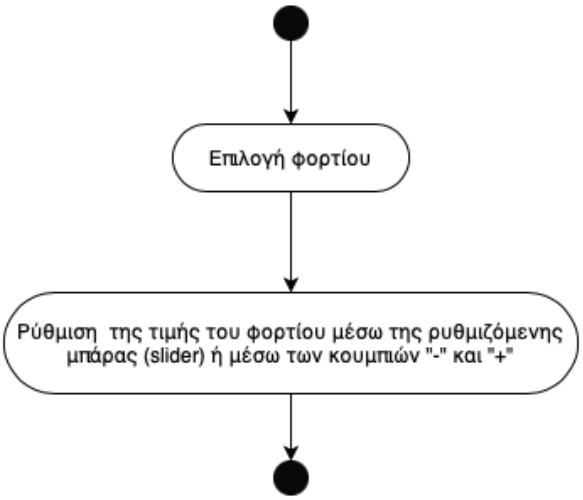


Σχήμα 3.4: Διάγραμμα ροής για την επιλογή ενός φορτίου.

Επεξεργασία τιμής φορτίου

Χρήστης	Εφαρμογή
Επιλογή και τοποθέτηση τοπολογίας (σενάριο 2)	
Παρατεταμένο πάτημα πάνω στο μοντέλο ενός φορτίου	
	Η εφαρμογή το θέτει ως τρέχον επιλεγμένο φορτίο
	Άνοιγμα μενού διαχείρισης τιμής του επιλεγμένου φορτίου
Επιλογή τιμής μέσω του slider	
	Ενημέρωση τιμής του φορτίου και επανυπολογισμός υπόλοιπων μεγεθών
	Ενημέρωση τρισδιάστατων και δισδιάστατων μοντέλων αναπαράστασης

Πίνακας 3.5



Σχήμα 3.5: Διάγραμμα ροής για την επεξεργασία τιμής ενός φορτίου.

Διαγραφή φορτίου

Χρήστης	**Εφαρμογή**
Επιλογή και τοποθέτηση τοπολογίας (σενάριο 2)	
Παρατεταμένο πάτημα πάνω στο μοντέλο ενός φορτίου	
	Η εφαρμογή το θέτει ως τρέχον επιλεγμένο φορτίο
	Άνοιγμα μενού διαχείρισης τιμής του επιλεγμένου φορτίου
Πάτημα κουμπιού “Διαγραφή”	
	Εμφάνιση ειδοποίησης επιβεβαίωσης διαγραφής στο χρήστη (“ΝΑΙ”/”ΟΧΙ”)
Πάτημα στην επιλογή “ΝΑΙ”	
	Διαγραφή φορτίου από την τοπολογία
	Επιλογή τυχαίου φορτίου από τα υπόλοιπα της τοπολογίας (αν υπάρχουν)

	Επανυπολογισμός μεγεθών των φορτίων της τρέχουσας τοπολογίας
	Ενημέρωση τρισδιάστατων και δισδιάστατων μοντέλων αναπαράστασης

Πίνακας 3.6

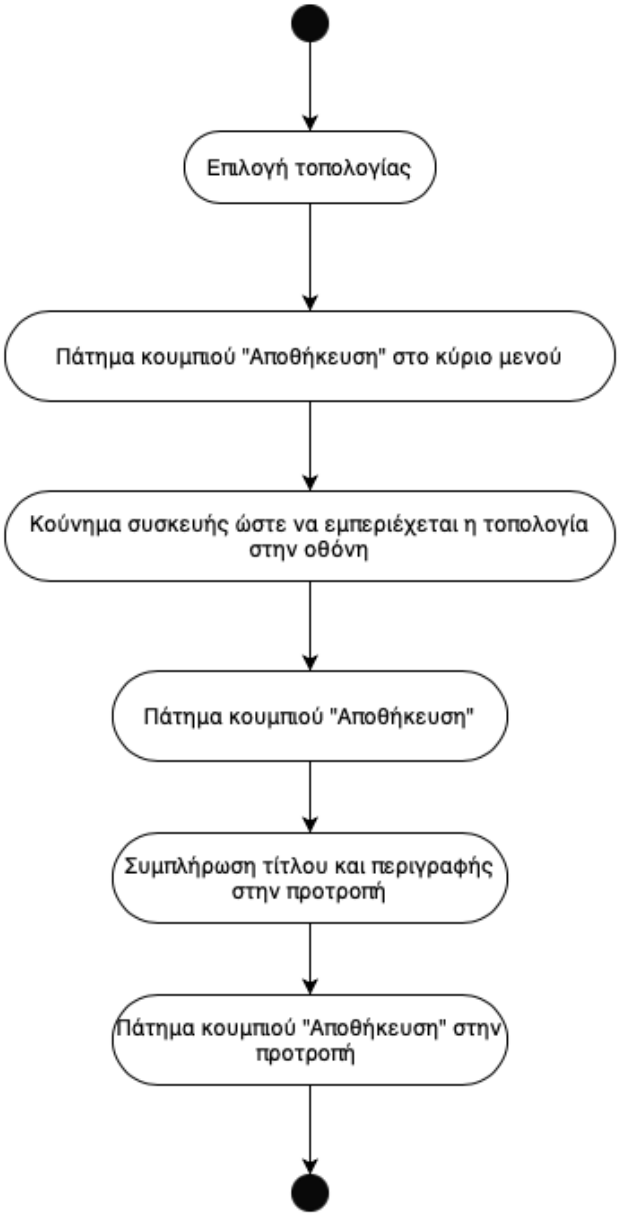
Σχήμα 3.6: Διάγραμμα ροής για τη διαγραφή ενός φορτίου.

Αποθήκευση τοπολογίας

Χρήστης	Εφαρμογή
Επιλογή και τοποθέτηση τοπολογίας (σενάριο 2)	
Πάτημα κουμπιού “Αποθήκευση”	
	Εναλλαγή του κύριου μενού με το μενού αποθήκευσης
Μετακίνηση συσκευής ώστε να	

περιλαμβάνεται η τοπολογία στη σκηνή	
Πάτημα κουμπιού “Αποθήκευση”	
	Δημιουργία στιγμιότυπου σκηνής
	Εμφάνιση φόρμας συμπλήρωσης στοιχείων τοπολογίας
Πληκτρολόγηση στοιχείων	
Πάτημα κουμπιού “Αποθήκευση”	
	Αποθήκευση των στοιχείων της τοπολογίας στη μνήμη της εφαρμογής (περιγραφή, φωτογραφία)

Πίνακας 3.7

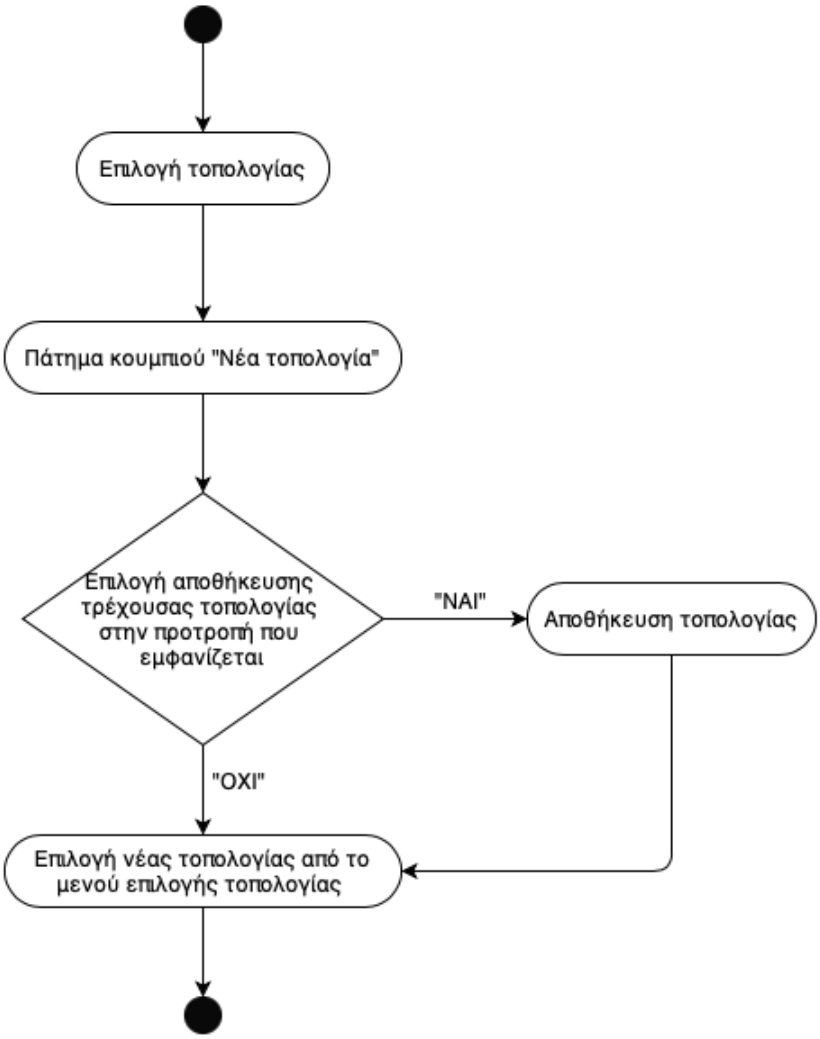


Σχήμα 3.7: Διάγραμμα ροής για την αποθήκευση μίας τοπολογίας.

Επιλογή νέας τοπολογίας

Χρήστης	Εφαρμογή
Επιλογή και τοποθέτηση τοπολογίας (σενάριο 2)	
(Επεξεργασίας της τοπολογίας)	
Πάτημα κουμπιού “Νέα τοπολογία” από το κύριο μενού	
	Εμφάνιση προτροπής αποθήκευσης της τρέχουσας τοπολογίας
Επιλογή “Ναι” και επανάληψη βημάτων αποθήκευσης ή επιλογή “Όχι”	
	Εμφάνιση μενού επιλογής τοπολογίας και επανάληψη βημάτων επιλογής τοπολογίας

Πίνακας 3.8



Σχήμα 3.8: Διάγραμμα ροής για την επιλογή νέας τοπολογίας.

Επανεκκίνηση εμπειρίας

Χρήστης	Εφαρμογή
Πάτημα κουμπιού “Επανεκκίνηση”	
	Διαγραφή τρεχόντων δεδομένων (δεδομένα τρέχουσας τοπολογίας, ανιχνευμένες επιφάνειες, κλπ)
	Εκκίνηση ανίχνευσης επιφανειών

	Εκκίνηση βοηθητικών μηνυμάτων (coaching overlay)

Πίνακας 3.9

Σχήμα 3.9: Διάγραμμα ροής για την επανεκκίνηση της εμπειρίας.

3.3 Σχεδιασμός της εφαρμογής

Ο σχεδιασμός της εφαρμογής εμπεριέχει την δημιουργία των διαφόρων οθονών από τις οποίες θα αποτελείται η εφαρμογή και με τις οποίες θα αλληλεπιδρά ο χρήστης, καθώς επίσης και μία πρώτη αναφορά στις οντότητες του κώδικα όπως κλάσεις, μεταβλητές και μεθόδους, και τον τρόπο με τον οποίο αυτές αλληλεπιδρούν μεταξύ τους.

3.3.1 Οθόνες της εφαρμογής

Για να ανταποκριθούμε στις απαιτήσεις της εφαρμογής θα πρέπει να δημιουργήσουμε ορισμένες οθόνες προκειμένου να καλυφθούν οι διάφορες λειτουργίες που περιγράψαμε προηγουμένως στην ανάλυση. Αυτές είναι οι εξής:

- **Εναρκτήρια οθόνη,** στην οποία η εφαρμογή φορτώνει τα δεδομένα από τη μνήμη της συσκευής και ζητάει άδεια χρήσης της κάμερας από το χρήστη.
- **Οθόνη καθοδήγησης χρήστη (Coaching Overlay View),** όπου ένα κινούμενο γραφικό καθοδηγεί το χρήστη στις κινήσεις που πρέπει να προβεί ώστε η εφαρμογή να αναγνωρίσει επιφάνειες.
- **Οθόνη εξερεύνησης επιφανειών,** όπου η εφαρμογή προβάλει ένα γραφικό δείκτη τοποθέτησης από το κέντρο της οθόνης στο πραγματικό χώρο, ώστε ο χρήστης να αντιλαμβάνεται τις επιφάνειες που έχουν ανιχνευθεί.
- **Οθόνη με το μενού επιλογής τοπολογίας,** όπου ο χρήστης θα μπορεί να εξερευνήσει τις αποθηκευμένες τοπολογίες, να δει την φωτογραφία και περιγραφή για την καθεμία και να επιλέξει ή να διαγράψει τελικά μία.
- **Οθόνη με την αναπαράσταση της επιλεγμένης τοπολογίας στη σκηνή,** όπου η τοπολογία που έχει επιλέξει ο χρήστης προβάλλεται στη σκηνή στην επιλεγμένη επιφάνεια και όπου ο χρήστης είναι σε θέση να επεξεργαστεί
- **Οθόνη με το μενού αποθήκευσης,** όπου ο χρήστης καλείται να κουνήσει τη συσκευή ώστε η σκηνή να περιλαμβάνει ολόκληρη την τοπολογία, και στη συνέχεια να προχωρήσει στην αποθήκευσή της.
- **Οθόνη με την προτροπή αποθήκευσης στοιχείων τοπολογίας,** όπου ο χρήστης καλείται να πληκτρολογήσει έναν τίτλο και μία περιγραφή για την τρέχουσα τοπολογία και στη συνέχεια να επιβεβαιώσει ή όχι την αποθήκευσή της.
- **Οθόνη με το μενού επεξεργασίας φορτίου,** όπου ο χρήστης έχει τη δυνατότητα να επεξεργαστεί την τιμή του φορτίου ή ακόμα και να το διαγράψουν μέσω του μενού.

- **Οθόνη με την προτροπή διαγραφής φορτίου,** όπου ο χρήστης καλείται να επιβεβαιώσει τη διαγραφή ή όχι του επιλεγμένου φορτίου.
- **Οθόνη με την προτροπή αποθήκευσης τοπολογίας πριν την επιλογή νέας,** όπου ο χρήστης καλείται να αποφασίσει αν θέλει να αποθηκεύσει την τρέχουσα τοπολογία πριν την επιλογή μίας καινούργιας.

3.3.2 Κλάσεις, μέθοδοι και μεταβλητές της εφαρμογής

Οι διάφορες κλάσεις, μέθοδοι και μεταβλητές της εφαρμογής περιγράφονται αναλυτικά στο Παράρτημα Α.

3.4 Υλοποίηση

Για να υλοποιήσουμε τα χαρακτηριστικά της εφαρμογής που αναπτύξαμε προηγουμένως αξιοποιήσαμε πολλά από τα διαφορετικά εργαλεία που προσφέρει το ARKit καθώς και το RealityKit. Παρακάτω αναλύουμε τον τρόπο με τον οποίο φέραμε εις πέρας τη δημιουργία των χαρακτηριστικών αυτών, παραθέτοντας ορισμένες φορές ακόμα και τον κώδικα, αν αυτό κρίνεται απαραίτητο για την κατανόηση της μεθοδολογίας μας. Για τα περισσότερα χαρακτηριστικά που υλοποιήσαμε, επιλέξαμε να μην εμβαθύνουμε στον κώδικα τους στο κεφάλαιο αυτό αλλά στο Παράρτημα Β της εργασίας.

3.4.1 Η όψη ARView

Για τη δημιουργία μίας iOS εφαρμογής ήταν απαραίτητη αρχικά η εισαγωγή της κύριας όψης της εφαρμογής μας, τύπου *UIViewController,* την οποία ονομάσαμε *ViewController.* Η όψη αυτή είναι η κύρια δίοδος επικοινωνίας με το χρήστη και όλες οι διεργασίες της εφαρμογής μας συνδέονται με την όψη αυτή. Στη συνέχεια, για να επιτρέψουμε την επαυξημένη πραγματικότητα στην κύρια όψη δημιουργήσαμε μία όψη τύπου *ARView* μέσω του ARKit [18]. Επομένως, κάθε λειτουργικότητα της εφαρμογής μας που έχει να κάνει με κάποιο χαρακτηριστικό επαυξημένης πραγματικότητας θα πρέπει να συνδέεται με την όψη αυτή, ώστε να είναι προσβάσιμο και από το χρήστη μέσω της κύριας όψης.

3.4.1.1 Εισαγωγή του αντικειμένου ARView

Για να εισάγουμε την όψη *ARView* στην κύρια όψη της εφαρμογής μας χρησιμοποιήσαμε το αρχείο *Storyboard*, το οποίο επιτρέπει τη διαχείριση των όψεων της εφαρμογής μέσω μίας οπτικής αναπαράστασης στο Xcode [11]. Έπειτα,για να αποκτήσουμε πρόσβαση στην όψη *ARView*, προσθέσαμε την παρακάτω γραμμή κώδικα στην κύρια όψη *ViewController*. Με το πρόθεμα *@IBOutlet* συνδέεται η μεταβλητή arView με το αντικείμενο *ARView* που προστέθηκε μέσω του Storyboard.

```
@IBOutlet var arView: ARView!
```

Για να έχουμε τον έλεγχο των συνεδριών και της παραμετροποίησης του *ARView* αντικειμένου, θέσαμε ως υπεύθυνο μέλος (delegate) την τρέχουσα κλάση *ViewController* στην οποία ανήκει. Συγκεκριμένα, η ανάθεση αυτή θέλαμε να συμβαίνει κάθε φορά που εμφανίζεται η κύρια όψη *ViewController*, επομένως υλοποιήθηκε μέσα στη μέθοδο *viewDidAppear(_ animated: Bool)*.

```
override func viewDidAppear(_ animated: Bool) {
       self.arView.session.delegate = self
       // …
```

```
}
```

3.4.1.2 Αρχικοποίηση του αντικειμένου συνεδρίας ARSession

Κατά την εισαγωγή ενός αντικειμένου *ARView* στην εφαρμογή, η παραμετροποίησή του γίνεται αυτόματα. Καθώς θέλαμε να δώσουμε μία δική μας εκδοχή στο αντικείμενο συνεδρίας *ARSession*, έπρεπε πρώτα να απενεργοποιήσουμε την αυτόματη παραμετροποίηση. Έτσι, δημιουργήσαμε ένα νέο αντικείμενο παραμετροποίησης το οποίο θα ικανοποιεί τις ανάγκες της εφαρμογής μας.

```
self.arView.automaticallyConfigureSession = false
```

Το αντικείμενο παραμετροποίησης είναι μία μεταβλητή τύπου *ARWorldTrackingConfiguration* [30], καθώς η συνεδρία που θα τρέχει θα πρέπει να διαβάζει τον πραγματικό χώρο μπροστά από την κάμερα της συσκευής. Στη συνέχεια, επιλέξαμε τους τύπους επιφανειών, οριζόντιες ή και κάθετες, που θα ανιχνεύονται μέσω της ιδιότητας *planeDetection*. Στην περίπτωσή μας , θέλαμε να ανιχνεύονται και τα δύο είδη επιφανειών.

```
let config = ARWorldTrackingConfiguration()
config.planeDetection = [.horizontal, .vertical]
config.environmentTexturing = .automatic
```

Τέλος, για να ξεκινήσει το αντικείμενο συνεδρίας με την παραμετροποίηση που θέσαμε, αρκεί να του δώσουμε την εντολή μέσω της μεθόδου *run*. Επιπλέον, κατά την εντολή εκκίνησης προσθέσαμε ορισμένες επιλογές στον τρόπο με τον οποίο η συνεδρία θα λειτουργεί μέσα από τον πίνακα παραμέτρων *options*. Οι επιλογές αυτές είναι με τη σειρά η δυνατότητα επανεκκίνησης της ιχνηλάτησης του χώρου και η απομάκρυνση υπαρχόντων κόμβων πριν την εκκίνηση της συνεδρίας. Ο λεπτομερής κώδικας της παραπάνω διαδικασίας βρίσκεται στο Παράρτημα Β.

```
self.arView.session.run(config, options: [.resetTracking, .removeExistingAnchors])
```

3.4.2 Η όψη καθοδηγητή ARCoachingOverlayView

Μία εφαρμογή επαυξημένης πραγματικότητας εξαρτάται σε μεγάλο βαθμό από το χώρο που θα ανιχνεύσει μέσω της κάμερας και των αισθητήρων της συσκευής. Είναι απαραίτητο ο χρήστης με αργές κινήσεις να κουνήσει την κάμερα προς τον περιβάλλοντα χώρο ώστε να ανιχνευθεί από την εφαρμογή. Αυτό είναι μία οδηγία την οποία ο χρήστης δεν γνωρίζει απαραίτητα, επομένως πρέπει να τον καθοδηγήσουμε μέσω της εφαρμογής. Για την καθοδήγηση του χρήστη χρησιμοποιήσαμε την όψη-κλάση *ARCoachingOverlayView* [19] καθώς και τη διεπαφή του *ARCoachingOverlayDelegate* [19] στην κύρια όψη *ViewController*, δημιουργώντας μία επέκτασή της που καλείται να υλοποιήσει την παραπάνω διεπαφή.

```
extension ViewController: ARCoachingOverlayViewDelegate {
```

```
    // ...
}
```

Στο μπλοκ της επέκτασης υλοποιήσαμε μεθόδους που αφορούν τα εξής τρία στάδια:

1) Λίγο πριν ενεργοποιηθεί η όψη *ARCoachingOverlayView*. Σε αυτό το στάδιο, η εφαρμογή ξεκινά την αναζήτηση επιφανειών. Στη μέθοδο απενεργοποιούμε όλες τις μεταβλητές ανίχνευσης επαφών (Gestures Recognizers) του *ARView* ώστε να μην επηρεάσει καμιά ενέργεια του χρήστη τη λειτουργία του καθοδηγητή. Επίσης, κρύβουμε όλα τα γραφικά που ίσως να υπάρχουν στην οθόνη.

2) Αμέσως μετά αφού απενεργοποιηθεί η όψη *ARCoachingOverlayView*. Σε αυτό το στάδιο έχει πλέον ανιχνευθεί μία επιφάνεια επομένως ο καθοδηγητής ολοκλήρωσε το έργο του. Στη μέθοδο ενεργοποιούμε τον δείκτη τοποθέτησης ώστε ο χρήστης να καταλαβαίνει ποια επιφάνεια κοιτάει η κάμερά του ανά πάσα στιγμή. Επίσης, ενεργοποιούμε τον ανιχνευτή απλής επαφής (αντικείμενο *UITapGestureRecognizer* [34]) ώστε να μπορεί ο χρήστης να πατήσει στην οθόνη και να ξεκινήσει την τοποθέτηση μίας τοπολογίας, στην τρέχουσα θέση του δείκτη τοποθέτησης.

3) Όταν η όψη *ARCoachingOverlayView* απαιτήσει επανεκκίνηση της τρέχοντος συνεδρίας. Η μέθοδος *restartExperience* αναλαμβάνει την επανεκκίνηση όλης της εμπειρίας.

Εκτός από τα παραπάνω τρία στάδια, υλοποιήσαμε μεθόδους για:

- Τον καθορισμό στόχου του καθοδηγητή. Η μέθοδος ονομάζεται *setGoal* και σε αυτή ορίζουμε ως στόχο οποιοδήποτε είδος επιφάνειας. Επομένως μόλις η εφαρμογή αναγνωρίσει μία οριζόντια ή κάθετη επιφάνεια, ο στόχος επιτεύχθηκε και ο καθοδηγητής απενεργοποιείται.

- Την αρχικοποίηση του καθοδηγητή. Στη μέθοδο αυτή ακολουθούμε ορισμένα βήματα παραμετροποίησης που απαιτούνται ώστε ο καθοδηγητής να λειτουργήσει σωστά. Αρχικά, εισάγουμε την όψη του καθοδηγητή *ARCoachingOverlayView* στην τρέχουσα όψη *ARView* και ορίζουμε την οριοθέτηση και στοίχιση της όψης. Στη συνέχεια, ορίζουμε ως υπεύθυνο μέλος (delegate) για τον καθοδηγητή την κύρια όψη *ViewController* και ως συνεδρία το τρέχον αντικείμενο *ARSession* στο οποίο τρέχει η εφαρμογή μας. Ο ορισμός του delegate αποσκοπεί στο να υπάρχει διαρκής επικοινωνία μεταξύ της όψης του καθοδηγητή και της κατάστασης της τρέχουσας AR εμπειρίας. Τέλος θέτουμε το στόχο του καθοδηγητή καλώντας τη μέθοδο *setGoal* που είδαμε προηγουμένως.

3.4.3 Χειρονομίες χρήστη (Gestures)

Όπως και σε κάθε εφαρμογή, έτσι και στην AR Coulomb χρησιμοποιούνται οι συνήθεις χειρονομίες για την επικοινωνία χρήστη - συσκευής πάνω σε όψεις όπως κουμπιά (*UIButton*), συρόμενα μενού

(*UIScrollView*) και άλλες. Πέρα από τις συνηθισμένες χειρονομίες, η εφαρμογή μας απαιτεί επικοινωνία μεταξύ του χρήστη και των τρισδιάστατων μοντέλων που χρησιμοποιούνται για την αναπαράσταση και επεξεργασία μίας τοπολογίας.

Ειδικότερα στην εφαρμογή μας, ο χρήστης πρέπει να είναι σε θέση:

- να πατήσει (tap) πάνω στην επιφάνεια όπου ανιχνεύει η κάμερα ώστε να τοποθετήσει μια τοπολογία,
- να επιλέξει ένα φορτίο, το οποίο πραγματοποιείται με παρατεταμένο πάτημα ενός δακτύλου (long press) πάνω στο φορτίο,
- να μετακινήσει ένα φορτίο με πάτημα πάνω σε αυτό και σύρσιμο (drag) στη θέση που επιθυμεί ο χρήστης

3.4.3.1 Πώς ενεργοποιούμε τις χειρονομίες στο τρισδιάστατο μοντέλο του φορτίου

Όπως αναφέραμε και παραπάνω, η εφαρμογή περιέχει αρκετά διαφορετικά τρισδιάστατα μοντέλα. Το μόνο που απαιτεί διαδραστικότητα με το χρήστη είναι αυτό του φορτίου. Επομένως φροντίζουμε το μοντέλο του φορτίου να υλοποιεί τη διεπαφή "*HasCollision*" [35] ώστε να ανιχνεύεται η επαφή του με το χρήστη. Έπειτα, το ARKit διευκολύνει την εγκατάσταση χειρονομιών στα μοντέλα, καθώς χρειάστηκε μία μόνο γραμμή κώδικα για να το καταφέρουμε.

```
self.viewController?.arView.installGestures([.translation], for: point as!
HasCollision)
```

Με τη μέθοδο *installGestures* [36] καλούμε το αντικείμενο *ARView* να εγκαταστήσει τις χειρονομίες που περιέχονται στον πίνακα σε ένα συγκεκριμένο τρισδιάστατο μοντέλο. Οι χειρονομίες αυτές είναι τρεις στο σύνολο, από τις οποίες μπορούμε να επιλέξουμε όσες θέλουμε να χρησιμοποιήσουμε στα μοντέλα της εφαρμογής.

- **translation**: Αφορά την μετακίνηση του μοντέλου με το σύρσιμο του δαχτύλου.
- **scale**: Αφορά την μεγέθυνση/σμίκρυνση με ταυτόχρονη κίνηση δύο δακτύλων προς τα έξω/μέσα, πάνω στο μοντέλο.
- **rotation**: Αφορά την περιστροφή του μοντέλου με ταυτόχρονη κίνηση δύο δακτύλων με περιστροφική κίνηση, πάνω στο μοντέλο.

Όπως φαίνεται στη γραμμή κώδικα παραπάνω, χρησιμοποιήσαμε μόνο την χειρονομία **"**translation**",** καθώς δεν μας ενδιαφέρει ο χρήστης να περιστρέψει κάποιο φορτίο και σίγουρα δε θέλουμε να δίνουμε τη δυνατότητα στο χρήστη να αλλάζει το μέγεθός του.

3.4.3.2 Χειρονομία απλού πατήματος για την τοποθέτηση τοπολογίας (Tap Gesture)

Μόλις το ARKit ανιχνεύσει μία επιφάνεια, ο χρήστης πρέπει να είναι σε θέση να πατήσει στην επιφάνεια με το δάχτυλο του ώστε να μεταβεί στο μενού επιλογής τοπολογίας. Για την ανίχνευση του πατήματος δημιουργήσαμε ένα αντικείμενο ανίχνευσης *UITapGestureRecognizer* [34], και στη συνέχεια το προσθέσαμε στην τρέχουσα όψη *ARView*. Το αντικείμενο ανίχνευσης δέχεται μία μέθοδο, η οποία θα εκτελείται σε κάθε πάτημα (Tap Gesture) του χρήστη. Σε αυτή τη μέθοδο, που ονομάσαμε *handleTap*, βρίσκεται όλη η λογική που υλοποιούμε για το πάτημα του χρήστη, όπως η μετάβαση στο μενού επιλογής τοπολογίας. Περισσότερες λεπτομέρειες για τη δημιουργία του *UITapGestureRecognizer* αντικειμένου και για τη μέθοδο *handleTap* βρίσκονται στο Παράρτημα Β.

3.4.3.3 Χειρονομία παρατεταμένου πατήματος για την επιλογή φορτίου (Long Press Gesture)

Από τη στιγμή που η τοπολογία τοποθετηθεί στη σκηνή πάνω στην επιλεγμένη επιφάνεια, ο χρήστης πρέπει να είναι σε θέση να επεξεργαστεί την τιμή των φορτίων. Κάθε φορτίο αντιπροσωπεύεται στη σκηνή από ένα τρισδιάστατο σφαιρικό μοντέλο. Ο πιο απλός τρόπος για το χρήστη να επιλέξει ένα από τα πολλά φορτία της τοπολογίας είναι να πατήσει πάνω του. Αν όμως επιλέξουμε το απλό πάτημα ως τη χειρονομία επιλογής φορτίου, θα είναι αδύνατον στη συνέχεια να σύρει ο χρήστης ένα φορτίο, καθώς κάθε φορά που θα επιχειρήσει να το σύρει, το συνεχόμενο πάτημα θα ενεργοποιεί την επιλογή του ξανά και ξανά.

Για αυτό το λόγο αποφασίσαμε να χρησιμοποιήσουμε τη χειρονομία παρατεταμένου πατήματος. Ο χρήστης αρκεί να πατήσει παρατεταμένα πάνω στο μοντέλο ενός φορτίου για να το επιλέξει. Δημιουργήσαμε ένα αντικείμενο ανίχνευσης *UILongPressGestureRecognizer* [34] με όνομα "Long Press Recognizer" και του δώσαμε ως μέθοδο εκτέλεσης την *handleLongPress*. Έπειτα προσθέτουμε το αντικείμενο *UILongPressGestureRecognizer* στο αντικείμενο *ARView* και του δώσαμε ελάχιστη διάρκεια ενεργοποίησης ενός δευτερολέπτου.

```
let longPressRecognizer = UILongPressGestureRecognizer(
                          target: self,
                          action: #selector(handleLongPress(recognizer:))
                          )
self.arView.addGestureRecognizer(longPressRecognizer)
longPressRecognizer.name = "Long Press Recognizer"
longPressRecognizer.minimumPressDuration = 1
```

Στη μέθοδο *handleLongPress(recognizer:)* έχουμε δύο σημεία ενδιαφέροντος. Το πρώτο είναι ο έλεγχος σημείου πατήματος, καθώς ο χρήστης μπορεί να πατήσει παρατεταμένα οποιοδήποτε σημείο της οθόνης. Πρέπει να είμαστε βέβαιοι ότι το σημείο αντιστοιχεί στο μοντέλο ενός φορτίου. Το δεύτερο σημείο είναι οι δύο καταστάσεις του αντικειμένου *UILongPressGestureRecognizer* [34], *.began* και *.ended*. Η κατάσταση *.began* ξεκινάει όταν συμπληρωθεί το ένα δευτερόλεπτο παρατεταμένου πατήματος, ενώ η κατάσταση *.ended* όταν ο χρήστης σηκώσει τελικά το δάχτυλό του και ολοκληρώσει το παρατεταμένο πάτημα. Στην εφαρμογή αυτή μας ενδιαφέρει η κατάσταση *.began*, διότι θέλουμε να ανοίγει το μενού επεξεργασίας τιμής του φορτίου αμέσως μόλις περάσει το ένα δευτερόλεπτο παρατεταμένου πατήματος. Η πρώτη εργασία που κάνουμε στην κατάσταση *.began* είναι να βεβαιωθούμε ότι η τοποθεσία αφής αντιστοιχεί σε κάποια οντότητα φορτίου.

```
if recognizer.state == .began {
    let location = recognizer.location(in: arView)
    guard let hitEntity = arView.entity(at: location) else {return}
    if hitEntity == trackedEntity {
        /// ....
    }
}
```

Αφού βεβαιωθούμε ότι ο χρήστης πάτησε παρατεταμένα ένα φορτίο, πραγματοποιείται μετάβαση στο μενού επεξεργασίας τιμής του επιλεγμένου φορτίου.

3.4.3.4 Σύρσιμο των φορτίων για τη μετακίνησή τους (Drag)

Όπως είδαμε προηγουμένως, για την μετακίνησή των φορτίων πάνω στην επιφάνεια που βρίσκεται η τοπολογία αρκεί να εγκαταστήσουμε τη χειρονομία *"translation"* με τη μέθοδο *installGestures* [36] και ο χρήστης θα είναι σε θέση να σύρει τα φορτία. Είναι φυσικό, κατά τη μετακίνηση των φορτίων, οι τιμές και τα διανύσματα των δυνάμεων να αλλάζουν, όπως και οι τιμές και τα διανύσματα των αποστάσεων. Επομένως πρέπει κατά τη διάρκεια της μετακίνησης να υπολογίζονται διαρκώς οι τιμές τους ώστε να ενημερώνεται κατάλληλα η σκηνή, δηλαδή τα μοντέλα διανυσμάτων και επιγραφών με τη σωστή τιμή, μήκος και προσανατολισμό. Για τον διαρκή υπολογισμό τιμών κατά τη διάρκεια της μετακίνησης των φορτίων, χρησιμοποιήσαμε τις παρακάτω τρεις μεθόδους του UIKit, ο κώδικας των οποίων παραθέτεται στο Παράρτημα Β.:

- **touchesBegan(_ touches: Set<UITouch>, with event: UIEvent)**: Η επαφή του χρήστη με την οθόνη μόλις ξεκίνησε.
- **touchesMoved(_ touches: Set<UITouch>, with event: UIEvent)**: Το σημείο της υπάρχουσας επαφής του χρήστη με την οθόνη άλλαξε τοποθεσία.

- **touchesEnded(_ touches: Set<UITouch>, with event: UIEvent)**: Η υπάρχουσα επαφή σταμάτησε, ο χρήστης σήκωσε το δάχτυλό του.

Πρώτη επαφή (touchesBegan)

Είναι σημαντικό κατά την πρώτη επαφή να επιβεβαιώσουμε ότι το σημείο επαφής αντιστοιχεί στην τοποθεσία ενός φορτίου, όπως και στην περίπτωση επιλογής φορτίου, επομένως ακολουθήσαμε την ίδια διαδικασία. Είναι σημαντικό να θέσουμε την οντότητα του φορτίου ως επιβλεπόμενη, κάτι το οποίο μας βοηθάει στη συνέχεια στην ενημέρωση των διαφόρων μεγεθών του. Σημειώνεται ότι το επιλεγμένο φορτίο είναι δυνατόν να διαφέρει από το επιβλεπόμενο, καθώς ο μόνος τρόπος να επιλέξει ο χρήστης φορτίο είναι μέσω παρατεταμένου πατήματος. Στη συνέχεια, εφόσον βεβαιωθούμε ότι η τοπολογία περιέχει φορτία, προχωράμε στις παρακάτω ενέργειες:

1) Βρίσκουμε τη δύναμη στο επιλεγμένο φορτίο η οποία προέρχεται από το φορτίο που έσυρε ο χρήστης και τη θέτουμε ως επιλεγμένη. Η επιλεγμένη δύναμη θα απεικονίζεται με διαφορετικό χρώμα από τις υπόλοιπες.

2) Κρύβουμε όλα τα μοντέλα δυνάμεων. Κατά την μετακίνηση φορτίου, οι δυνάμεις αλλάζουν συνεχώς τιμές. Η επεξεργασία των τιμών τους και ειδικά η επεξεργασία των τρισδιάστατων μοντέλων είναι χρονοβόρες διεργασίες, οι οποίες μπορούν να προκαλέσουν δραστική μείωση στην απόδοση της εφαρμογής και των εικόνων που προβάλλει ανά δευτερόλεπτο (frames per second). Επομένως κρύβουμε τα μοντέλα ώστε να μη χρειαστεί να υπολογίζονται συνεχώς κατά τη μετακίνηση του φορτίου.

3) Κρύβουμε τα μοντέλα επιγραφών των τιμών των φορτίων. Οι τιμές των φορτίων μπορεί να μην αλλάζουν κατά τη μετακίνηση, όμως εφόσον κρύβουμε τα υπόλοιπα στοιχεία και επιγραφές της τοπολογίας, είναι ζήτημα αισθητικής της εφαρμογής να κρύψουμε και τις επιγραφές των φορτίων.

Μετακίνηση υπάρχουσας επαφής (touchesMoved)

Κατά τη διάρκεια της πρώτης επαφής, αν ο χρήστης εξακολουθήσει να πατάει την οθόνη στο ίδιο σημείο για ένα δευτερόλεπτο, αυτή θα θεωρηθεί ως παρατεταμένο πάτημα και θα πάμε στη περίπτωση επιλογής φορτίου. Αλλιώς, αν ο χρήστης σύρει το δάχτυλό του στην οθόνη, το φορτίο μετακινείται. Κατά τη μετακίνηση του φορτίου, ενημερώνουμε τις τιμές των δυνάμεων και των αποστάσεων, καθώς και την τιμή της επιλεγμένης δύναμης.

Τερματισμός υπάρχουσας επαφής (touchesEnded)

Κατά τον τερματισμό της επαφής, εμφανίζουμε ξανά όλες τις δυνάμεις, αποστάσεις και τις επιγραφές τους, καθώς ενημερώνουμε και τις τιμές τους. Επιπλέον, επειδή η τοπολογία δεν

βρίσκεται πάνω σε ένα ορατό σύστημα αξόνων, είναι δύσκολο για το χρήστη να ευθυγραμμίσει τα φορτία. Επομένως, στο τέλος της μετακίνησης ελέγχουμε αν το φορτίο έχει απόκλιση μικρότερη από 2 εκατοστά κατά το μήκος και πλάτος με κάποιο άλλο φορτίο. Αν ναι, τότε ο χρήστης πολύ πιθανόν να προσπάθησε να ευθυγραμμίσει τα φορτία, οπότε τον διευκολύνουμε μηδενίζοντας την απόκλιση αυτή.

3.4.3.5 Πρόβλημα με την ακύρωση χειρονομιών και αντιμετώπιση.

Το RealityKit κάνει την διαχείριση χειρονομιών στα τρισδιάστατα μοντέλα εύκολη. Όμως, όταν εγκαθιστούμε το σύνολο χειρονομιών στα μοντέλα, αυτομάτως απενεργοποιούνται οι υπόλοιπες χειρονομίες του χρήστη στην οθόνη. Αυτό εμποδίζει το χρήστη να χρησιμοποιήσει πολλές διαφορετικές λειτουργικότητες της εφαρμογής, όπως πάτημα κουμπιών. Για να αντιμετωπίσουμε αυτό το πρόβλημα χρησιμοποιήσαμε την παρακάτω γραμμή κώδικα για κάθε αντικείμενο αναγνώρισης χειρονομιών (Gesture Recognizer), ώστε να εμποδίσουμε τις χειρονομίες των μοντέλων από το να αποκλείσουν τις υπόλοιπες.

```
/// Installed gestures (EntityGesturesRecognizers for each point charge) were
/// cancelling other touches, so turn that to false
recognizer.cancelsTouchesInView = false
```

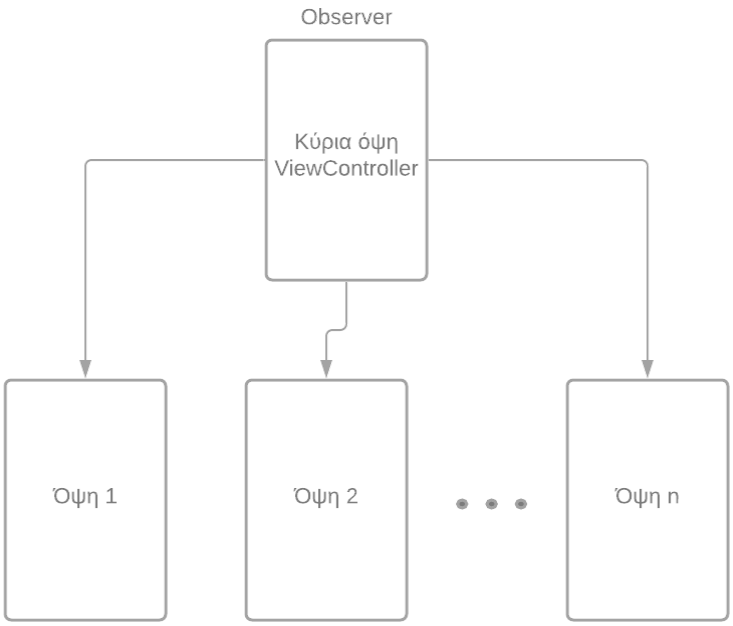
3.4.4 Σύστημα ειδοποιήσεων και παρατηρητών (Notifications & Observers)

Η εφαρμογή AR Coulomb χρησιμοποιεί διαφορετικές όψεις *UIViewController* οι οποίες περιγράφουν διάφορες όψεις, όπως την κύρια όψη της εφαρμογής και την όψη του μενού επεξεργασίας τιμής φορτίου. Οι όψεις δεν μοιράζονται μεταβλητές και μεθόδους μέλη μεταξύ τους, επομένως η επικοινωνία μεταξύ των όψεων δεν είναι απλή διαδικασία. Μία λύση είναι η χρήση καθολικών μεταβλητών στην εφαρμογή, οι οποίες είναι προσβάσιμες από όλες τις όψεις. Παρότι οι καθολικές μεταβλητές είναι μία λύση στο ζήτημα επικοινωνίας των όψεων, είναι προτιμότερο να αποφύγουμε αυτή τη μέθοδο όσο είναι δυνατόν. Ο ορισμός πολλών καθολικών μεταβλητών ενισχύει τον κίνδυνο μία όψη να επηρεάσει την τιμή μίας μεταβλητής την οποία χρησιμοποιεί μία δεύτερη όψη, έστω και προσωρινά. Αντί των καθολικών μεταβλητών, χρησιμοποιήσαμε την κλάση κέντρου ειδοποιήσεων *NotificationCenter* [27] της γλώσσας Swift, ώστε οι όψεις να επικοινωνούν μέσω ειδοποιήσεων.

3.4.4.1 Κέντρο ειδοποιήσεων (Notification Center)

Κάθε ειδοποίηση έχει έναν αποστολέα και έναν παραλήπτη. Ο παραλήπτης πρέπει να είναι συνεχώς σε ετοιμότητα για κάποια νέα ειδοποίηση, επομένως τον ονομάζουμε παρατηρητή (observer). Στην εφαρμογή μας, παραλήπτης είναι πάντα η κύρια όψη *ViewController*, ενώ οι

ειδοποιήσεις αποστέλλονται από τις άλλες όψεις, όταν χρειαστεί να ενημερώσουν την κύρια για κάποια αλλαγή στην τιμή ενός μεγέθους που θα επηρεάσει τη σκηνή.



Σχήμα 3.10: Η κύρια όψη παρακολουθεί για εισερχόμενες ειδοποιήσεις από τις υπόλοιπες όψεις.

Για κάθε ειδοποίηση δηλώνεται ένα όνομα και ένας παρατηρητής. Το όνομα μίας ειδοποίησης ορίζεται μέσω της μεθόδου *name* της κλάσης *Notification*, η οποία παίρνει ως όρισμα μία συμβολοσειρά. Ο παρατηρητής ορίζεται μέσω της μεθόδου *addObserver* της κλάσης *NotificationCenter*, η οποία μεταξύ άλλων δέχεται ως παραμέτρους την όψη η οποία θα έχει το ρόλο παρατηρητή, τη μέθοδο που θα εκτελεστεί όταν λάβει την ειδοποίηση, καθώς και το όνομα της ειδοποίησης. Οι ορισμοί των συμβολοσειρών των ονομάτων βρίσκονται στο αρχείο "ViewController.swift" και ορίζονται ως καθολικές σταθερές της εφαρμογής. Για παράδειγμα, το όνομα για την ειδοποίηση διαγραφής φορτίου ορίζεται ως εξής:

```
let removalNotificationKey = "com.leomav.coulombRemoval"
```

3.4.4.2 Τα σενάρια ειδοποιήσεων στην εφαρμογή

Στην εφαρμογή μας χρησιμοποιήσαμε ειδοποιήσεις για 7 διαφορετικά σενάρια.

- **Επιλογή νέας τοπολογίας από το μενού επιλογής**
  Αφού ο χρήστης επιλέξει μία τοπολογία, η όψη του μενού επιλογής στέλνει ειδοποίηση

στην κύρια όψη με πληροφορίες της νέας επιλεγμένης τοπολογίας. Η ειδοποίηση περιέχει ως πληροφορία την επιλεγμένη τοπολογία.

- **Επιλογή φορτίου**
  Ο χρήστης πατάει παρατεταμένα πάνω στο μοντέλο ενός φορτίου για να το επιλέξει. Μόλις το επιλέξει, εμφανίζεται το μενού επεξεργασίας του φορτίου.

- **Επεξεργασία τιμής φορτίου στο μενού επεξεργασίας**
  Ο χρήστης αλλάζει την τιμή του φορτίου μέσα από το μενού επεξεργασίας. Σε κάθε αλλαγή, η όψη του μενού επεξεργασίας στέλνει ειδοποίηση στην κύρια όψη με τη νέα τιμή. Η ειδοποίηση περιέχει ως πληροφορία τη νέα τιμή του φορτίου.

- **Διαγραφή φορτίου από το μενού επεξεργασίας**
  Αφού ο χρήστης διαγράψει το επιλεγμένο φορτίο, η όψη του μενού επεξεργασίας φορτίου στέλνει ειδοποίηση στην κύρια όψη για να την πληροφορήσει για τη διαγραφή του φορτίου, ώστε να επιλεχθεί ένα νέο φορτίο και να ενημερωθεί η τοπολογία.

- **Κλείσιμο όψης επεξεργασίας φορτίου**
  Το μενού επεξεργασίας φορτίου ειδοποιεί την κύρια όψη ότι έκλεισε.

- **Κλείσιμο όψης στιγμιότυπου τοπολογίας**
  Ο χρήστης τραβάει ένα στιγμιότυπο της τοπολογίας, σημειώνει ένα όνομα και μία περιγραφή και την αποθηκεύει. Η όψη στιγμιότυπου της τοπολογίας κλείνει και στέλνει ειδοποίηση στην κύρια όψη για την αποθήκευση της τοπολογίας. Η ειδοποίηση στέλνει ως πληροφορία το στιγμιότυπο της τοπολογίας (τύπου *pngData*).

- **Επιλογή δύναμης**
  Επιλέγεται μία από τις δυνάμεις που δέχεται το επιλεγμένο φορτίο.

3.4.4.3 Ορισμοί των παρατηρητών ειδοποιήσεων (Observers)

Οι παρατηρητές ειδοποιήσεων ορίζονται στην κλάση *ViewController* και συγκεκριμένα στην επέκτασή της που βρίσκεται στο αρχείο "ViewController_+_Observers.swift". Οι ορισμοί έχουν τη μορφή μεθόδων, οι οποίες καλούνται κατά την εκκίνηση της εφαρμογής καθώς και κάθε φορά που ο χρήστης επανεκκινεί την εμπειρία. Οι μέθοδοι αυτοί έχουν την παρακάτω μορφή και είναι η ίδια για όλους τους παρατηρητές:

```
func setupObserver() {
    let notifName = Notification.Name(rawValue: someNotificationKey)
    NotificationCenter.default.addObserver(
        self,
        selector: #selector(someMethod(notification:)),
        name: notifName,
```

```
        object: nil
    )
}
```

Στον κώδικα αυτό, η παράμετρος "*someNotificationKey*" αποτελεί την ονομασία της ειδοποίησης και η παράμετρος "*someMethod(notification:)*" αποτελεί τη μέθοδο που θα εκτελεστεί. Ο κώδικας για τις μεθόδους αυτές βρίσκεται αναλυτικά στο Παράρτημα Β.

3.4.4.4 Κλήση ειδοποιήσεων

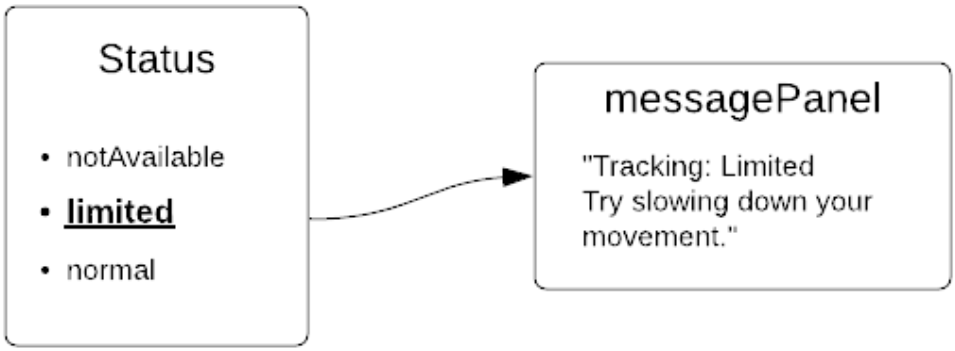
Η κλήση των ειδοποιήσεων γίνεται από οποιοδήποτε σημείο της εφαρμογής. Ο τρόπος που γίνεται είναι παρόμοιος με τον ορισμό των ειδοποιήσεων και έχει την παρακάτω μορφή:

```
let notifName = Notification.Name(rawValue: someNotificationKey)
let valueDict: [String: SomeDataType] = ["updatedValue": someNewValue!]
NotificationCenter.default.post(name: notifName, object: nil, userInfo: valueDict)
```

Η παράμετρος "*someNotificationKey*" αποτελεί και πάλι το όνομα της ειδοποίησης. Η μεταβλητή "*valueDict"* είναι τύπου λεξικού (dictionary) και περιέχει τις πληροφορίες που θέλουμε να δημοσιευτούν με την ειδοποίηση. Τέλος, αρκεί να καλέσουμε τη μέθοδο *post* του κέντρου ειδοποιήσεων *NotificationCenter* ώστε να πραγματοποιηθεί η κλήση της ειδοποίησης με όνομα "*someNotificationKey*" με τις απαραίτητες πληροφορίες προς την κύρια όψη.

3.4.5 Προβολή κατάστασης της εφαρμογής και προτεινόμενων ενεργειών (Status & Message Panel)

Η εφαρμογή AR Coulomb, ως εφαρμογή επαυξημένης πραγματικότητας, περνάει από διαφορετικά στάδια διαθεσιμότητας προς το χρήστη. Για παράδειγμα όταν ξεκινάει η εφαρμογή, οι λειτουργίες επιλογής και επεξεργασίας τοπολογίας δεν είναι διαθέσιμες στο χρήστη καθώς ακόμα δεν έχει αναγνωριστεί καμία επιφάνεια. Πρέπει επομένως ανά πάσα στιγμή να ενημερώνουμε το χρήστη για την κατάσταση της εφαρμογής και να του προτείνουμε τις απαραίτητες ενέργειες, ώστε να γνωρίζει τι πρέπει και τι μπορεί να κάνει στην εφαρμογή μας. Για το σκοπό αυτό, δημιουργήσαμε μία όψη με όνομα *messagePanel* που προβάλλει την πληροφορία της κατάστασης της εφαρμογής στην πάνω αριστερή γωνία της οθόνης. Την πληροφορία αυτή τη λαμβάνει από ένα αντικείμενο της κλάσης *Status*, την οποία επίσης δημιουργήσαμε. Η υλοποίηση της κλάσης *Status* και της όψης *messagePanel* αναλύεται στο Παράρτημα Β.



Σχήμα 3.11: Επικοινωνία μεταξύ κλάσης *Status* και της όψης μηνυμάτων *messagePanel.*

3.4.6 Τρισδιάστατα μοντέλα

Στην εφαρμογή μας χρησιμοποιήσαμε τρισδιάστατα μοντέλα για την απεικόνιση διαφόρων στοιχείων μίας τοπολογίας. Η διαδικασία δημιουργίας και επεξεργασία τους ποικίλει. Υπάρχουν διαφορετικές μέθοδοι διαχείρισης των μοντέλων μεταξύ των εργαλείων ARKit και RealityKit. Το δεύτερο, το οποίο χρησιμοποιήσαμε και στη δική μας εφαρμογή, προσφέρει ένα δενδρικό σύστημα οντοτήτων όπου κάθε οντότητα ανήκει στην κλάση *Entity*. Κάθε σκηνή, περιέχει το αντικείμενο *Entity* που αντιπροσωπεύει τη ρίζα, και στη συνέχεια κάθε επιπρόσθετο αντικείμενο μπορεί να είναι παιδί του, εγγόνι του κλπ. Τα μοντέλα που δημιουργήσαμε για τις ανάγκες της εφαρμογής μας είναι τα παρακάτω και θα τα αναλύσουμε στη συνέχεια:

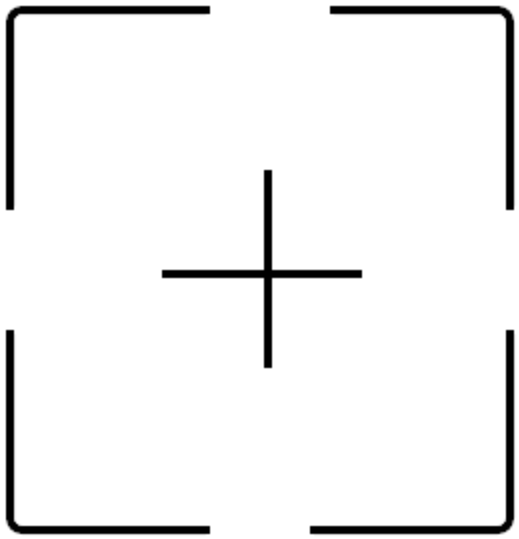
- **Δείκτης τοποθέτησης (Placement Indicator)**
- **Φορτίο**
- **Επιγραφή φορτίου**
- **Διάνυσμα δύναμης**
    - **Τριγωνική κεφαλή του διανύσματος**
    - **Κύριο σώμα διανύσματος**
- **Επιγραφή δύναμης**
- **Δείκτης απόστασης (Distance Indicator)**

- **Επιγραφή απόστασης**

Για τη διαχείριση των τρισδιάστατων μοντέλων δημιουργήσαμε μία κλάση με όνομα *EntityStore*, η οποία περιέχει μεθόδους για τη δημιουργία και ενημέρωση των μοντέλων. Για τη πρόσβαση στις μεθόδους της κλάσης *EntityStore* χρησιμοποιήσαμε μία στατική μεταβλητή της ίδιας της κλάσης.

3.4.6.1 Μοντέλο δείκτη τοποθέτησης

Ο δείκτης τοποθέτησης χρησιμεύει στο να υποδεικνύει στο χρήστη το σημείο της επιφάνειας στο οποίο είναι στραμμένη η συσκευή. Έχει έντονο κίτρινο χρώμα για να ξεχωρίζει και τετραγωνικό σχήμα με ένα σταυρό στο κέντρο ως σημάδι. Εμφανίζεται στη σκηνή μόνο όταν ο χρήστης έχει αναγνωρίσει μία επιφάνεια και την εξετάζει κουνώντας τη συσκευή, πριν τοποθετήσει οποιαδήποτε τοπολογία. Η τοπολογία που θα επιλεχθεί θα τοποθετηθεί στη θέση του δείκτη.



Σχήμα 3.12: Το μοντέλο δείκτη τοποθέτησης. Κατά τη χρήση του θα εφάπτεται με την ανιχνευμένη επιφάνεια που αντικρίζει η κάμερα του χρήστη.

Δημιουργία

Για τη δημιουργία του δείκτη τοποθέτησης δημιουργήσαμε δύο μεθόδους. Η πρώτη μέθοδος δημιουργεί μία οντότητα (Entity) και στη συνέχεια δημιουργεί ένα αντικείμενο *ModelComponent* καλώντας τη δεύτερη μέθοδο και το προσθέτει στην οντότητα.

- **load_PlacementIndicator**: ευθύνεται για τη δημιουργία της οντότητας του μοντέλου
- **load_PlacementIndicator_ModelComponent**: ευθύνεται για τη δημιουργία ενός *ModelComponent* αντικειμένου, το οποίο περιέχει τις ιδιότητες μήκους, πλάτους, βάθους, χρώματος και εικόνας του μοντέλου.

Ενημέρωση

Για την ενημέρωση και ανανέωση του δείκτη τοποθέτησης δημιουργήσαμε δύο μεθόδους.

- **update_PlacementIndicator_Transform**: αφορά την ενημέρωση της τοποθεσίας και προσανατολισμού του μοντέλου. Επομένως όταν ο χρήστης κουνάει τη συσκευή και η προβολή της κάμερας στην επιφάνεια αλλάζει θέση ή προσανατολισμό, ενημερώνεται και η τοποθεσία και ο προσανατολισμός του δείκτη μέσω αυτής της μεθόδου.
- **update_PlacementIndicator_ModelComponent**: αφορά την ανανέωση του αντικειμένου *ModelComponent* του τρισδιάτατου μοντέλου. Στη μέθοδο αυτή καλείται η μέθοδος *load_PlacementIndicator_ModelComponent* με νέα χαρακτηριστικά που θέλουμε να δώσουμε στο μοντέλο του δείκτη τοποθέτησης.

Ενεργοποίηση - Απενεργοποίηση

Για την εύκολη ενεργοποίηση και απενεργοποίηση του δείκτη τοποθέτησης, δημιουργήσαμε μία μέθοδο η οποία αναλόγως με την παράμετρο που της δίνουμε, εμφανίζει ή κρύβει τον δείκτη τοποθέτησης.

```
func toggle_PlacementIndicator(anchor: AnchorEntity, show: Bool) {
    anchor.isEnabled = show
}
```

3.4.6.2 Μοντέλο φορτίου

Για τα μοντέλα των φορτίων αντί να χρησιμοποιήσουμε την κλάση *Entity*, δημιουργήσαμε μία νέα κλάση με όνομα *PointChargeEntity*, η οποία κληρονομεί από την κλάση *Entity* και υλοποιεί τη διεπαφή *HasCollision*.

```
class PointChargeEntity: Entity, HasCollision { // Nothing to implement }
```

Ο λόγος για τον οποίο δημιουργήσαμε την παραπάνω κλάση είναι ότι η κλάση *Entity* από μόνη της δεν υλοποιεί την διεπαφή *HasCollision*, η οποία είναι απαραίτητη ώστε ο χρήστης να αλληλεπιδρά με τα τρισδιάστατα μοντέλα. Μοντέλα που χρησιμοποιούν μόνο την κλάση *Entity* δεν ανταποκρίνονται στις χειρονομίες του χρήστη. Αυτό δεν επηρεάζει τη λειτουργικότητα των περισσότερων μοντέλων της εφαρμογής μας, εκτός από αυτά των φορτίων τα οποία ο χρήστης θέλουμε να αλληλεπιδρά μαζί τους. Επομένως, δημιουργούμε τα μοντέλα των φορτίων με την κλάση *PointChargeEntity*.

Δημιουργία

Για τη δημιουργία του μοντέλου του φορτίου δημιουργήσαμε δύο μεθόδους.

- **load_PointCharge**: δημιουργεί μία οντότητα τύπου *PointChargeEntity*
- **load_PointCharge_ModelComponent**: δημιουργεί ένα *ModelComponent* αντικείμενο το οποίο θα προστεθεί στην οντότητα του μοντέλου και προσίδει ιδιότητες όπως το σχήμα, τις διαστάσεις, το χρώμα και το υλικό του μοντέλου.

Στην πρώτη μέθοδο, αφού δημιουργήσουμε την οντότητα τύπου *PointChargeEntity*, καλούμε τη δεύτερη μέθοδο και δημιουργούμε ένα αντικείμενο *ModelComponent*. Στη συνέχεια προστίθεται το αντικείμενο αυτό στην οντότητα που έχουμε ήδη δημιουργήσει. Στη δεύτερη μέθοδο δημιουργούμε ένα αντικείμενο *SimpleMaterial* με συγκεκριμένες μεταβλητές υλικού και χρώματος. Οι ιδιότητες *.metallic* και *.roughness* του *SimpleMaterial* αντικειμένου αντιστοιχούν στην μεταλλική απόδοση της επιφάνειας και πόσο λεία θα παρουσιάζεται. Η ιδιότητα *.tintColor* αφορά το χρώμα που δίνουμε στο μοντέλο του φορτίου. Στη συνέχεια, δημιουργούμε ένα αντικείμενο *ModelComponent* με σχήμα σφαίρας και με υλικό αυτό το αντικείμενο *SimpleMaterial* που δημιουργήσαμε προηγουμένως.

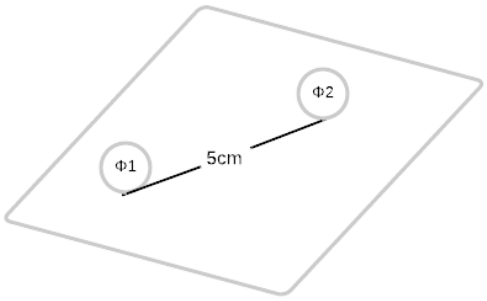
Ενημέρωση

Για την ενημέρωση του μοντέλου και των ιδιοτήτων του δημιουργήσαμε δύο μεθόδους.

- **update_PointCharge**: Δημιουργεί ένα νέο μοντέλο στη θέση του παλιού με νέες ιδιότητες καλώντας τη δεύτερη μέθοδο.
- **update_PointCharge_ModelComponent**: Ενημερώνει τις ιδιότητες του μοντέλου προσθέτοντας ένα νέο *ModelComponent* αντικείμενο στη θέση του παλιού, καλώντας τη μέθοδο *load_PointCharge_ModelComponent*.

3.4.6.3 Μοντέλο δείκτη απόστασης

Ο δείκτης απόστασης είναι ένα μοντέλο σε σχήμα ευθύγραμμου τμήματος που ενώνει δύο φορτία, με ένα κενό στη μέση για να μπει το μοντέλο επιγραφής τιμής απόστασης. Με άλλα λόγια, πρόκειται για δύο μοντέλα γραμμών, τα οποία ξεκινούν από δύο διαφορετικά μοντέλα φορτίων και καταλήγουν στο μέσο της απόστασής τους.



Σχήμα 3.13: Το μοντέλο δείκτη απόστασης μεταξύ δύο φορτίων ως δύο ευθύγραμμα τμήματα.

Δημιουργία

Για τη δημιουργία του μοντέλου δείκτη απόστασης, δημιουργήσαμε τρεις μεθόδους.

- **load_DistanceIndicator**: δημιουργεί μία οντότητα με όνομα “Distance Indicator”
- **load_DistanceLine**: δημιουργεί μία οντότητα που αντιπροσωπεύει μία από τις δύο γραμμές του δείκτη απόστασης.
- **load_DistanceLine_ModelComponent**: δημιουργεί ένα αντικείμενο *ModelComponent* με σχήμα γραμμής και συγκεκριμένες ιδιότητες χρώματος.

Ενημέρωση

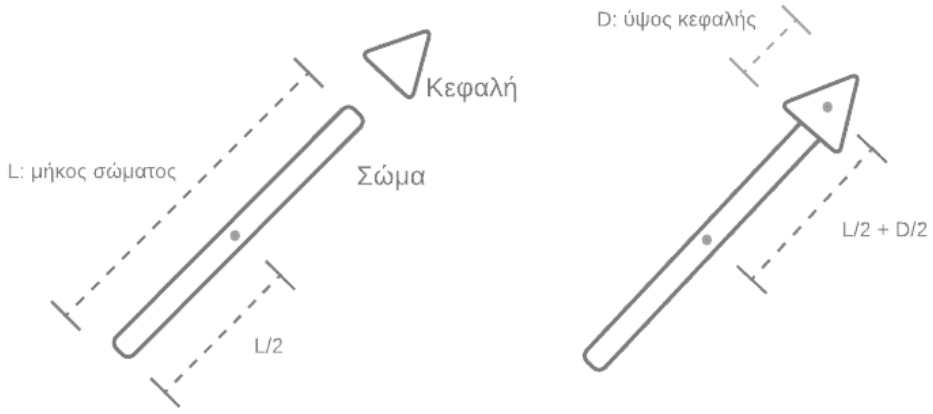
Για την ενημέρωση του μήκους των μοντέλων των γραμμών δημιουργήσαμε δύο μεθόδους.

- **update_DistanceLine_Length**: δέχεται μία οντότητα γραμμής και μία νέα τιμή μήκους, δημιουργεί ένα νέο αντικείμενο *ModelComponent* βασισμένο στη νέα τιμή, και το προσθέτει στην οντότητα στη θέση του προηγούμενου.
- **update_DistanceLines**: δέχεται δύο μοντέλα γραμμών ως παράμετρο καθώς και μία νέα τιμή συνολικού μήκους της απόστασης των δύο φορτίων. Υπολογίζει το μήκος που αντιστοιχεί στην κάθε γραμμή και καλεί την πρώτη μέθοδο για την υλοποίησή τους.

3.4.6.4 Μοντέλο διανύσματος δύναμης

Το μοντέλο διανύσματος δύναμης αποτελείται από δύο διαφορετικά μοντέλα, το μοντέλο του σώματος του βέλους του διανύσματος και το μοντέλο της τριγωνικής κεφαλής του βέλους. Το μοντέλο του σώματος έχει σχήμα ευθύγραμμου ορθογώνιου παραλληλόγραμμου και το σημείο αναφοράς του βρίσκεται στο κέντρο του, επομένως για να το τοποθετήσουμε με τρόπο ώστε να φαίνεται ότι ξεκινάει από το φορτίο, πρέπει να υπολογίζουμε το μισό του μήκους του και να το μετατοπίσουμε κατάλληλα κατά αυτή την τιμή. Σε κάθε αλλαγή τιμής ή μετακίνηση ενός φορτίου, υπολογίζουμε κατάλληλα τη θέση και τον προσανατολισμό του ευθύγραμμου τμήματος σε σχέση με το φορτίο στο οποίο ανήκει.

Κάτι αντίστοιχο συμβαίνει και με το κέντρο του μοντέλου της τριγωνικής κεφαλής. Στην περίπτωση της κεφαλής όμως υπολογίζουμε τη θέση και τον προσανατολισμό σε σχέση με το μοντέλο του σώματος του διανύσματος δύναμης κι όχι σε σχέση με το φορτίο στο οποίο ασκείται. Συγκεκριμένα, η θέση και ο προσανατολισμός της κεφαλής παραμένουν σταθερά σε σχέση με το μοντέλο του σώματος, οπότε δεν χρειάζεται κάποιος υπολογισμός για τη σωστή ενημέρωσή τους. Έτσι, κάθε φορά που ενημερώνεται το μοντέλο του σώματος, ενημερώνεται κατάλληλα και το μοντέλο της κεφαλής. Παρακάτω αναλύουμε τα μοντέλα του σώματος και της κεφαλής ξεχωριστά.



Σχήμα 3.14: Το μοντέλο διανύσματος σε σχήμα βέλους. Αποτελείται από το συνδυασμό του ευθύγραμμου μοντέλου σώματος και του τριγωνικού μοντέλου κεφαλής.

3.4.6.5 Μοντέλο σώματος του διανύσματος δύναμης

Δημιουργία

Για τη δημιουργία του μοντέλου του σώματος χρησιμοποιήσαμε δύο μεθόδους.

- **load_ArrowBody**: δημιουργεί μία οντότητα με όνομα “Arrow Body Entity” και την προσθέτει ως παιδί στο μοντέλο του φορτίου στο οποίο ασκείται η δύναμη. Με αυτό τον τρόπο το μοντέλο του διανύσματος θα ακολουθεί πάντα στο χώρο το μοντέλο του φορτίου που του αντιστοιχεί.

- **load_ArrowBody_ModelComponent**: δημιουργεί ένα αντικείμενο *ModelComponent* με το κατάλληλο σχήμα, μήκος, χρώμα και αφή.

Ενημέρωση

Για την ενημέρωση του μήκους του μοντέλου δημιουργήσαμε δύο μεθόδους.

- **update_ArrowBody**: δέχεται το μοντέλο προς ενημέρωση και τη νέα τιμή της δύναμης, υπολογίζει το νέο μήκος του διανύσματος και το μετατοπίζει κατάλληλα βάσει της νέας τιμής του μήκους του ώστε να φαίνεται στο χρήστη ότι η αρχή του είναι πάνω στο μοντέλο του φορτίου.

- **update_ArrowBody_Length**: καλεί την *load_ArrowBody_ModelComponent* για να δημιουργήσει ένα νέο αντικείμενο *ModelComponent* με τη νέα τιμή μήκους και να το προσθέσει στο μοντέλο διανύσματος στη θέση του προηγούμενου.

3.4.6.6 Μοντέλο κεφαλής του διανύσματος δύναμης

Το μοντέλο της κεφαλής του διανύσματος δύναμης δε το φτιάξαμε προγραμματιστικά όπως τα υπόλοιπα μοντέλα. Για τα υπόλοιπα μοντέλα που ήταν πιο απλά σχήματα ήταν αρκετό να δημιουργήσουμε μία οντότητα και να της προσθέσουμε ένα *ModelComponent* αντικείμενο δικής μας κατασκευής. Δυστυχώς δε δίνονται πολλές επιλογές στο σχήμα των μοντέλων που μπορεί να δημιουργήσει κανείς προγραμματιστικά, παρά μόνο τα βασικά, τα οποία είναι:

- Σφαίρα

- Ορθογώνιο παραλληλεπίπεδο

- Επιγραφή - Κείμενο

Το πλεονέκτημα της δημιουργίας ενός μοντέλου προγραμματιστικά είναι ότι ο προγραμματιστής μπορεί να παρέμβει στις ιδιότητες του κατά τη διάρκεια λειτουργίας της εφαρμογής, κι αυτό βοηθάει στη διαδραστικότητα. Για παράδειγμα, μπορεί ο χρήστης να αλλάξει το κείμενο μίας τρισδιάστατης επιγραφής, το μήκος ενός παραλληλεπιπέδου, το χρώμα μίας σφαίρας κα.

Στην περίπτωση της τριγωνικής κεφαλής χρειαζόμαστε ένα τριγωνικό μοντέλο το οποίο δε γίνεται να φτιάξουμε προγραμματιστικά. Το ARKit3 με το RealityKit δίνουν τη δυνατότητα δημιουργίας τρισδιάστατων μοντέλων σε ένα νέο εργαλείο (editor), το Reality Composer. Η επιφάνεια εργασίας

του είναι σχεδιασμένη για την εύρεση, εισαγωγή και παραμετροποίηση έτοιμων τρισδιάστατων μοντέλων. Ορισμένα από αυτά μπορεί να τα βρει ο προγραμματιστής στη βιβλιοθήκη του Reality Composer, ενώ άλλα μπορεί να τα εισάγει με την κατάλληλη μορφή αρχείου από τον υπολογιστή του.

Δημιουργία

Επιλέξαμε και παραμετροποιήσαμε το μοντέλο της τριγωνικής κεφαλής στο Reality Composer και το εξάγαμε ως αρχείο τύπου *.rcproject*. Από τη στιγμή που εξάγουμε ένα project του Reality Composer, δεν μπορούμε να μεταβάλλουμε τις τιμές των ιδιοτήτων του μοντέλου, όπως χρώμα, σχήμα κλπ. Δε μας πειράζει όμως καθώς η κεφαλή του διανύσματος παραμένει ίδια. Για να φορτώσουμε το μοντέλο που δημιουργήσαμε στο Reality Composer δημιουργήσαμε τη μέθοδο:

- **load_ArrowHead**: δέχεται ως όρισμα την οντότητα του διανύσματος δύναμης που έχουμε ήδη δημιουργήσει και το μέγεθος της δύναμης. Φορτώνουμε το μοντέλο της τριγωνικής κεφαλής, του δίνουμε όνομα "Arrow Head Entity" και το προσθέτουμε στο μοντέλο του διανύσματος δύναμης. Φέρνουμε το μοντέλο σε ακόμα μικρότερο μέγεθος χρησιμοποιώντας τη μέθοδο *scale* ώστε να ταιριάζει η κλίμακά του με αυτή του σώματος της δύναμης του διανύσματος. Τέλος, υπολογίζουμε και ορίζουμε την απόσταση του μοντέλου κεφαλής από το μοντέλο του σώματος ώστε να δημιουργεί τη ψευδαίσθηση στο χρήστη ότι είναι ένα συμπαγές μοντέλο, με την κεφαλή στην άκρη του σώματος.

Ενημέρωση

Η ενημέρωση του μοντέλου της κεφαλής αφορά την ενημέρωση της θέσης του και του προσανατολισμού του στο χώρο. Κάθε φορά που η τιμή ή η θέση ενός φορτίου αλλάζει, μεταβάλλεται και η θέση του κάθε διανύσματος δύναμης, επομένως η κεφαλή του κάθε διανύσματος πρέπει να ακολουθήσει τη νέα θέση του σώματος, ώστε να διατηρείται συνεχώς η ψευδαίσθηση ενός ενιαίου βέλους. Για την ενημέρωση του μοντέλου δημιουργήσαμε τη μέθοδο:

- **update_ArrowHead:** ενημερώνει τη θέση και τον προσανατολισμό του μοντέλου της κεφαλής σε σχέση με το μοντέλο του σώματος του διανύσματος, αφού πρώτα τα υπολογίσει χρησιμοποιώντας τη νέας τιμή της δύναμης αυτής.

3.4.6.7 Μοντέλο επιγραφής

Σε κάθε άσκηση φυσικής είναι απαραίτητο να αναγράφονται οι τιμές των διαφόρων μεγεθών. Στην εφαρμογή μας χρειαζόμαστε επιγραφές για τις τιμές των:

- δυνάμεων
- αποστάσεων
- φορτίων

Τα μοντέλα των επιγραφών ακολουθούν ίδιους κανόνες, επομένως δημιουργήσαμε καθολικές ιδιότητες και μεθόδους για όλα τα μοντέλα επιγραφών. Για παράδειγμα, όλες οι επιγραφές είναι άσπρες, έχουν ίδια γραμματοσειρά και είναι όλα προσανατολισμένα προς την κάμερα (το χρήστη).

Δημιουργία

Για τη δημιουργία ενός μοντέλου επιγραφής δημιουργήσαμε την παρακάτω μέθοδο:

- **load_TextEntity:** δημιουργεί ένα μοντέλο επιγραφής, με τους κατάλληλους χαρακτήρες που δέχεται ως παράμετρο, και το τοποθετεί σε μία σχετική θέση με την οντότητα την οποία περιγράφει. Για παράδειγμα η επιγραφή ενός φορτίου θα τοποθετηθεί σε θέση σχετική με τη θέση του φορτίου αυτού.

Ενημέρωση

Η ενημέρωση της επιγραφής αφορά την ενημέρωση των χαρακτήρων που την αποτελούν. Για παράδειγμα, όταν αλλάζουμε την τιμή ενός φορτίου πρέπει και η επιγραφή να αλλάξει από “5 Cb” σε “6 Cb”. Για την ενημέρωση της επιγραφής δημιουργήσαμε τη μέθοδο:

- **update_TextEntity:** δέχεται τη νέα τιμή της επιγραφής ως όρισμα, δημιουργεί ένα νέο αντικείμενο *ModelComponent* χρησιμοποιώντας την τιμή αυτή και το προσθέτει στο μοντέλο της επιγραφής.

Κατά τη διάρκεια λειτουργίας της εφαρμογής θέλουμε όλες οι επιγραφές να είναι προσανατολισμένες προς το χρήστη. Για αυτό δημιουργήσαμε μία μέθοδο η οποία αφορά την ενημέρωση όλων των επιγραφών:

- **update_AllTextOrientation**: ενημερώνει τον προσανατολισμό όλων των μοντέλων επιγραφών στη σκηνή ώστε να “κοιτάνε” το χρήστη. Αυτό το καταφέρνουμε με τη μέθοδο *look(at:)* και θέτοντας ως στόχο τη φυσική τοποθεσία της κάμερας της συσκευής.

3.4.7 Τοπολογία

Στην εφαρμογή μας, ένα αντικείμενο τοπολογίας αποτελείται από ένα αντικείμενο *AnchorEntity* για την τοποθέτησή της σε μία πραγματική επιφάνεια, και από τα φορτία, τις δυνάμεις και τους δείκτες απόστασής τους. Για την αναπαράστασή μίας τοπολογίας δημιουργήσαμε την κλάση *Topology*. Κάθε αντικείμενο της κλάσης αυτής περιέχει:

- Το αντικείμενο της κλάσης *UIViewController* στο οποίο τρέχει η εφαρμογή μας
- Το αντικείμενο *ARAnchorEntity* για την τοποθέτησή της τοπολογίας στο χώρο
- Τα φορτία
- Τις θέσεις των φορτίων της τοπολογίας
- Τις δυνάμεις των φορτίων
- Τους δείκτες αποστάσεων των φορτίων
- Τις επιγραφές των διαφόρων μεγεθών (φορτίο, απόσταση, δύναμη)

3.4.7.1 Η διαχείριση της κλάσης Topology

Κατά την αρχικοποίηση της κύριας όψης *UIViewController*, δημιουργούμε ένα αντικείμενο της κλάσης *Topology* το οποίο ονομάζουμε *topology*.

```
var topology: Topology = Topology()
```

Αυτό το αντικείμενο θα αντιπροσωπεύει πάντα την τρέχουσα τοπολογία που βλέπει ο χρήστης. Το αντικείμενο αυτό χρησιμεύει στο να έχει ο προγραμματιστής εύκολη πρόσβαση στην τρέχουσα τοπολογία, ώστε να την ενημερώσει ή και να τη διαγράψει. Για να τοποθετήσει ο χρήστης μία τοπολογία σε μία αναγνωρισμένη επιφάνεια στο χώρο, πρέπει απλώς να πατήσει πάνω στην οθόνη. Με το πάτημα, καλείται η μέθοδος *pinToScene* και τοποθετείται ένα αντικείμενο *ARAnchorEntity* στην επιφάνεια, στο σημείο όπου βρίσκεται ο δείκτης τοποθέτησης. Το αντικείμενο της τοπολογίας παραμένει ακόμα χωρίς στοιχεία, όπως φορτία και δυνάμεις, μέχρι να επιλεχθεί μία τοπολογία από το μενού επιλογής. Έτσι, μόλις εκτελεστεί η μέθοδος *pinToScene*, γίνεται μετάβαση στο μενού επιλογής τοπολογίας.

Μόλις ο χρήστης επιλέξει μία, καλείται η μέθοδος *placeTopology*, η οποία διαγράφει την τρέχουσα τοπολογία, αν υπήρχε, και δημιουργεί μία νέα με τα στοιχεία της επιλεγμένης. Η διαδικασία αυτή περιλαμβάνει την δημιουργία φορτίων, δυνάμεων, δεικτών αποστάσεων των φορτίων και στη συνέχεια την αποθήκευσή τους. Για τη διαγραφή της τρέχουσας τοπολογίας χρησιμοποιείται η μέθοδος *clearTopology*, η οποία είναι υπεύθυνη για τη διαγραφή όλων των στοιχείων της

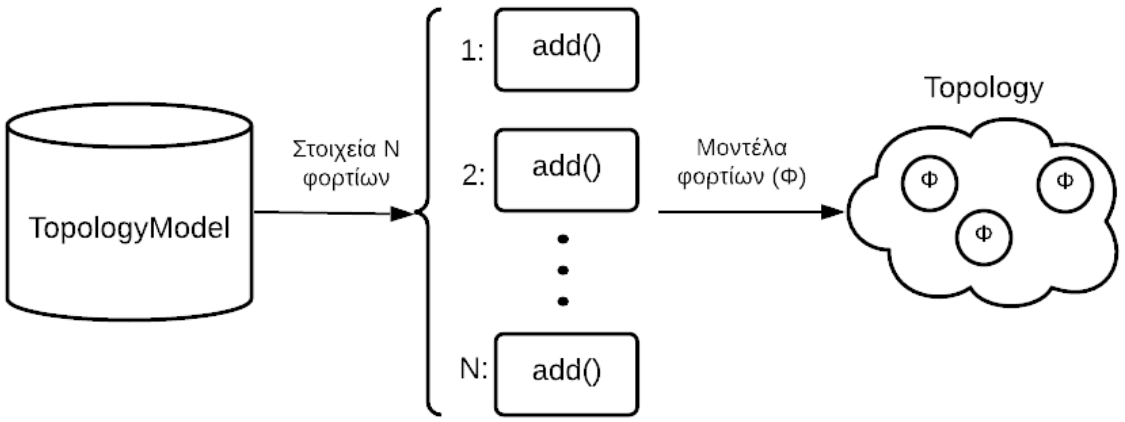
τοπολογίας, όπως τα φορτία και οι δυνάμεις τους. Οι κώδικας για τις μεθόδους *pintToScene*, *placeTopology*, *clearTopology* βρίσκονται στο Παράρτημα Β.

3.4.7.2 Η διαχείριση των φορτίων της τοπολογίας

Για την καλύτερη διαχείριση των φορτίων της τοπολογίας, δημιουργήσαμε μία επέκταση της κλάσης *Topology***,** στην οποία αναπτύξαμε μεθόδους για το χειρισμό τους. Σε αυτήν καλύπτονται τα σενάρια προσθήκης ενός φορτίου στην τοπολογία, αφαίρεσής του, καθώς και η δυνατότητα απόκρυψης και επανεμφάνισής του.

Μεθοδολογίες προσθήκης φορτίου

Υπάρχουν δύο τρόποι προσθήκης ενός φορτίου στην τοπολογία. Ο πρώτος αφορά την προσθήκη των προκαθορισμένων φορτίων της τοπολογίας αυτής. Προκαθορισμένα είναι τα φορτία που περιέχονται ήδη στις αποθηκευμένες τοπολογίες στη μνήμη της εφαρμογής. Οι αποθηκευμένες τοπολογίες αντιπροσωπεύονται από την κλάση *TopologyModel*, η οποία δεν περιέχει τα μοντέλα της τοπολογίας αλλά όλα τα υπόλοιπα στοιχεία που την περιγράφουν, όπως τα φορτία της, τις τιμές, τις θέσεις των φορτίων και άλλα. Επομένως, δημιουργήσαμε μία μέθοδο όπου κατά την επιλογή ενός αντικειμένου *TopologyModel* από το μενού επιλογής, διαβάζει τα δεδομένα για κάθε φορτίο, δηλαδή την τιμή και τις συντεταγμένες του, δημιουργεί ένα μοντέλο φορτίου βάσει αυτών και τα προσθέτει στο αντικείμενο τοπολογίας της σκηνής. Κάθε φορά που προστίθεται ένα φορτίο στην τοπολογία, το σημειώνουμε ως “επιλεγμένο” φορτίο. Επομένως το φορτίο που προστίθεται τελευταίο είναι πάντα και το επιλεγμένο.



Σχήμα 3.15: Η δημιουργία φορτίων στην τοπολογία από τα αποθηκευμένα δεδομένα της εφαρμογής.

Ο δεύτερος τρόπος αφορά την προσθήκη ενός φορτίου κατά τη διάρκεια αναπαράστασης μίας τοπολογίας, δηλαδή όταν η τοπολογία και τα προκαθορισμένα φορτία της έχουν ήδη προστεθεί στη σκηνή. Ο χρήστης μπορεί να προσθέσει ένα φορτίο σε μία τέτοια σκηνή με το πάτημα του ανάλογου κουμπιού από το πλαϊνό μενού. Με το πάτημά του, δημιουργείται ένα νέο φορτίο και προστίθεται στο υπάρχων αντικείμενο τοπολογίας στη σκηνή, σε μία τυχαία θέση. Η θέση αυτή είναι εντός μίας προκαθορισμένη ακτίνας μεταξύ των υπόλοιπων φορτίων. Μόλις τοποθετηθεί, γίνεται αυτόματα μετάβαση στο μενού επεξεργασίας τιμής φορτίου ώστε ο χρήστης να επιλέξει την τιμή του νέου φορτίου. Στην περίπτωση που πριν την προσθήκη υπάρχει στη σκηνή μόνο ένα φορτίο, επιλέγεται μία θέση σε ακτίνα 10 εκατοστών από αυτό. Αν υπάρχουν δύο ή περισσότερα φορτία, επιλέγεται μία θέση μεταξύ των φορτίων αυτών. Η επιλογή τυχαίας θέσης πραγματοποιείται με τη μέθοδο *randomPosition*.

Διαγραφή φορτίου

Για την αφαίρεση ενός φορτίου δημιουργήσαμε τη μέθοδο *removePointCharge*, στην οποία ανατρέχουμε τα φορτία της τοπολογίας, βρίσκουμε το επιλεγμένο και το αφαιρούμε. Είναι σημαντικό μόλις αφαιρέσουμε το φορτίο να ανανεώσουμε τις δυνάμεις, τις αποστάσεις και τα αντίστοιχα μοντέλα τους, καθώς και να επιλέξουμε ένα νέο φορτίο ως “επιλεγμένο”. Για την αφαίρεση όλων των φορτίων από την τοπολογία δημιουργήσαμε τη μέθοδο *removeAllPointCharges*, στην οποία καλούμε τη μέθοδο *removePointCharge* για κάθε φορτίο στην τοπολογία.

Απόκρυψη και επανεμφάνιση των μοντέλων των φορτίων

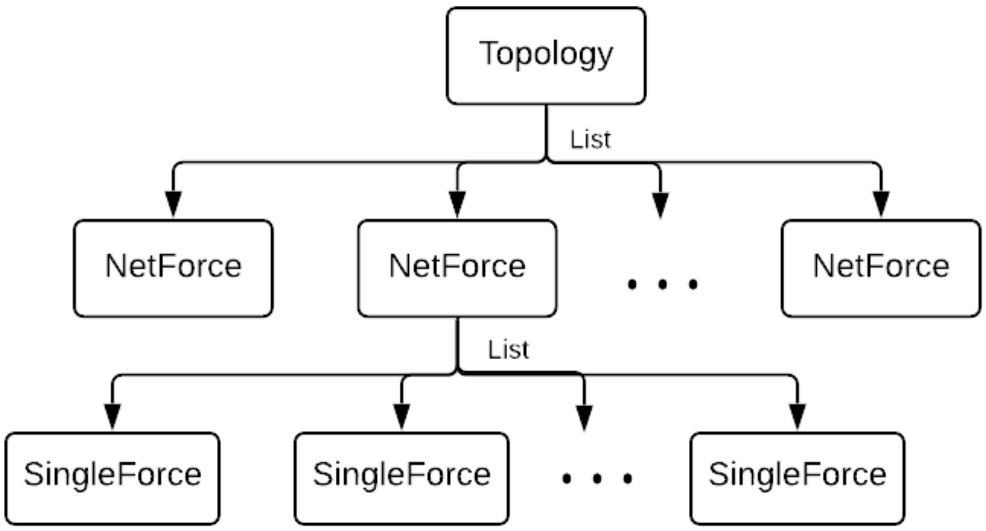
Η εμφάνιση ή απόκρυψη των μοντέλων των φορτίων είναι πολύ απλή διαδικασία, αρκεί για το φορτίο που επιθυμούμε να θέσουμε την ιδιότητα *isEnabled* στο μοντέλο του με την κατάλληλη τιμή (true/false αντίστοιχα). Για τα μοντέλα των φορτίων δημιουργήσαμε τη μέθοδο *togglePointCharges(show: Bool)*, η οποία ανάλογα με την αληθή ή ψευδή τιμή της παραμέτρου *show* εμφανίζει ή αποκρύπτει όλα τα φορτία. Για τα μοντέλα των επιγραφών των φορτίων δημιουργήσαμε αντίστοιχα τη μέθοδο *toggleCoulombsLabels(show: Bool)*, η οποία λειτουργεί ομοίως.

3.4.7.3 Η διαχείριση των δυνάμεων της τοπολογίας

Για τις δυνάμεις που δέχονται τα φορτία δημιουργήσαμε μία δεύτερη επέκταση της κλάσης *Topology*, στην οποία βρίσκονται μέθοδοι για τη διαχείριση των αντικειμένων δύναμης που υπάρχουν στην τοπολογία. Ο χειρισμός των δυνάμεων διαφέρει από των φορτίων, καθώς σε περίπτωση που χρειαστεί να προσθέσουμε μία δύναμη, δεν προσθέτουμε μόνο αυτή, αλλά διαγράφονται και δημιουργούνται όλες οι δυνάμεις εξαρχής, συμπεριλαμβανομένου αυτής. Επιλέξαμε αυτή την τεχνική καθώς οι συνισταμένες δυνάμεις εξαρτώνται από τις συνιστώσες τους, και είναι πιο εύκολο να αποφευχθούν σφάλματα κατά τον υπολογισμό τους.

Προσθήκη και αφαίρεση δυνάμεων

Η προσθήκη δυνάμεων στο αντικείμενο τοπολογίας απαιτεί τη δημιουργία ενός αντικειμένου δύναμης μεταξύ κάθε ζεύγους φορτίων, το μοντέλο του οποίου θα έχει την κατάλληλη διεύθυνση, καθώς και τιμή στην επιγραφή του. Δημιουργήσαμε τη μέθοδο *addAllForces*, η οποία προσπελάζει όλα τα φορτία της τοπολογίας και δημιουργεί μία δύναμη για κάθε ζεύγος. Ορισμένες από τις δυνάμεις που δημιουργούνται είναι και οι συνισταμένες (*NetForce*) των φορτίων, οι οποίες περιέχουν μία λίστα με τις συνιστώσες δυνάμεις τους (*SingleForce*). Το αντικείμενο τοπολογίας διατηρεί επίσης μία λίστα με όλες τις συνισταμένες δυνάμεις που εμπεριέχονται σε αυτή.



Σχήμα 3.16: Ένα αντικείμενο τοπολογίας περιέχει μία λίστα από συνισταμένες δυνάμεις, οι οποίες με τη σειρά τους περιέχουν από μία λίστα με τις συνιστώσες τους η κάθε μία.

Για τη διαγραφή των δυνάμεων δημιουργήσαμε τη μέθοδο *clearAllForces*, στην οποία προσπελάζονται όλες οι συνισταμένες δυνάμεις της τοπολογίας και για κάθε μιά διαγράφουμε όλα τα μοντέλα των συνιστωσών τους και στη συνέχεια το μοντέλο της ίδιας της συνισταμένης. Με αυτόν τον τρόπο επιβεβαιώνουμε ότι δεν θα εξακολουθούν να υπάρχουν οντότητες των μοντέλων στην εφαρμογή. Τέλος, αφού διαγραφούν τα μοντέλα, προχωράμε στη διαγραφή των συνισταμένων δυνάμεων από τη λίστα της τοπολογίας.

Ενημέρωση δυνάμεων

Ένα αντικείμενο δύναμης περιέχει αρκετές ιδιότητες που την περιγράφουν. Αυτές είναι η τιμή του, το μοντέλο του διανύσματός της και το μοντέλο της επιγραφής της τιμής της. Όταν μία δύναμη αλλάζει τιμή ή/και φορά, πρέπει να βεβαιωθούμε πως ενημερώνουμε τόσο την τιμή και το μοντέλο της επιγραφής της, όσο και το μήκος και τη φορά του μοντέλου του διανύσματός της. Με τη μέθοδο

updateForces, ενημερώνεται κάθε συνισταμένη δύναμη της τοπολογίας, καθώς και οι συνιστώσες δυνάμεις τους, δίνοντας έμφαση στην ενημέρωση όλων των ιδιοτήτων που είδαμε προηγουμένως.

Εμφάνιση και απόκρυψη δυνάμεων

Η εμφάνιση και η απόκρυψη των δυνάμεων πραγματοποιείται εύκολα με την ανάθεση της ανάλογης τιμής στην ιδιότητα *isEnabled* των μοντέλων των δυνάμεων. Η μέθοδος *toggleAllForces(show: Bool)* προσπελάζει όλες τις συνισταμένες δυνάμεις της τοπολογίας καθώς και τις συνιστώσες τους και θέτει την ιδιότητα *isEnabled* ως αληθή ή ψευδή, ανάλογα με την παράμετρο *show*.

Επανυπολογισμός των δυνάμεων

Κατά την αναπαράσταση της τοπολογίας, υπάρχουν σενάρια όπου επιθυμούμε τον επανυπολογισμό των δυνάμεων και των ιδιοτήτων τους. Τέτοια σενάρια είναι η προσθήκη ή αφαίρεση ενός φορτίου, η επιλογή μίας νέας τοπολογίας και γενικότερα οποιοδήποτε σενάριο εμπεριέχει την αλλαγή ή ενημέρωση των φορτίων της τοπολογίας. Για τη διαδικασία αυτή δημιουργήσαμε τη μέθοδο *reloadAllForces*, στην οποία αρχικά διαγράφουμε όλες τις υπάρχουσες δυνάμεις της τοπολογίας καλώντας τη μέθοδο *clearAllForces*. Στη συνέχεια δημιουργούμε ξανά όλες τις απαιτούμενες δυνάμεις και τις προσθέτουμε στην τοπολογία με τη μέθοδο *addAllForces* και τέλος καλούμε τη μέθοδο *updateForces* για να βεβαιωθούμε πως οι δυνάμεις και οι ιδιότητές τους είναι ενημερωμένες.

3.4.7.4 Η διαχείριση των δεικτών αποστάσεων

Για την αναπαράσταση των αποστάσεων και των μοντέλων τους ακολουθήσαμε την ίδια μεθοδολογία με αυτή των δυνάμεων. Δημιουργήσαμε μία επέκταση της κλάσης *Topology*, η οποία περιέχει μεθόδους για τη δημιουργία, διαγραφή, ενημέρωση και επανυπολογισμό όλων των δεικτών αποστάσεων, ακολουθώντας την ίδια λογική με τις μεθόδους διαχείρισης των δυνάμεων. Όπως είδαμε προηγουμένως, κάθε δείκτης απόστασης αποτελείται από δύο ευθύγραμμα τμήματα, με μία επιγραφή ανάμεσά τους.

Προσθήκη και αφαίρεση δεικτών αποστάσεων

Με τη μέθοδο *addDistanceIndicators* προσπελάζονται όλα τα φορτία και για κάθε ζεύγος δημιουργούμε ένα αντικείμενο δείκτη απόσταση. Κάθε φορτίο διαθέτει μία λίστα από δείκτες αποστάσεων με τα γειτονικά τους φορτία, επομένως μετά τη δημιουργία κάθε δείκτη τον προσθέτουμε στις αντίστοιχες λίστες του ζεύγους. Επίσης, προσθέτουμε το δείκτη στη λίστα με όλους τους δείκτες απόστασης που διαθέτει το αντικείμενο της τοπολογίας. Για την διαγραφή όλων των δεικτών αποστάσεων δημιουργήσαμε τη μέθοδο *clearDistanceIndicators*, στην οποία για κάθε φορτίο της τοπολογίας προσπελάζεται η λίστα του με τους δείκτες απόστασής του με τα γειτονικά φορτία και τους διαγράφουμε αφαιρώντας πρώτα την οντότητα του μοντέλου και στη συνέχεια το ίδιο το αντικείμενο του δείκτη από τη λίστα του φορτίου.

Ενημέρωση δεικτών απόσταστης

Ομοίως με την ενημέρωση των δυνάμεων, με τη μέθοδο *updateDistanceIndicators* ενημερώνουμε την τιμή της απόστασης, το μοντέλο του δείκτη καθώς και το μοντέλο της επιγραφής της τιμής του δείκτη. Σημειώνεται πως το μοντέλο του δείκτη αποτελείται από δύο ξεχωριστά μοντέλα με σχήμα ευθύγραμμου τμήματος, τοποθετημένα με τέτοιο τρόπο ώστε να υπάρχει κενό στη μέση της απόστασης για να χωρέσει το μοντέλο επιγραφής της τιμής της απόστασης.

Εμφάνιση και απόκρυψη δεικτών απόστασης

Όπως και στην περίπτωση των δυνάμεων, δημιουργήσαμε τη μέθοδο *toggleDistanceIndicators(show: Bool)* στην οποία εμφανίζονται ή αποκρύπτονται τα μοντέλα των δεικτών απόστασης θέτοντας αληθή ή ψευδή την ιδιότητα *isEnabled* των μοντέλων τους.

Επανυπολογισμός των δεικτών απόστασης

Σε περίπτωση που απαιτείται ο επανυπολογισμός των δεικτών απόστασης της τοπολογίας, επειδή για παράδειγμα προστέθηκε ένα φορτίο στην τοπολογία, ακολουθούμε τα ίδια βήματα με τον επανυπολογισμό δυνάμεων, δημιουργώντας τη μέθοδο *reloadDistanceIndicators*. Αρχικά αφαιρούμε όλους τους τρέχοντες δείκτες από τη σκηνή και την τοπολογία καλώντας τη μέθοδο *clearDistanceIndicators* και στη συνέχεια καλούμε τη μέθοδο *addDistanceIndicators* για την εκ νέου δημιουργία τους.

3.4.7.5 Αποθήκευση τοπολογίας

Μέσα από την εφαρμογή AR Coulomb δίνεται η δυνατότητα στον χρήστη να αποθηκεύει τις τοπολογίες που δημιουργεί στη μνήμη της εφαρμογής, ώστε να είναι διαθέσιμες για μελλοντική χρήση. Επίσης, είναι εφικτό να φορτώνει τις αποθηκευμένες τοπολογίες καθώς και να τις διαγράφει. Για το σκοπό αυτό δημιουργήσαμε την κλάση *TopologyStore*, η οποία περιέχει μεθόδους με τις οποίες επικοινωνούμε τις τοπολογίες με το χώρο αποθήκευσης της εφαρμογής. Η κλάση περιέχει μία στατική σταθερά που ονομάζεται *sharedInstance* και ο τύπος της είναι της ίδιας της κλάσης, δηλαδή *TopologyStore*.

```
class TopologyStore {
    static let sharedInstance = TopologyStore()
    // ...
}
```

Η στατική σταθερά χρησιμοποιείται για να έχουμε πρόσβαση στις μεθόδους από όλα τα σημεία της εφαρμογής. Επίσης, η κλάση περιέχει μία λίστα με όλες τις αποθηκευμένες τοπολογίες ώστε

να είναι εύκολα προσβάσιμες από το μενού επιλογής τοπολογίας. Για την αποθήκευση μόνιμων ή προσωρινών δεδομένων της εφαρμογής στη συσκευή χρησιμοποιήσαμε το Core Data framework.

3.4.8 Το περιεχόμενο μίας τοπολογίας

Για να γίνει αντιληπτή η αναπαράσταση του νόμου Coulomb, κάθε αντικείμενο τοπολογίας στην εφαρμογή μας περιέχει τα φορτία που μελετά ο χρήστης, τις δυνάμεις τους, τους δείκτες απόστασης μεταξύ των φορτίων καθώς και επιγραφές για το κάθε μέγεθος. Για καθένα από τα παραπάνω δημιουργήσαμε μία κλάση. Οι παρακάτω κλάσεις κι ο τρόπος που αλληλεπιδρούν αναλύονται σε βάθος στο Παράρτημα Β.

- **PointChargeClass**: το φορτίο
- **Force**: μία γενικότερη κλάση για την αναπαράσταση όλων των δυνάμεων
- **SingleForce**: η συνιστώσα δύναμη μεταξύ δύο φορτίων, η οποία κληρονομεί από την κλάση *Force*
- **NetForce**: η συνισταμένη δύναμη που δέχεται ένα φορτίο, η οποία κληρονομεί από την κλάση *Force*
- **DistanceIndicator**: ο δείκτης απόστασης μεταξύ δύο φορτίων

3.4.8.1 Η κλάση PointChargeClass

Η κλάση *PointChargeClass* αντιπροσωπεύει τα φορτία της τοπολογίας και περιέχει ιδιότητες για την περιγραφή της τιμής του (με μονάδα μέτρησης το 1 Coulomb), τη συνισταμένη δύναμη που δέχεται, τις δυνάμεις που ασκεί σε άλλα φορτία καθώς και την ίδια την οντότητα μοντέλου που αναπαριστά το φορτίο. Οι ιδιότητες και μεθόδοι της κλάσης αναλύονται σε βάθος στο Παράρτημα Β.

3.4.8.2 Η κλάση Force

Η κλάση Force περιέχει ιδιότητες και μεθόδους που περιγράφουν και τα δύο είδη δυνάμεων, δηλαδή τις συνιστώσες και τις συνισταμένες δυνάμεις που δέχεται ένα φορτίο, τόσο ως έννοιες όσο και ως τρισδιάστατα μοντέλα. Οι κλάσεις και των δύο αυτών ειδών κληρονομούν από την κλάση Force. Ορισμένες από τις πιο βασικές ιδιότητες είναι το μέτρο και γωνία της δύναμης, η οντότητα βέλους που την αντιπροσωπεύει και η οντότητα επιγραφής που τη συνοδεύει. Αναλυτικότερα οι ιδιότητες βρίσκονται στο Παράρτημα Β, όπως και μία περιγραφή για την αλληλεπίδρασή τους.

3.4.8.3 SingleForce

Η κλάση *SingleForce* αντιπροσωπεύει τη δύναμη που ασκείται μεταξύ δύο φορτίων, ανεξάρτητα από την ύπαρξη άλλων φορτίων και δυνάμεων. Το μέτρο της υπολογίζεται απευθείας από το νόμο του Coulomb. Κληρονομεί τις ιδιότητες και μεθόδους από την κλάση *Force* και διαθέτει επιπλέον δύο ιδιότητες τύπου *PointChargeClass*, οι οποίες αντιπροσωπεύουν το φορτίο από το οποίο πηγάζει η δύναμη και το φορτίο το οποίο τη δέχεται.

3.4.8.4 NetForce

Η κλάση NetForce αντιπροσωπεύει τη συνισταμένη δύναμη που δέχεται ένα φορτίο, αν συνδυάσουμε όλα τα αντικείμενα *SingleForce* των δυνάμεων που ασκούνται σε αυτό. Οι ιδιότητες και οι μέθοδοι της κλάσης βρίσκονται στο Παράρτημα Β.

3.4.8.5 DistanceIndicator

Η κλάση *DistanceIndicator* αντιπροσωπεύει το δείκτη απόστασης μεταξύ δύο φορτίων. Αποτελείται από ένα τρισδιάστατο μοντέλο του δείκτη και ένα τρισδιάστατο μοντέλο επιγραφής της απόστασης. Το μοντέλο επιγραφής βρίσκεται στο μέσο του μοντέλου δείκτη. Η κλάση περιέχει ιδιότητες που αφορούν τα δύο φορτία καθώς και την ίδια την οντότητα του δείκτη. Η οντότητα αυτή αποτελείται από δύο ευθύγραμμα μοντέλα με ένα μοντέλο επιγραφής στο μέσο τους, τα οποία βρίσκονται ανάμεσα στα δύο φορτία. Οι ιδιότητες καταγράφονται αναλυτικά στο Παράρτημα Β, όπως και οι μέθοδοι για τη δημιουργία και ενημέρωσή του μοντέλου.

3.4.9 Τα δεδομένα της εφαρμογής

Στην εφαρμογή AR Coulomb επιλέξαμε τη συσκευή ως χώρο αποθήκευσης των δεδομένων της εφαρμογής, αντί κάποιου απομακρυσμένου εξυπηρετητή. Συγκεκριμένα, τα δεδομένα που αποθηκεύονται στην εφαρμογή μας είναι οι τοπολογίες καθώς και τα φορτία που αυτές περιέχουν, όπως επίσης και οι ιδιότητές τους (θέση, τιμή κλπ). Για την αποθήκευση μόνιμων ή προσωρινών δεδομένων στη συσκευή χρησιμοποιήσαμε το εργαλείο (framework) Core Data.

Το Core Data παρέχει στο XCode ένα εργαλείο για την αναπαράσταση μοντέλων των δεδομένων της εφαρμογής καθώς και της σχέσης που έχουν μεταξύ τους μέσα από ειδικά κατασκευασμένες κλάσεις. Αυτό μας δίνει τη δυνατότητα να αποθηκεύουμε δεδομένα στη συσκευή χωρίς να απαιτείται η απευθείας πρόσβαση σε κάποια βάση δεδομένων και κάνει το χειρισμό των δεδομένων πολύ πιο εύκολο. Στην περίπτωσή μας, δημιουργήσαμε δύο μοντέλα, ένα μοντέλο για την αποθήκευση αντικειμένων τοπολογίας και ένα μοντέλο για αντικείμενα φορτίων, τα οποία αναλύονται σε βάθος στο Παράρτημα Β. Παρακάτω αναλύουμε τη διαχείριση των τοπολογιών ως δεδομένα της εφαρμογής.

3.4.9.1 Ο χώρος διατήρησης πληροφορίας (NSPersistentContainer)

Εκτός των μοντέλων του Core Data όμως έπρεπε να δημιουργήσουμε και έναν εικονικό χώρο στον οποίο θα αποθηκεύεται η πληροφορία. Στην κλάση *PersistenceService* δημιουργήσαμε μία στατική μεταβλητή τύπου *NSPersistenceContainer* , η οποία είναι υπεύθυνη για τη διατήρηση, ενημέρωση και διάθεση της πληροφορίας της εφαρμογής [37].

```
static var persistentContainer: NSPersistentContainer = {
    let container = NSPersistentContainer(name: "Topologies")
    container.loadPersistentStores(completionHandler: { (storeDescription, error) in
        if let error = error as NSError? {
            fatalError("Unresolved error \(error), \(error.userInfo)")
        }
    })

    return container
}()
```

Για να αποθηκεύονται οι αλλαγές που πραγματοποιούμε στα δεδομένα της εφαρμογής δημιουργήσαμε μία στατική μέθοδο που ελέγχει για τυχόν αλλαγές και τις αποθηκεύει στον εικονικό χώρο που ορίζεται από την μεταβλητή *persistentContainer* και είναι διαθέσιμος μέσω της ιδιότητας *viewContext*, η οποία είναι τύπου *NSManagedObjectContext*. Κάθε φορά λοιπόν που επιθυμούμε να αποθηκεύσουμε αλλαγές στο Core Data μοντέλο της εφαρμογής αρκεί να καλέσουμε τη μέθοδο *saveContext*.

```
static func saveContext() {
    let context = persistentContainer.viewContext
    if context.hasChanges {
        do {
            try context.save()
        } catch {
            let nserror = error as NSError
            fatalError("Unresolved error \(nserror), \(nserror.userInfo)")
        }
    }
}
```

3.4.9.2 Η κλάση TopologyStore

Είναι απαραίτητο να υπάρχει ένας δίαυλος επικοινωνίας της εφαρμογής με το Core Data και το αντικείμενο *NSPersistentContainer* που είδαμε προηγουμένως, για αυτό και δημιουργήσαμε την κλάση *TopologyStore*, η οποία περιέχει μεθόδους αποθήκευσης, διαγραφής και φόρτωσης τοπολογιών μέσω της κλάσης *PersistenceService***.** Η κλάση *TopologyStore* φροντίσαμε να περιέχει μία στατική σταθερά που ονομάζεται *sharedInstance*, την οποία χρησιμοποιούμε για να έχουμε

πρόσβαση στις μεθόδους της από όλα τα σημεία της εφαρμογής. Επίσης, η κλάση περιέχει μία λίστα με όλες τις αποθηκευμένες τοπολογίες ώστε να είναι εύκολα προσβάσιμες από το μενού επιλογής τοπολογίας.

3.4.9.3 Αποθήκευση τοπολογιών

Για την αποθήκευση μίας τοπολογίας και των φορτίων της δημιουργήσαμε τη μέθοδο *saveTopologyToCoreData*, η οποία δέχεται ως παραμέτρους τα φορτία της τρέχουσας τοπολογίας καθώς και το όνομα, την περιγραφή και το στιγμιότυπο που την περιγράφουν. Η διαδικασία που ακολουθεί για το μοντέλο της τοπολογίας και για κάθε φορτίο είναι η εξής:

1. Δημιουργούμε ένα αντικείμενο της ανάλογης κλάσης περνώντας ως παράμετρο τον εικονικό χώρο που δημιουργήσαμε στην κλάση *PersistenceService*, δηλαδή το *PersistenceService.context*.

2. Αναθέτουμε τις κατάλληλες τιμές στις ιδιότητες του αντικειμένου. Για παράδειγμα στο αντικείμενο τοπολογίας θα αναθέσουμε τις τιμές για το όνομα, την περιγραφή και το στιγμιότυπο της.

3. Αποθηκεύουμε τις αλλαγές στον εικονικό χώρο καλώντας τη μέθοδο *saveContext* της κλάσης *PersistenceService*.

Παρακάτω παραθέτουμε το κομμάτι κώδικα για την αποθήκευση ενός αντικειμένου *NSTopology* σύμφωνα με τα παραπάνω βήματα

```
// Save topology info
let topology = NSTopology(context: PersistenceService.context)
topology.name = name
topology.descr = description
topology.image = capturedImage
PersistenceService.saveContext()
```

3.4.9.4 Διαγραφή αποθηκευμένων τοπολογιών

Για τη διαγραφή αποθηκευμένης τοπολογίας από την εφαρμογή δημιουργήσαμε τη μέθοδο *deleteSavedTopologyFromCoreData*, η οποία δέχεται ως παράμετρο το μοντέλο τοπολογίας που αντιστοιχεί στο αντικείμενο *NSTopology* που επιθυμούμε να διαγράψουμε. Για να βρούμε το αντικείμενο αυτό μέσα από τη λίστα με τις αποθηκευμένες τοπολογίες αρκεί να την προσπελάσουμε ελέγχοντας τον κωδικό ταυτοποίησης του κάθε αντικειμένου. Για να πάρουμε τη λίστα με τις αποθηκευμένες τοπολογίες από τη μνήμη της εφαρμογής, δημιουργούμε ένα αίτημα τύπου *NSFetchRequest* προς στο Core Data και στη συνέχεια το εκτελούμε μέσω ενός αιτήματος *fetch* της ιδιότητας *PersistenceService.context* όπως φαίνεται παρακάτω.

```
let fetchRequest: NSFetchRequest<NSTopology> = NSTopology.fetchRequest()
do {
    let savedTopos = try PersistenceService.context.fetch(fetchRequest)
    //….
} catch { print("No saved topologies!") }
```

Στη συνέχεια, ψάχνουμε στη λίστα αυτή μέχρι να βρούμε την αποθηκευμένη τοπολογία με τον κατάλληλο κωδικό ταυτοποίησης. Μόλις τη βρούμε, τη διαγράφουμε καλώντας τη μέθοδο *delete*.

```
for topo in savedTopos {
    if topology.id == topo.id {
        /// Delete the NS topo from Core Data
        PersistenceService.context.delete(topo)
    }
}
```

3.4.9.5 Φόρτωση αποθηκευμένων τοπολογιών

Ο λόγος που αποθηκεύουμε τις τοπολογίες είναι για να μπορεί ο χρήστης να τις επιλέξει και χρησιμοποιήσει μέσα από ένα μενού, επομένως πρέπει να φορτώσουμε τις αποθηκευμένες τοπολογίες από τη μνήμη της εφαρμογής στο μενού. Για το σκοπό αυτό δημιουργήσαμε τη μέθοδο *loadSavedTopologies*, στην οποία αρχικά δημιουργούμε ένα αίτημα προς το *PersistenceService.context* για να πάρουμε τη λίστα με τις αποθηκευμένες τοπολογίες, όπως είδαμε προηγουμένως και στη περίπτωση διαγραφής τοπολογιών. Στη συνέχεια, προσπελάζουμε τη λίστα αυτή και δημιουργούμε τις κλάσεις-μοντέλα που αντιπροσωπεύουν τις αποθηκευμένες τοπολογίες και τα φορτία τους.

```
/// Convert NSTopology items to TopologyModel items
for topo in savedTopos {
    let newTopo = TopologyModel(id: topo.id, pointCharges: [], image: UIImage(data:
topo.image!)!, name: topo.name ?? "", description: topo.descr ?? "")
    for pointCharge in topo.pointCharges! {
        let p: NSPointCharge = (pointCharge as AnyObject) as! NSPointCharge

        let x: Float = p.posX
        let y: Float = p.posY
        let z: Float = p.posZ
        let pos: SIMD3<Float> = SIMD3<Float>(x, y, z)

        let val: Float = p.value
```

```
        let newPoint = PointChargeModel(position: pos, value: val)

        newTopo.addPointChargeModel(model: newPoint)
    }
    /// Save TopologyModel to savedTopologies[]
    savedTopologies.append(newTopo)
}
```

Μόλις ολοκληρωθεί αυτή η διαδικασία, η λίστα *savedTopologies* της κλάσης *TopologyStore* διαθέτει όλα τα μοντέλα για τις τοπολογίες και τα φορτία τους και είμαστε σε θέση να τα χρησιμοποιήσουμε για να δημιουργήσουμε αντικείμενα *Topology* και *PointChargeClass* τα οποία θα χρησιμοποιήσουμε στην εφαρμογή.

3.4.10 Όψη επισκόπησης γωνιών διανυσμάτων

Η ταυτόχρονη χρήση πολλών τρισδιάστατων μοντέλων επιβαρύνουν σημαντικά την εφαρμογή μας, ειδικά όταν αυτά ενημερώνονται πολλές φορές το δευτερόλεπτο. Όσο ο όγκος της διεργασίας απόδοσης αυξάνεται, παρατηρείται μείωση των καρέ ανά δευτερόλεπτο (frames per second, fps), κάτι το οποίο κάνει την εμπειρία του χρήστη λιγότερο ευχάριστη και ρεαλιστική. Παρακάτω αναλύουμε περισσότερο το πρόβλημα αυτό, αλλά και τον τρόπο με τον οποίο το αντιμετωπίσαμε στη δική μας εφαρμογή.

3.4.10.1 Η μείωση στην απόδοση της εφαρμογής λόγω των τρισδιάστατων μοντέλων και πώς την αντιμετωπίσαμε

Η εφαρμογή AR Coulomb περιέχει πολλά τρισδιάστατα μοντέλα όπως διανύσματα δυνάμεων, φορτία, επιγραφές και άλλα. Επομένως προσπαθήσαμε κατά τη δημιουργία της εφαρμογής να μειώσουμε τον όγκο της διεργασίας απόδοσης (rendering) ώστε η εμπειρία του χρήστη να είναι όσο το δυνατόν πιο ρεαλιστική γίνεται. Μία τεχνική που χρησιμοποιήσαμε ήταν στις αλλαγές τοποθεσίας των φορτίων να κρύβουμε τα διανύσματα δυνάμεων και τις επιγραφές τους, και να κρατάμε μόνο τους δείκτες απόστασης μεταξύ των φορτίων, καθώς κατά τη μετακίνηση ενός φορτίου το μόνο που ενδιαφέρει το χρήστη είναι να βλέπει τις αποστάσεις των φορτίων ώστε να γνωρίζει πότε θα σταματήσει. Με αυτό τον τρόπο η εφαρμογή δε χρειάζεται να αποδίδει όλα τα μοντέλα της σκηνής, τα οποία θα έπρεπε να ανανεώνονται πολλές φορές το δευτερόλεπτο λόγω των συνεχών αλλαγών στις τιμές των διαφόρων μεγεθών. Ως αποτέλεσμα, τα καρέ ανά δευτερόλεπτο παραμένουν υψηλά.

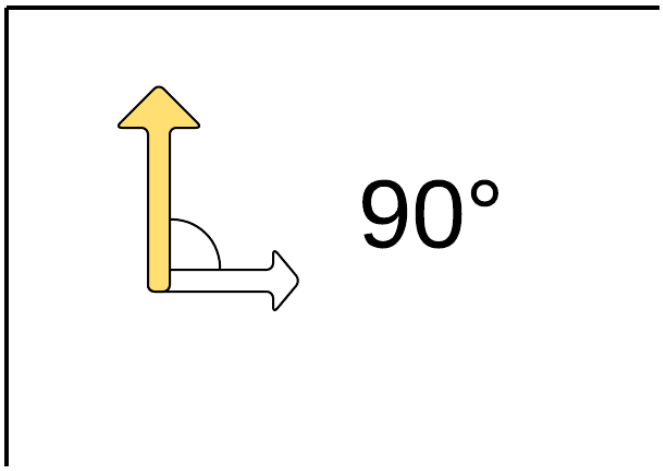
Εκτός από την απόσταση των φορτίων κατά τη μετακίνησή τους, ο χρήστης πρέπει να γνωρίζει και τις μεταξύ τους γωνίες. Μία ιδέα ήταν η δημιουργία τρισδιάστατων μοντέλων των γωνιών, τα οποία θα ανανεώνονται συνεχώς βάσει των νέων τιμών των γωνιών των φορτίων. Αυτό όμως θα πρόσθετε σημαντικό επιπρόσθετο φόρτο δεδομένων που η εφαρμογή θα έπρεπε να αποδώσει γραφικά, κάτι το οποίο προσπαθούμε να αποφύγουμε. Άλλωστε, με τα εργαλεία που προσφέρει

μέχρι σήμερα το ARKit και το RealityKit, ο βαθμός πολυπλοκότητας για τη δημιουργία του γεωμετρικού τρισδιάστατου μοντέλου μίας γωνίας είναι αρκετά υψηλός, καθιστώντας αυτόν τον τρόπο αναπαράστασης γωνιών απαγορευτικό.

3.4.10.2 Η λύση για την επισκόπηση των γωνιών με δισδιάστατα γραφικά

Για την αποδοτική αναπαράσταση των γωνιών των δυνάμεων καταλήξαμε στη δημιουργία μίας όψης η οποία θα προβάλει τις γωνίες ως δισδιάστατα γραφικά και θα συνυπάρχει με τη σκηνή επαυξημένης πραγματικότητας. Επομένως, ακόμα και κατα τη μετακίνηση των φορτίων, ο χρήστης θα είναι σε θέση να παρακολουθεί τις αποστάσεις μεταξύ των φορτίων μέσω τρισδιάστατων δεικτών, αλλά και τις γωνίες των διανυσμάτων δυνάμεων μέσω της όψης προβολής γωνιών, χωρίς να επιβαρύνεται ο όγκος της διεργασίας απόδοσης και διατηρώντας την εμπειρία ευχάρτιστη.

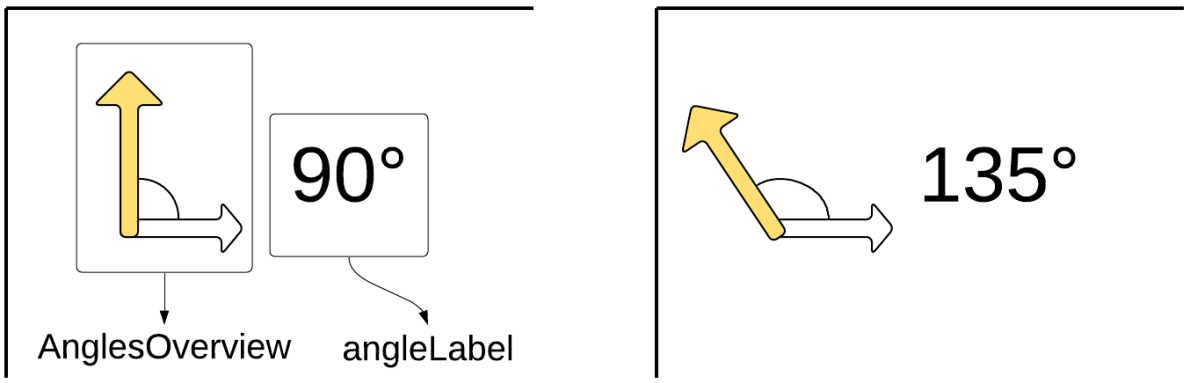
Δημιουργήσαμε την κλάση *AnglesOverview* που κληρονομεί από την κλάση *UIView*. Πρόκειται για την όψη στην οποία θα σχεδιάζονται τα διανύσματα των δυνάμεων που δέχεται το επιλεγμένο φορτίο, και θα σχεδιάζεται και η γωνία μίας εκ των δυνάμεων που θα επιλέγει ο χρήστης. Ο τρόπος με τον οποίο θα επιλέγει ο χρήστης μία δύναμη είναι να πατήσει (όχι παρατεταμένα) ή να σύρει το φορτίο από την οποία προέρχεται Αν ο χρήστης θέλει να προβάλλεται η γωνία της συνισταμένης δύναμης που δέχεται το επιλεγμένο φορτίο, αρκεί να πατήσει ή να σύρει το ίδιο επιλεγμένο φορτίο.



Σχήμα 3.17: Η όψη AnglesOverview στην πάνω αριστερή γωνία της οθόνης παρουσιάζει τη γωνία της δύναμης με τη χρήση δισδιάστατων γραφικών.

3.4.10.3 Η όψη angleLabel της κλάσης UILabel σε συνδυασμό με την όψη της κλάσης AnglesOverview

Η όψη της κλάσης *AnglesOverview* προβάλει τη γραφική αναπαράσταση των διανυσμάτων και γωνιών. Για την αριθμητική αναπαράσταση της επιλεγμένης γωνίας δημιουργήσαμε μία ακόμα όψη, την *angleLabel* η οποία είναι τύπου UILabel. Αυτή η όψη ενημερώνεται διαρκώς όπως και η όψη της κλάσης *AnglesOrverview* για την επιλεγμένη γωνία και την τιμή της, και την αναπαραστικά ως αριθμό. Τοποθετήσαμε τη αριθμητική αναπαράσταση δίπλα από τη γραφική, ώστε να είναι εύκολο για το χρήστη να παρακολουθεί και τις δύο ταυτόχρονα. Οι δύο όψεις μαζί αποτελούν την όψη επισκόπησης γωνιών της εφαρμογής μας.



Σχήμα 3.18: Η όψη *AnglesOverview* με τη σχηματική αναπαράσταση και η όψη *angleLabel* με την αριθμητική τιμή της γωνίας της επιλεγμένης δύναμης.

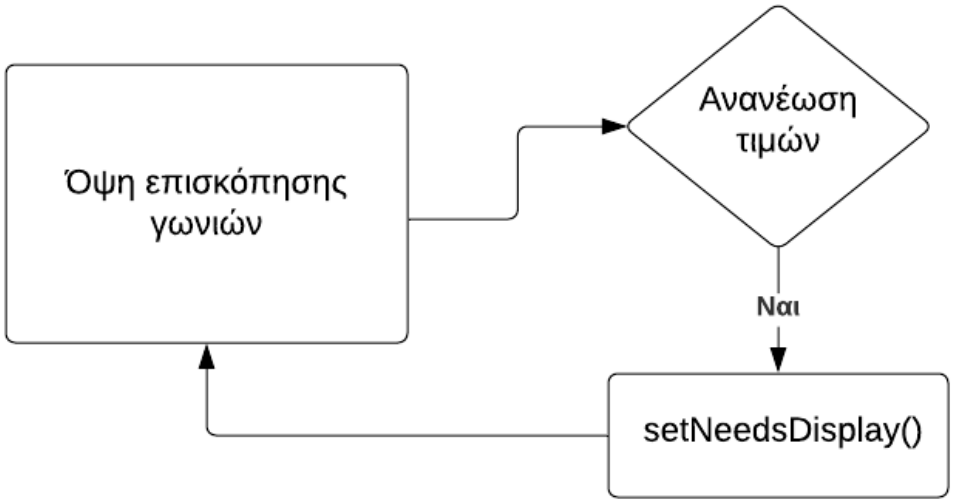
3.4.10.4 Αρχικοποίηση και ανανέωση της όψης στην εφαρμογή AR Coulomb

Οι δύο όψεις αρχικοποιούνται στην κλάση *ViewController*. Μόλις η κλάση *ViewController* εκκινήσει, καλούμε δύο μεθόδους που αφορούν την παραμετροποίηση των όψεων.

- **configureAngleOverview**: Αφορά την παραμετροποίηση της όψης τύπου *AnglesOverview*, και τη στοίχιση της στο πάνω αριστερά μέρος της οθόνης.
- **configureAngleLabel**: Αφορά την παραμετροποίηση της όψης *angleLabel* και τη στοίχιση της δίπλα από την όψη τύπου *AnglesOverview.* Επιπρόσθετα, ορίζεται η συμοβολοσειρά της όψης να προβάλει την τιμή της επιλεγμένης δύναμης.

Στην αρχή της εμπειρίας, όταν και εκκινεί η κλάση *ViewController* και καλούνται αυτές οι δύο μέθοδοι, κρύβουμε αρχικά τις όψεις καθώς δεν έχει επιλεχθεί ακόμα καμία τοπολογία. Οι όψεις εμφανίζονται μόνο όταν υπάρχει μία τοπολογία τοποθετημένη στη σκηνή. Ο τρόπος με τον οποίο ανανεώνονται οι όψεις, δηλαδή η γραφική αλλά και η αριθμητική αναπαράσταση των γωνιών,

είναι πολύ απλός. Οι όψεις αυτές λαμβάνουν τις τιμές των γωνιών κατά την κλήση της μεθόδου *draw* και προβάλλουν την ανάλογη αναπαράσταση. Οπότε αρκεί κάθε φορά που θέλουμε να ανανεώσουμε την αναπαράσταση να βρούμε τρόπο να καλούμε ξανά τη μέθοδο *draw* στην κάθε όψη. Ο τρόπος για να το καταφέρουμε αυτό είναι να καλούμε τη μέθοδο *setNeedsDisplay* στην κάθε όψη. Η μέθοδος αυτή αναλαμβάνει να ξανασχεδιάσει την όψη, κι επομένως να λάβει τις ανανεωμένες τιμές για να προβάλει την ανάλογη αναπαράσταση.



Σχήμα 3.19: Κάθε φορά που ανανεώνονται οι δυνάμεις της τοπολογίας, καλείται η μέθοδος *setNeedsDisplay* για την ενημέρωση της όψης επισκόπησης γωνιών.

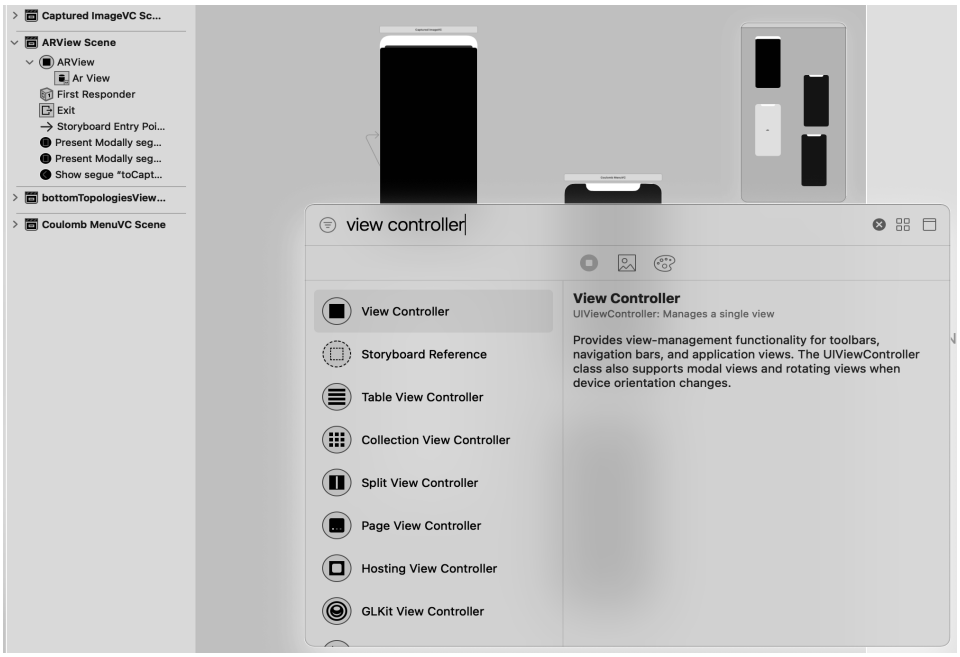
3.4.11 Επιπρόσθετες όψεις τύπου UIViewController

Καθώς επιθυμούμε να δώσουμε στο χρήστη περισσότερο έλεγχο πάνω στην αναπαράσταση του νόμου Coulomb στην εφαρμογή μας, δημιουργήσαμε ορισμένες όψεις τύπου *UIViewController* στις οποίες του δίνουμε τη δυνατότητα να επεξεργαστεί ορισμένα μεγέθη της εφαρμογής. Συγκεκριμένα, δημιουργήσαμε τρεις όψεις, μία για την επεξεργασία των αποθηκευμένων τοπολογιών, μία για την επεξεργασία των φορτίων και μία για την επισκόπηση του στιγμιοτύπου της τοπολογίας κατά τη διαδικασία αποθήκευσής της.

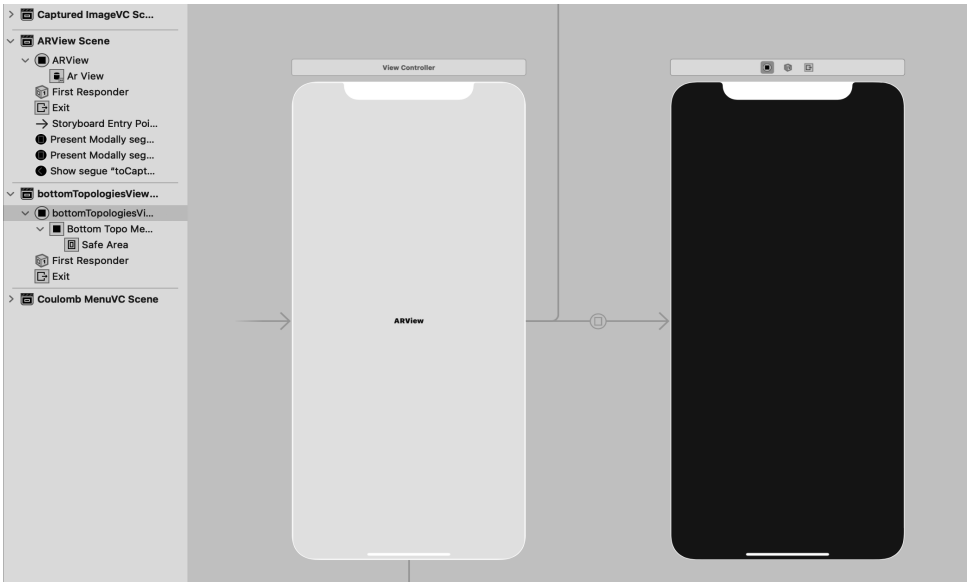
Για τη δημιουργία μίας όψης *UIViewController* ακολουθήσαμε τα παρακάτω βήματα:

1) Μετάβαση στο αρχείο “Main.storyboard”
2) Άνοιγμα της βιβλιοθήκης και αναζήτησης της όψης τύπου “View Controller”
3) Εισαγωγή της όψης στο *storyboard* καθώς και εισαγωγή κατάλληλου ονόματος

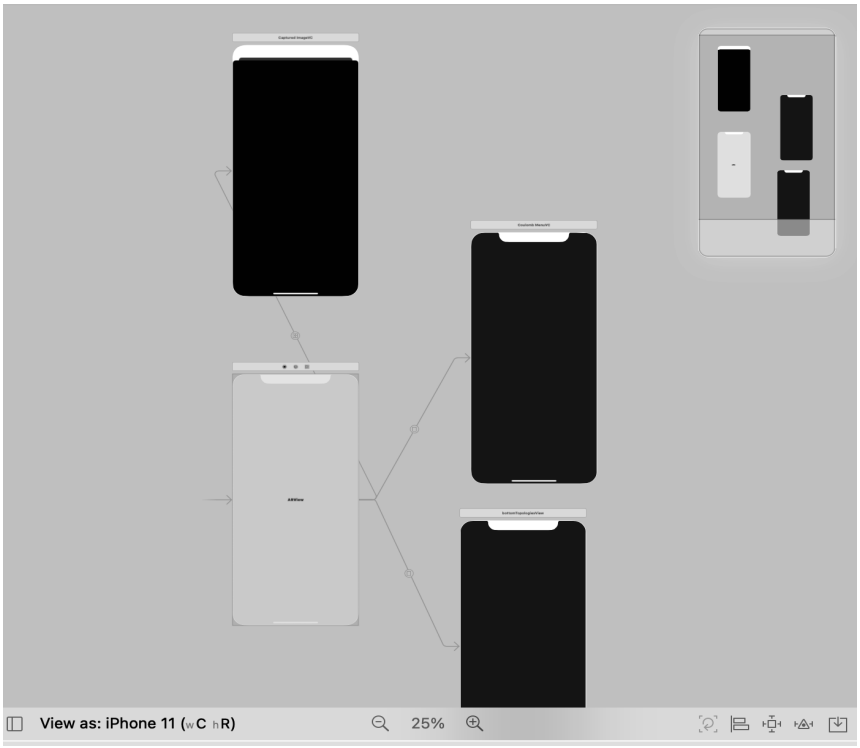
4) Σύνδεση της νέας όψης με την κύρια όψη “View Controller”



Εικόνα 3.1: Εύρεση της όψης “View Controller” από τη βιβλιοθήκη του Xcode.



Εικόνα 3.2: Η όψη στο storyboard του Xcode.



Εικόνα 3.3: Οι συνδέσεις των επιπρόσθετων όψεων με την κύρια όψη.

3.4.11.1 Μενού επιλογής τοπολογίας (BottomTopoMenuVC)

Το μενού επιλογής τοπολογίας αποτελείται από τρεις βασικές περιοχές. Η πρώτη και κύρια εκτείνεται στο σύνολο της οθόνης και αποτελεί την επισκόπηση του στιγμιότυπου της επιλεγμένης τοπολογίας. Η δεύτερη περιέχει τον τίτλο και την περιγραφή της επιλεγμένης τοπολογίας που φαίνεται στο φόντο στην κύρια περιοχή και η τρίτη περιέχει ένα οριζόντια συρόμενο μενού στο πάνω μέρος της οθόνης που περιέχει όλες τις τοπολογίες και επιτρέπει στο χρήστη να επιλέγει ποια θα προεπισκοπηθεί στην κύρια περιοχή. Εκτός των τριών βασικών περιοχών, η όψη BottomTopoMenuVC περιέχει και τρία κουμπιά. Ένα για το κλείσιμο της όψης, ένα για την επιλογή της προεπισκοπούμενης τοπολογίας και ένα για τη διαγραφή της.

Κατά την εκκίνηση της όψης φορτώνονται όλες οι αποθηκευμένες τοπολογίες από τη μνήμη της εφαρμογής με τη μέθοδο *TopologyStore.reloadSavedTopologies*.

```
/// Reload savedTopologies, cause Object Identifiers for custom saved topos
/// change for some reason. Also, do the same before deleting!
TopologyStore.sharedInstance.reloadSavedTopologies()
```

Με την εκτέλεση αυτής της μεθόδου ενημερώνεται η λίστα αποθηκευμένων τοπολογιών *savedTopologies* της κλάσης *TopologyStore* κι είναι εύκολη πλέον η δημιουργία του οριζόντιου συρόμενου μενού τοπολογιών στο πάνω μέρος της οθόνης. Για τη λειτουργικότητα ενός οριζόντια συρόμενου μενού δημιουργήσαμε μία όψη *UIStackView* με οριζόντιο προσανατολισμό μέσα σε μία όψη *UIScrollView*. Στη συνέχεια δημιουργήσαμε ένα κουμπί για κάθε τοπολογία από τη λίστα των αποθηκευμένων τοπολογιών και το προσθέσαμε στην όψη *UIStackView*.

Για να μπορούμε να προβάλουμε στο χρήστη το στιγμιότυπο και τις πληροφορίες της τοπολογίας που επιλέγει από το οριζόντια συρόμενο μενού στο πάνω μέρος της οθόνης δημιουργήσαμε μία μεταβλητή με όνομα *selectedTopo* τύπου *TopologyModel* στην όψη *BottomTopoMenuVC*. Κάθε φορά που ο χρήστης επιλέγει μία τοπολογία από το συρόμενο μενού ενημερώνουμε αυτή την μεταβλητή ώστε να αντιπροσωπεύει την επιλεγμένη τοπολογία.

```
@objc func buttonSelectTopoAction(sender: UIButton!) {
    self.selectTopo(button: sender)
}
```

Στη συνέχεια αρκεί να φορτώσουμε το στιγμιότυπο που περιέχει στο φόντο της όψης και τις πληροφορίες στην περιοχή επισκόπησης τους στο κάτω μέρος της οθόνης. Για το σκοπό αυτό δημιουργήσαμε τη μέθοδο *selectTopo* η οποία καλείται κατά το πάτημα μίας τοπολογίας από το οριζόντιο συρόμενο μενού.

```
private func selectTopo(button: UIButton) {
    let totalTopologies = TopologyStore.sharedInstance.totalTopologies()
    if button.tag > -1 && button.tag < (totalTopologies) {
        /// Set the new topology
        self.selectedTopo = TopologyStore.sharedInstance.savedTopologies[button.tag]

        /// De-highlight all StackView buttons
        self.stackView.arrangedSubviews.forEach{ btn in
            btn.layer.borderColor = UIColor.white.withAlphaComponent(0).cgColor
        }

        /// Highlight the selected one
        button.layer.borderColor = UIColor.white.cgColor

        /// Update Preview Backgound Image
        self.updatePreview()
    }
}
```

Από τη στιγμή που ο χρήστης επιλέξει μία τοπολογία από το συρόμενο μενού για επισκόπηση, καλείται είτε να την επιλέξει για να τη φορτώσει στη σκηνή πατώντας το κουμπί επιλογής, είτε να τη διαγράψει πατώντας το κουμπί διαγραφής αντίστοιχα. Τα δύο κουμπιά βρίσκονται κάτω από την περιοχή επισκόπησης πληροφοριών της τοπολογίας.

Κατά το πάτημα του κουμπιού επιλογής, αποστέλλεται μία ειδοποίηση μέσω του Notification Center μεταφέροντας ως πληροφορία την επιλεγμένη τοπολογία, ώστε να ενημερωθεί η κύρια όψη της εφαρμογής μας για τη νέα επιλεγμένη τοπολογία που καλείται να φορτώσει στη σκηνή. Μόλις αποσταλεί η ειδοποίηση, κλείνουμε την όψη *BottomTopoMenuVC* και επιστρέφουμε στην κύρια όψη με την κλήση της μεθόδου *dismiss*.

Σε περίπτωση που ο χρήστης πατήσει το κουμπί διαγραφής, βρίσκουμε την τοπολογία προς διαγραφή από τη λίστα αποθηκευμένων που διατηρεί η κλάση *TopologyStore*. Αφού την αφαιρέσουμε, καλούμε τη μέθοδο *TopologyStore.reloadSavedTopologies* για να βεβαιωθούμε πως διαθέτουμε την ενημερωμένη εκδοχή της λίστας. Τέλος, ενημερώνουμε το οριζόντια συρόμενο μενού τοπολογιών στο πάνω μέρος της οθόνης και επιλέγουμε τυχαία μία τοπολογία για επισκόπηση πληροφοριών καθώς η προηγούμενη επιλεγμένη μόλις διαγράφηκε.

3.4.11.2 Μενού επεξεργασίας φορτίου (CoulombMenuVC)

Το μενού επεξεργασίας φορτίου χωρίζεται σε δύο βασικές περιοχές, την περιοχή επεξεργασίας τιμής του φορτίου στο κάτω μέρος της οθόνης και το κουμπί διαγραφής του στην πάνω δεξιά γωνία της οθόνης. Η περιοχή επεξεργασίας τιμής αποτελείται από ένα ολισθαίνων εργαλείο, στο οποίο θα ανεφερόμαστε ως *slider* και με το οποίο μπορεί ο χρήστης να αυξομειώνει την τιμή του φορτίου. Εκτός του *slider*, η περιοχή επεξεργασίας διαθέτει και δύο κουμπιά μείωσης και αύξησης της τιμής αριστερά και δεξιά του *slider* αντίστοιχα. Τέλος, πάνω από το *slider* βρίσκεται μία επιγραφή της τρέχουσας τιμής του φορτίου.

Για να κατασκευάσουμε αυτή τη διάταξη που περιγράψαμε προηγουμένως στην περιοχή επεξεργασίας τιμής του φορτίου δημιουργήσαμε μία όψη *UIStackView* με κάθετο προσανατολισμό που θα περιέχει ως πρώτο στοιχείο μία όψη *UILabel* που περιέχει την επιγραφή της τιμής του φορτίου και ως δεύτερο μία όψη με το *slider* και τα δύο κουμπιά αυξομείωσης. Για το *slider* και τα δύο κουμπιά δημιουργήσαμε μία ακόμα όψη *UIStackView*, η οποία έχει οριζόντιο προσανατολισμό και σε αυτήν προσθέσαμε με σειρά το κουμπί μείωσης, το *slider* και το κουμπί αύξησης. Κάθε φορά που ο χρήστης μετακινεί το δείκτη του slider καλείται η μέθοδος *sliderValueDidChange*. Η μέθοδος αυτή δέχεται ως παράμετρο το ίδιο το αντικείμενο *slider*, εξάγει την τιμή του και στη συνέχεια καλεί τη μέθοδο *updateSlider* με παράμετρο την τιμή αυτή.

```
@objc func sliderValueDidChange(_ sender: UISlider!) {
    let roundedStepValue = round(sender.value / step) * step
    slider.value = roundedStepValue
    updateSlider(constant: 0)
```

```
}
```

Η μέθοδος *updateSlider* με τη σειρά της ενημερώνει την όψη επιγραφής του φορτίου και στέλνει μία ειδοποίηση στην εφαρμογή για τη νέα τιμή του φορτίου, ώστε να ενημερωθούν οι τιμές και τα μοντέλα των δυνάμεων, καθώς και το μοντέλο επιγραφής της τιμής του φορτίου.

```
private func updateSlider(constant: Float) {
    slider.setValue(slider.value + constant, animated: true)
    textUpdate(sliderValue: slider.value)

    /// Notify Coulomb Value Change Observer
    notifyObserver(withKey: cbNotificationKey, value: slider.value)
}
```

Σε περίπτωση που ο χρήστης χρησιμοποιήσει τα κουμπιά αυξομείωσης τιμής αντί για το *slider* εκτελείται η μέθοδος *sliderButtonAction*. Η μέθοδος αυτή δέχεται ως παράμετρο το κουμπί που πατήθηκε και αναλόγως με την τιμή της ιδιότητας tag πραγματοποιείται η πράξη αφαίρεσης ή πρόσθεσης 0.5 μονάδων στην τιμή του φορτίου. Για το κουμπί μείωσης έχουμε δώσει στην ιδιότητα *tag* τιμή "0" ενώ στο κουμπί αύξησης "1". Και πάλι, καλείται η μέθοδος *updateSlider* για την ενημέρωση της εφαρμογής.

```
@objc
func sliderButtonAction(sender: UIButton) {
    if sender.tag == 0 {
        if (slider.value > slider.minimumValue) {
            updateSlider(constant: -step)
        }
    } else if sender.tag == 1 {
        if (slider.value < slider.maximumValue) {
            updateSlider(constant: step)
        }
    }
}
```

3.4.11.3 Επισκόπηση στιγμιοτύπου τοπολογίας (CapturedImageVC)

Κατά τη διαδικασία αποθήκευσης μίας τοπολογίας ο χρήστης καλείται να τραβήξει ένα στιγμιότυπο της τρέχουσας οθόνης της εφαρμογής, τέτοιο ώστε να συμπεριλαμβάνεται η τοπολογία της σκηνής σε βαθμό που τον ικανοποιεί. Για το σκοπό αυτό δημιουργήσαμε την όψη CapturedImageVC, η οποία χρησιμοποιεί μία όψη UIImageView με την εικόνα του στιγμιότυπου ως

φόντο και δύο κουμπιά αποθήκευσης και ακύρωσης, ώστε ο χρήστης να μπορεί να αποφασίσει αν θέλει να κρατήσει το στιγμιότυπο ή να το ακυρώσει και να τραβήξει ένα καινούργιο.

Τα δύο κουμπιά βρίσκονται στο ίδιο ύψος της οθόνης, δίπλα το ένα στο άλλο. Για να τα τοποθετήσουμε σε αυτό το σχηματισμό χρησιμοποιήσαμε μία όψη UIStackView με οριζόντιο προσανατολισμό στην οποία προσθέσαμε πρώτα το κουμπί ακύρωσης και έπειτα το κουμπί αποθήκευσης. Όταν πατηθεί το κουμπί ακύρωσης, καλείται η μέθοδος cancelSaveTopology που με τη σειρά της καλεί τη μέθοδο dismiss για να κλείσει η όψη CapturedImageVC και να επιστρέψει ο χρήστης στην κύρια όψη ViewController.

```
@objc func cancelSaveTopology(sender: UIButton) {
    dismiss(animated: true, completion: nil)
}
```

Όταν πατηθεί το κουμπί αποθήκευσης, καλείται η μέθοδος saveTopology η οποία με τη σειρά της στέλνει την κατάλληλη ειδοποίηση στην υπόλοιπη εφαρμογή μέσω του Notification Center και μεταφέροντας ως πληροφορία το στιγμιότυπο σε μορφή pngData. Μόλις σταλθεί η ειδοποίηση καλείται η μέθοδος dismiss ώστε να επιστρέψει και πάλι ο χρήστης στην κύρια όψη της εφαρμογής.

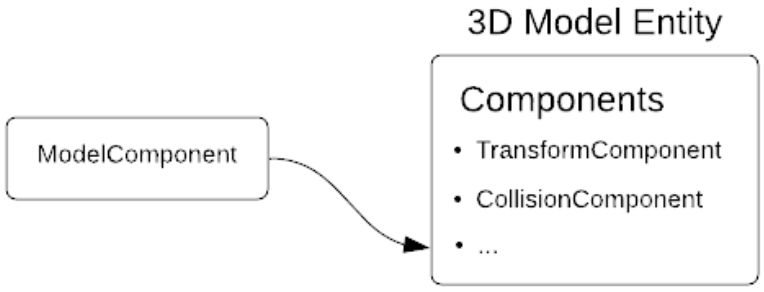
```
@objc func saveTopology(sender: UIButton) {
    /// Dismiss the CapturedImageVC by sending a notification
    self.dismiss(animated: true, completion: {

        /// Notify the observer (view controller) and pass the image binary data
        let notifName = Notification.Name(rawValue: photoTakenNotificationKey)
        let valueDict: [String: Data] = ["imageData": self.image!.pngData()!]
        NotificationCenter.default.post(
            name: notifName,
            object: nil,
            userInfo: valueDict
        )
    })
}
```

3.5 Υλοποίηση συνεχούς ενημέρωσης των τρισδιάστατων μοντέλων

3.5.1 Τα τρισδιάστατα μοντέλα και τα συστατικά τους

Στην εφαρμογή μας χρησιμοποιούμε τρισδιάστατα μοντέλα για την απεικόνιση διαφόρων στοιχείων μίας τοπολογίας χρησιμοποιώντας το εργαλείο RealityKit, όπου κάθε μοντέλο εκπροσωπείται από μία οντότητα. Το RealityKit προσφέρει ένα δενδρικό σύστημα οντοτήτων και κάθε αντικείμενο-οντότητα ανήκει στην κλάση *Entity*. Βάσει αυτού του δενδρικού συστήματος, κάθε σκηνή περιέχει μία οντότητα-ρίζα, και κάθε οντότητα που προστίθεται στη σκηνή αποτελεί παιδί της, εγγόνι κλπ.

Κάθε οντότητα αποτελείται από συστατικά, και μπορεί να έχει διαφορετικές ιδιότητες ανάλογα με τα συστατικά από τα οποία αποτελείται. Οι οντότητες των τρισδιάστατων μοντέλων πρέπει να περιέχουν το συστατικό μοντέλου, *ModelComponent*, το οποίο είναι υπεύθυνο για την οπτική αναπαράσταση της οντότητας, όπως οι διαστάσεις της, το χρώμα και η υφή της. Στην εφαρμογή μας δημιουργούμε για κάθε μοντέλο ένα συστατικό μοντέλου με τις διαστάσεις και το χρώμα που θέλουμε να έχει, και στη συνέχεια το εισάγουμε στην οντότητα του μοντέλου.



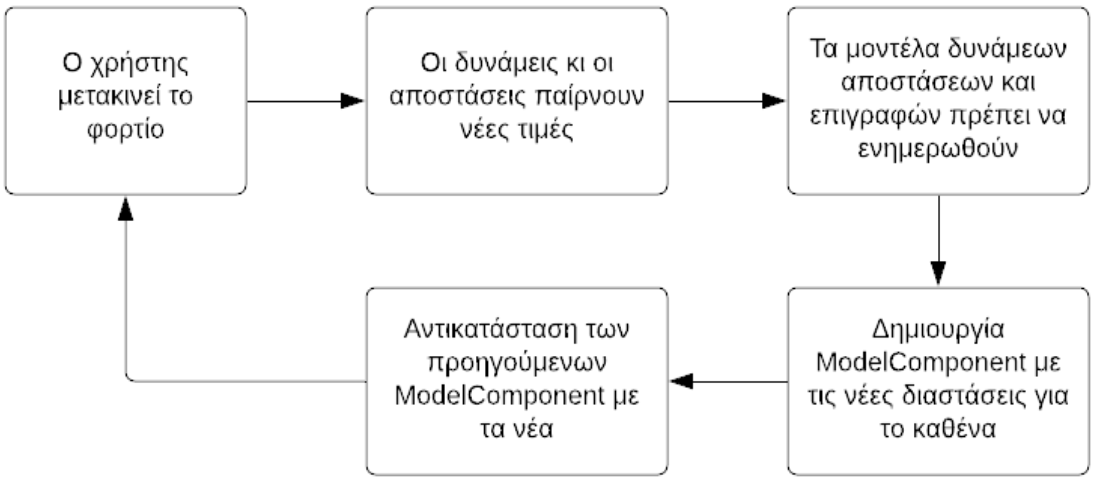
Σχήμα 3.20: Το συστατικό μοντέλου τοποθετείται στη λίστα συστατικών του τρισδιάστατου μοντέλου.

Από τη στιγμή που το μοντέλο λάβει το συστατικό μοντέλου, οι οπτικές του ιδιότητες δεν μπορούν να αλλάξουν, καθώς το RealityKit δε δίνει πρόσβαση στον προγραμματιστή στο συστατικό και τις ιδιότητες που του έχουμε δώσει. Αν για παράδειγμα, δημιουργήσουμε ένα μοντέλο ευθύγραμμου τμήματος και δώσουμε στο συστατικό του διαστάσεις μήκους 2 μέτρων, τότε από εκεί και πέρα δε δίνεται η δυνατότητα να μεταβάλουμε το μήκος αυτό. Προφανώς αυτό αποτελεί ένα σημαντικό

εμπόδιο για την περάτωση της εφαρμογής μας, καθώς επιθυμούμε τα μοντέλα να αυξομειώνουν τις διαστάσεις τους καθώς ο χρήστης αλληλεπιδρά με την τοπολογία.

3.5.2 Πώς αντιμετωπίσαμε το πρόβλημα ενημέρωσης των μοντέλων

Το γεγονός ότι η ενημέρωση του συστατικού μοντέλου, *ModelComponent*, μίας οντότητας κατά τη διάρκεια λειτουργίας της εφαρμογής είναι αδύνατη δεν καθιστά και την αυξομείωση των διαστάσεων των μοντέλων ανέφτικη. Για να το καταφέρουμε αυτό σε μία οντότητα, δεν της αναθέτουμε ένα συστατικό μοντέλου από την αρχή μέχρι το τέλος της ζωής της, αλλά πολλά συστατικά τα οποία αντικαθίστανται με σειρά. Κάθε φορά που οι χειρονομίες του χρήστη επιβάλλουν την μεταβολή των τρισδιάστατων μοντέλων, ξεκινάει μία διαδικασία κατά την οποία δημιουργείται ένα συστατικό μοντέλου με νέες διαστάσεις για κάθε οντότητα, το οποίο αντικαθιστά το προηγούμενο συστατικό που η οντότητα είχε ήδη. Η διαδικασία αυτή δημιουργίας και αντικατάστασης συστατικών μοντέλου στις οντότητες επαναλαμβάνεται μέχρι ο χρήστης να σταματήσει να αλληλεπιδρά με την τοπολογία και τα μεγέθη να παραμείνουν σταθερά.



Σχήμα 3.21: Ο κύκλος δημιουργίας και αντικατάστασης συστατικών μοντέλων για κάθε τρισδιάστατο μοντέλο που θέλουμε να ενημερώσουμε τις διαστάσεις του.

Ένα συστατικό μοντέλου *ModelComponent* δέχεται ένα πλέγμα και μία λίστα από υλικά. Το πλέγμα είναι η σχηματική αναπαράσταση του μοντέλου με τις κατάλληλες διαστάσεις. Κάθε φορά που θέλουμε να αλλάξουμε τις διαστάσεις του μοντέλου, δημιουργούμε ένα νέο συστατικό μοντέλου με πλέγμα το σχήμα του μοντέλου σε κατάλληλες διαστάσεις. Για τη δημιουργία του πλέγματος ενός μοντέλου μπορούμε να στραφούμε στο εργαλείο RealityComposer σε περίπτωση που το σχήμα

είναι περίπλοκο. Σε διαφορετική περίπτωση, το RealityKit προσφέρει ορισμένες μεθόδους για τη δημιουργία απλών σχημάτων, όπως η σφαίρα και το ορθογώνιο παραλληλεπίπεδο.

Στη δική μας εφαρμογή χρησιμοποιήσαμε το εργαλείο Reality Composer μόνο για τη δημιουργία της τριγωνικής κεφαλής των διανυσμάτων των δυνάμεων, καθώς οι μέθοδοι του RealityKit δεν αρκούσαν για τη δημιουργία ενός τριγωνικού μοντέλου. Οι διαστάσεις της τριγωνικής κεφαλής των δυνάμεων δεν αλλάζουν ποτέ, επομένως δε χρειάζεται να δημιουργούμε νέο τριγωνικό πλέγμα για τα μοντέλα των δυνάμεων. Για τα υπόλοιπα μοντέλα, όπως τα φορτία, τα σώματα των διανυσμάτων, τους δείκτες αποστάσεων και τις επιγραφές, χρησιμοποιήσαμε τις μεθόδους δημιουργίας που προσφέρει το RealityKit. Παρακάτω παραθέτουμε τον κώδικα που φτιάξαμε για το σκοπό της συνεχούς ενημέρωσης του μοντέλου σώματος διανύσματος δύναμης.

Δημιουργήσαμε τη μέθοδο *load_ArrowBody_ModelComponent()*, η οποία δέχεται ως όρισμα το μήκος του διανύσματος δύναμης και δημιουργεί ένα συστατικό μοντέλου ευθύγραμμου τμήματος το οποίο αντιπροσωπεύει το σώμα του διανύσματος. Για τη δημιουργία του χρησιμοποιούμε τη μέθοδο *generateBox()* του RealityKit, στην οποία παρέχουμε τις διαστάσεις του ευθύγραμμου τμήματος βάσει του μήκους που δόθηκε στην αρχή. Επιπλέον, για να δώσουμε χρώμα στο ευθύγραμμο τμήμα δημιουργούμε και προσθέτουμε στο μοντέλο το κατάλληλο υλικό *SimpleMaterial.* Επομένως, με τη μέθοδο *load_ArrowBody_ModelComponent()* είμαστε σε θέση να δημιουργούμε ανά πάσα στιγμή ένα συστατικό μοντέλου ευθύγραμμου τμήματος με το μήκος που μας εξυπηρετεί σε κάθε περίπτωση.

```
private func load_ArrowBody_ModelComponent(length: Float) -> ModelComponent{
    let material: SimpleMaterial = {
        var mat = SimpleMaterial()
        mat.metallic = MaterialScalarParameter(floatLiteral: 0.2)
        mat.roughness = MaterialScalarParameter(floatLiteral: 0.1)
        mat.tintColor = UIColor.white
        return mat
    }()

    let model: ModelComponent =
        ModelComponent(
            mesh: .generateBox(width: 0.001, height: 0.001, depth: length),
            materials: [material]
        )
    return model
}
```

Έχοντας στη διάθεσή μας τη μέθοδο δημιουργίας του μοντέλου δίνοντας ως όρισμα το μήκος που επιθυμούμε, την καλούμε κατ εξακολούθηση όποτε ο χρήστης πραγματοποιεί αλλαγές στην τοπολογία. Κάθε φορά που παίρνουμε το συστατικό μοντέλου από τη μέθοδο, προσκολλάται στη λίστα των συστατικών που διατηρεί η οντότητα του μοντέλου που μας ενδιαφέρει, όπως η

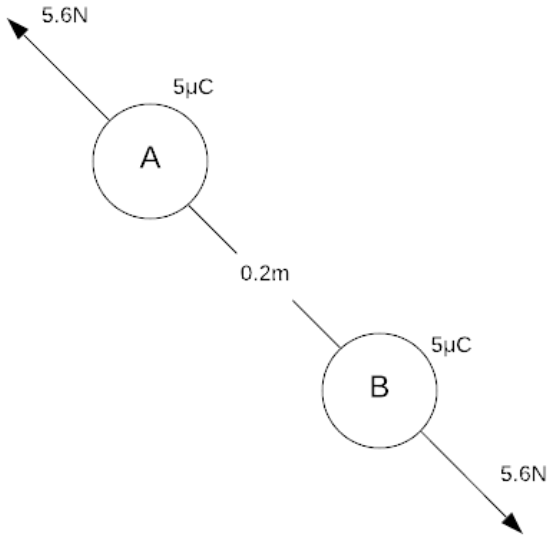
οντότητα του σώματος διανύσματος σε αυτή την περίπτωση. Όταν προσθέτουμε ένα συστατικό μοντέλου στη λίστα, αυτό αντικαθιστά αυτόματα οποιοδήποτε συστατικό μοντέλου υπήρχε ως πρότινος.

```
private func update_ArrowBody_Length(arrowBodyEntity: Entity, length: Float) {
    let model = load_ArrowBody_ModelComponent(length: length)
    arrowBodyEntity.components.set(model)
}
```

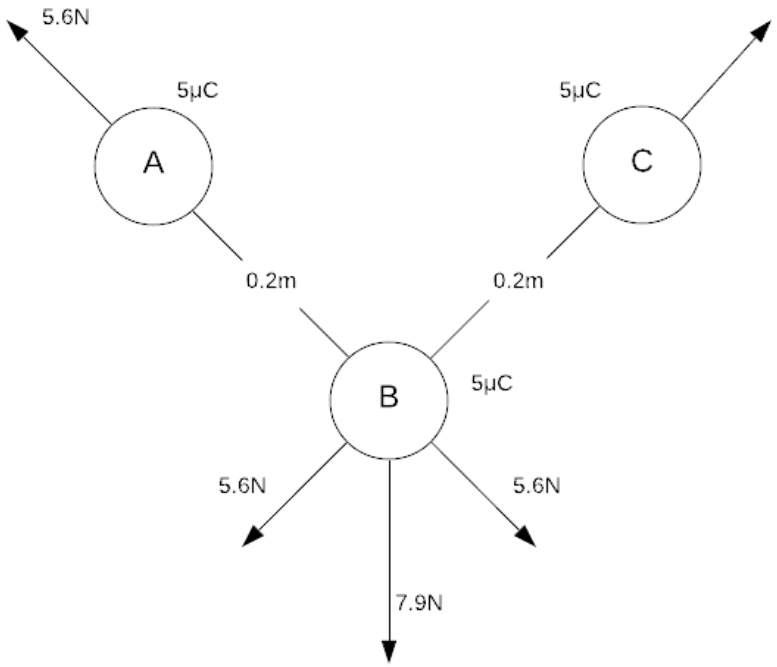
3.5.3 Η επίδοση της εφαρμογής σε σχέση με τον αριθμό μοντέλων στη σκηνή

Η απόδοση των τρισδιάστατων μοντέλων στην οθόνη του χρήστη γίνεται χάρις τον επεξεργαστή της συσκευής και απαιτεί την εκτέλεση πολλών διεργασιών από αυτόν. Η ισχύς των επεξεργαστών πλέον επιτρέπουν την άνετη λειτουργία της εφαρμογής με τρισδιάστατα μοντέλα σε ικανοποιητικό αριθμό καρέ ανά δευτερόλεπτο (frames per second, fps), κοντά στα 60. Όσο όμως αυξάνονται τα μοντέλα στη σκηνή, αυξάνεται και το υπολογιστικό φορτίο στο οποίο καλείται να ανταπεξέλθει ο επεξεργαστής. Ειδικά με τη λύση για τη συνεχή ενημέρωση των μοντέλων που προτείνουμε στην εργασία αυτή, απαιτούμε από την εφαρμογή μας να δημιουργεί επαναλαμβανόμενα πλέγματα μοντέλων με νέες διαστάσεις πολλές φορές το δευτερόλεπτο και για διαφορετικά μοντέλα ταυτόχρονα. Όπως είναι επόμενο, μετά από έναν αριθμό

Για να αναλογιστούμε λίγο το φόρτο εργασίας που εναποθέτουμε στον επεξεργαστή κατά τη λειτουργία της εφαρμογής μας, ας δούμε τα μοντέλα που απαιτούνται για κάθε ζευγάρι φορτίων. Αρχικά υπάρχουν τα δύο σφαιρικά μοντέλα φορτίων και δύο μοντέλα επιγραφών για την τιμή του φορτίου τους που τα συνοδεύουν. Έπειτα, υπάρχει το μοντέλο δείκτη απόστασης που με τη σειρά του αποτελείται από δύο μοντέλα ευθύγραμμου τμήματος και ένα μοντέλο επιγραφής για τη μεταξύ τους απόσταση. Τέλος, κάθε φορτίο δέχεται από μία δύναμη για κάθε άλλο φορτίο, ενώ κάθε δύναμη αποτελείται από το μοντέλο του σώματος του διανύσματος δύναμης, το τριγωνικό μοντέλο της κεφαλής του διανύσματος και τελικά το μοντέλο επιγραφής της τιμής της δύναμης. Όταν δε υπάρχουν πάνω από δύο φορτία στην τοπολογία, οι δείκτες αποστάσεων και οι δυνάμεις πολλαπλασιάζονται.



Σχήμα 3.22: Τα μοντέλα δυνάμεων, δεικτών αποστάσεων και επιγραφών για μία τοπολογία δύο φορτίων.



Σχήμα 3.23: Τα μοντέλα δυνάμεων, δεικτών αποστάσεων και επιγραφών για μία τοπολογία τριών φορτίων. Το φορτίο B είναι το επιλεγμένο φορτίο. Ο αριθμός μοντέλων αυξάνεται για κάθε νέο φορτίο στην τοπολογία

Επομένως, με τη συνεχή ενημέρωση των μοντέλων, δηλαδή με τη συνεχή ανακατασκευή πλεγμάτων και συστατικών μοντέλων με τις κατάλληλες διαστάσεις, η απόδοση της εφαρμογής αρχίζει να μειώνεται όσο τα φορτία αυξάνονται. Για να επιτρέψουμε στην εφαρμογή να διατηρεί υψηλές επιδόσεις και καρέ ανά δευτερόλεπτο καταφύγαμε στις δύο παρακάτω ενέργειες.

- Απόκρυψη δυνάμεων κατά τη μεταφορά των φορτίων με τη χειρονομία drag του χρήστη
- Απόκρυψη όλων των δυνάμεων εκτός από αυτών που αφορούν το επιλεγμένο φορτίο καθόλη τη διάρκεια λειτουργίας της εφαρμογής

Η μέθοδος *toggleAllForces(show: Bool)* μας επιτρέπει να κρύβουμε όλες τις δυνάμεις ταυτόχρονα όταν ξεκινάει η επαφή του χρήστη πάνω σε κάποιο φορτίο, επομένως την καλούμε μέσα στη μέθοδο *touchesBegan().*

```
// Show or Hide all forces
func toggleAllForces(show: Bool) {
    self.netForces.forEach{ netForce in
```

```
        netForce.forces.forEach{ force in
            force.arrowEntity.isEnabled = show
        }
        netForce.arrowEntity.isEnabled = show
    }
}

// MARK: - Drag & Drop PointCharge Entities
override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
    guard let location = touches.first?.location(in: self.arView) else {return}
    guard let hitEntity = self.arView.entity(at: location) else {return}

    if hitEntity.name == "pointCharge" {
        trackedEntity = hitEntity

        // Deselect the previous selected force by deselecting them all first
        // ...

        // Find the tapped PointCharge Object
        let tappedPointCharge: PointChargeClass =
            self.topology.pointCharges.first { (p) -> Bool in
                p.entity.id == trackedEntity.id
        }!

        // ...
    }
    /// Hide all forces
    self.topology.toggleAllForces(show: false)

    /// Hide all Coulomb Labels
    self.topology.toggleCoulombLabels(show: false)
}
```

Μόλις ο χρήστης σταματήσει να σέρνει το επιλεγμένο φορτίο, πρέπει να επανεμφανιστούν οι απαραίτητες δυνάμεις που περιγράψαμε προηγουμένως, δηλαδή όσες αφορούν το επιλεγμένο φορτίο. Με τη μέθοδο *showForces(for pointChargeObj: PointChargeClass)* είμαστε σε θέση να εμφανίσουμε τις δυνάμεις που αφορούν το αντικείμενο *PointChargeClass* που περνάμε ως παράμετρο, το οποίο αποτελεί το επιλεγμένο. Καλούμε τη μέθοδο αυτή μέσα από τη μέθοδο *touchesEnded()*.

```
// Show forces relative only to the pointChargeObj
func showForces(for pointChargeObj: PointChargeClass) {
    self.toggleAllForces(show: false)
    pointChargeObj.netForce?.arrowEntity.isEnabled = true
    pointChargeObj.netForce?.forces.forEach{ force in
        force.arrowEntity.isEnabled = true
```

```
    }
    pointChargeObj.forcesOnOthers.forEach{ force in
        force.arrowEntity.isEnabled = true
    }
}

override func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?) {

    // ...

    /// Enable Forces again
    self.topology.showForces(for: selectedPointChargeObj)

    /// Show all Coulomb Labels
    self.topology.toggleCoulombLabels(show: true)

    /// Update all forces magnitude,directions and all Distance Indicators
    self.topology.updateForces(for: selectedPointChargeObj)
    self.topology.updateDistanceIndicators(for: selectedPointChargeObj)

    // ...
}
```

4. Συμπεράσματα και μελλοντικές επεκτάσεις

4.1 Συμπεράσματα και παρατηρήσεις

Με την ολοκλήρωσης της υλοποίησης της εφαρμογής AR Coulomb καταλήξαμε σε ορισμένα συμπεράσματα, όσον αφορά τη δημιουργία μίας εφαρμογής επαυξημένης πραγματικότητας με το εργαλείο ARKit αλλά και την μετέπειτα χρήση της από τον τελικό χρήστη. Συγκεκριμένα, αναφέρεται κατά πόσο τέτοιου είδους εφαρμογές περιορίζονται από παράγοντες όπως το υλισμικό αλλά και το γύρω περιβάλλον. Αναλυτικότερα:

- Όπως είναι λογικό, συσκευές με καλύτερο υλισμικό είναι σε θέση να υποστηρίξουν μεγαλύτερη ποικιλία εφαρμογών επαυξημένης πραγματικότητας, ειδικότερα όσες έχουν καινούργιους επεξεργαστές καθώς διαθέτουν τις νεότερες εκδόσεις λογισμικού iOS. Παλαιότερες εκδόσεις λογισμικού δεν υποστηρίζουν όλες τις λειτουργικότητες του εργαλείου ARKit και των νεότερων εκδόσεών του. Επομένως, η εμπειρία του χρήστη στην ίδια εφαρμογή μπορεί να διαφέρει πολύ, καθώς θα είναι πολύ ομαλή και ρεαλιστική σε μία καινούργια συσκευή, ενώ σε παλαιότερες θα παρουσιαστούν δυσκολίες σε ορισμένες λειτουργικότητες όπως η γρήγορη ταυτοποίηση προσώπου, η έμπιστη ταυτοποίηση επιφανειών κ.α.

- Ένας ακόμα περιορισμός προκύπτει από την ποικιλομορφία του περιβάλλοντος στο οποίο μπορεί να βρεθεί ο χρήστης ανά πάσα στιγμή. Το εργαλείο ARKit βασίζεται στον επαρκή φωτισμό και την κατάλληλη μορφοποίηση του χώρου ώστε να παρέχει στο χρήστη μία ευχάριστη και αξιόπιστη εμπειρία επαυξημένης πραγματικότητας. Είναι απαραίτητο για τον σχεδιαστή της εφαρμογής να παροτρύνει το χρήστη να βρίσκεται σε χώρους με τις συνθήκες αυτές.

- Η επεξεργαστική ισχύς που απαιτείται για την ομαλή λειτουργία μίας εφαρμογής επαυξημένης πραγματικότητας καθιστά την κατανάλωση των επεξεργαστικών πόρων και της μπαταρίας μόνιμο πρόβλημα για το χρήστη. Αυτομάτως, η πολύωρη χρήση της εφαρμογής κρίνεται απαγορευτική καθώς είτε πέφτουν πολύ τα επίπεδα της μπαταρίας είτε αρχίζει η δυσλειτουργία της συσκευής και η εμπειρία του χρήστη από ομαλή γίνεται μη ευχάριστη.

- Τα τρισδιάστατα ψηφιακά μοντέλα μπορούν να συνυπάρχουν σε ικανοποιητικό πληθυσμό σε μία σκηνή κατά τη διάρκεια λειτουργίας της εφαρμογής, αρκεί οι διαστάσεις και τα χαρακτηριστικά όπως υφή και χρώμα να παραμένουν ίδια. Με άλλα λόγια, η απόδοση της εφαρμογής παραμένει σε υψηλά επίπεδα, αρκεί τα μοντέλα να μην χρειαστεί να αποδοθούν πάνω από μία φορά σε ένα μικρό χρονικό διάστημα. Στην εφαρμογή μας υλοποιήσαμε μία λύση για τη συνεχή ενημέρωση των μοντέλων και επομένως την απόδοσή τους πολλές φορές σε μικρό χρονικό διάστημα. Η συνεχής προσπάθεια για επανειλημμένη απόδοση όμως δημιουργεί μεγάλο επεξεργαστικό όγκο και βάζει

περιορισμούς στη χρήση της εφαρμογής, όσον αφορά τον αριθμό των μοντέλων που μπορούν να συνυπάρχουν και να ενημερώνονται ταυτόχρονα, αλλά και ως προς τον αριθμό των καρέ ανά δευτερόλεπτο που μπορεί να παράγει η εφαρμογή. Επομένως, είναι προτιμότερο σε μία εμπειρία επαυξημένης πραγματικότητας να περιορίζεται η ανάγκη για συνεχείς αλλαγές στα συστατικά των μοντέλων κατά τη διάρκεια λειτουργίας.

4.2 Μελλοντικές επεκτάσεις

Δεδομένων των παρατηρήσεων και συμπερασμάτων που διατυπώθηκαν προηγουμένως, είμαστε σε θέση να αναφέρουμε ορισμένες πιθανές μελλοντικές υλοποιήσεις της εφαρμογής AR Coulomb καθώς και χρήσεις του εργαλείου ARKit. Όσον αφορά την εφαρμογή AR Coulomb, ορισμένα επόμενα βήματα θα μπορούσαν να είναι τα εξής:

- Η χρήση ενός εξωτερικού εξυπηρετητή στο διαδίκτυο (Cloud Service) για την αποθήκευση τοπολογιών και διαφορετικών ασκήσεων ώστε να μην εναποτίθεται περαιτέρω φορτίο στη μνήμη της συσκευής.
- Η δημιουργία και διατήρηση προσωπικών λογαριασμών των χρηστών στην εφαρμογή ώστε να είναι σε θέση να ανταλλάσουν δεδομένα όπως τοπολογίες και ασκήσεις.
- Η ενεργοποίηση μίας ομαδικής εμπειρίας μεταξύ των χρηστών, όπου η τοπολογία στη σκηνή θα είναι ορατή από όσους χρήστες χρησιμοποιούν την εφαρμογή τους ταυτόχρονα, κάτι το οποίο θα βοηθούσε στην ομαδική μελέτη των ασκήσεων με τη βοήθεια της επαυξημένης πραγματικότητας.

Στο κομμάτι του σχεδιασμού και της ανάπτυξης μίας εφαρμογής επαυξημένης πραγματικότητας, η διεπαφή του ARKit σε συνδυασμό με άλλα εργαλεία όπως το RealityKit καθιστούν τη πορεία συγγραφής της μία διαδικασία με συγκεκριμένα βήματα και ροή, κάτι το οποίο αποτελεί πλεονέκτημα για τον σχεδιαστή του έργου. Παρόλα αυτά, εναποθέτουμε παρακάτω ορισμένα σημεία τα οποία επιδέχονται βελτίωσης ή θα μπορούσαν να αποτελούν μελλοντικές υλοποιήσεις του εργαλείου.

- Με τη συνεχή βελτίωση και ανάπτυξη των αρχιτεκτονικών των επεξεργαστών, είναι εφικτό στο επόμενο διάστημα να γίνεται καλύτερη διαχείριση της κατανάλωσης ισχύος και επεξεργαστικών πόρων από τις εφαρμογές του ARKit. Ο φόρτος που δέχεται η συσκευή για την αναπαράσταση τρισδιάστατων μοντέλων θέτει ορισμένα όρια στην εμπειρία του χρήστη, η οποία μπορεί να βελτιωθεί με τη χρήση καλύτερου υλισμικού αλλά και αλγορίθμων τεχνητής νοημοσύνης για την ανίχνευση χαρακτηριστικών του πραγματικού χώρου.

- Με προηγμένους αλγόριθμους μηχανικής εκμάθησης είναι δυνατόν να υπάρχει ακριβέστερη ταυτοποίηση αντικειμένων στο περιβάλλον του χρήστη χωρίς να απαιτείται από τον ίδιο ιδιαίτερη προσοχή στην κίνηση ή εστίαση της συσκευής του. Ένα βήμα σε αυτή την κατεύθυνση θα ελευθέρωνε ακόμα περισσότερο το χρήστη στην κίνησή του και τη συμπεριφορά του κατά τη διάρκεια χρήσης της εφαρμογής.

- Τη δεδομένη στιγμή, ειδικά σε περιπτώσεις όπου επιχειρούμε μία υλοποίηση συνεχούς ενημέρωσης των μοντέλων, όπως η μεταβολή των διαστάσεών τους, απαιτούμε η διαχείριση διαγραμμένων μοντέλων και η δημιουργία νέων να επαναλαμβάνεται πολλές φορές το δευτερόλεπτο. Αυτό επιβαρύνει κατά πολύ την ομαλή λειτουργία της συσκευής και χαλάει την εμπειρία του χρήστη. Η δυνατότητα πρόσβασης στον προγραμματιστή στα συστατικά ενός τρισδιάστατου μοντέλου με σκοπό την επεξεργασία τους, καθιστά την αλλαγή των χαρακτηριστικών του λιγότερο απαιτητική, καθώς μέχρι στιγμής τα συστατικά είναι διαθέσιμα μόνο για ανάγνωση των δεδομένων τους. Με την πρόσβαση αυτή δε θα απαιτείται η δημιουργία του μοντέλου εξαρχής και κατ επέκταση η διαχείριση διαγραμμένων προηγούμενων εκδόσεων του, απελευθερώνοντας επεξεργαστικούς πόρους και κύκλους διεργασιών από το λογισμικό της συσκευής.

- Τέλος, για την πρακτικότερη διαχείριση των πόρων και κατ επέκταση για την παρουσίαση μίας πιο αρεστής τελικής εμπειρίας στο χρήστη, το ARKit θα μπορούσε να δέχεται τα δεδομένα της συσκευής κατά τη λειτουργία μίας εφαρμογής και να προσαρμόζει τα μοντέλα ώστε να μη γίνεται υπερβολική κατανάλωση επεξεργαστικών πόρων και μπαταρίας. Διαβάζοντας τα τρέχοντα δεδομένα κατάστασης της συσκευής (όπως το ποσοστό λειτουργίας της επεξεργαστικής μονάδας), θα ήταν λογικό το ARKit να αφαιρεί αυτόματα ορισμένα χαρακτηριστικά από τα μοντέλα, μειώνοντας έτσι τις λεπτομέρειές τους όπως για παράδειγμα η υφή, οι αντανακλάσεις ή οι σκιές, διατηρώντας παράλληλα την κύρια ταυτότητα του μοντέλου ώστε να είναι αντιληπτό από το χρήστη τι είναι αυτό που βλέπει στην κάμερά του. Με αυτό τον τρόπο, θα ήταν εφικτή μία ευχάριστη εμπειρία με πολλά καρέ το δευτερόλεπτο και για χρήστες με παλαιότερες συσκευές.

Παράρτημα Α: Αναφορά σε μεταβλητές, μεθόδους και κλάσεις της εφαρμογής

Α.1 Καθολικές μεταβλητές και συναρτήσεις

<table>
<tr><td>Ιδιότητες</td><td>
<table>
<tr><td>Σταθερές:</td></tr>
<tr><td>(Notification Keys)
cbNotificationKey: String
topoNotificationKey: String
removalNotificationKey: String
dismissalNotificationKey: String
photoTakenNotificationKey: String
newSelectedPointChargeNotificationKey: String
newSelectedForceValueNotificationKey: String

ZOOM_IN_5_4: Float
ZOOM_OUT_4_5: Float
Ke: Float

defaultTopologies: [DefaultTopologyModel]</td></tr>
<tr><td>Μεταβλητές:</td></tr>
<tr><td>selectedForceObj: Force
previouslySelectedForceAngleFloatValue: Float

Εξαρτημένες
selectedForceAngleFloatValue: Float
selectedPointChargeObj: PointChargeObj
selectedForceValue: String</td></tr>
</table>
</td></tr>
<tr><td>Μέθοδοι</td><td></td></tr>
</table>

A.2 Καθολικές Επεκτάσεις (Extensions)

BinaryInteger

Τύπος	Επέκταση
Μέθοδος	**degreesToRadians**<F: FloatingPoint> → F

FloatingPoint

Τύπος	Επέκταση
Μεταβλητή	(Computed) **degreesToRadians**: FloatingPoint
Μεταβλητή	(Computed) **radiansToDegrees**: FloatingPoint

UIView

Τύπος	Επέκταση
Μέθοδος	**snapshot**(of rect: CGRect, afterScreenUpdates: Bool) → UIImage

ARCamera.TrackingState

Τύπος	Επέκταση
Μεταβλητή	(Computed) **presentationString**: String
Μεταβλητή	(Computed) **recommendation**: String

UIBezierPath

Τύπος	Επέκταση

Μέθοδος Κλάσης	**arrow**(from start: CGPoint, to end: CGPoint, tailWidth: CGFloat, headWidth: CGFloat, headLength: CGFloat) → UIBezierPath

Α.3 Περιγραφή Κλάσεων

Α.3.1 Κύρια κλάση AppDelegate: UIResponder, UIApplicationDelegate

Μεταβλητές	Public**:** **window**: UIWindow
Μεθόδοι	Public: **application(**_ application: UIApplication**,** didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) → Bool **applicationWillResignActive**(_ application: UIApplication) **applicationDidEnterBackground**(_ application: UIApplication) **applicationWillEnterBackground**(_ application: UIApplication) **applicationDidBecomeActive**(_ application: UIApplication) **applicationWillTerminate**(_ application: UIApplication)

Α.3.2 Κλάσεις διαχείρισης αρχείων της εφαρμογής

NSPointCharge: NSManagedObject, Identifiable

Μεταβλητές	Public: @NSManaged public **value**: Float @NSManaged public **multiplier**: Float @NSManaged public **posX**: Float @NSManaged public **posY**: Float @NSManaged public **posZ**: Float @NSManaged public **topology**: NSTopology
Μεθόδοι	

	Public: @nonobjc class fetchRequest() → NSFetchRequest<NSPointCharge>

NSTopology: NSManagedObject, Identifiable

Μεταβλητές	Public: @NSManaged public **descr**: String @NSManaged public **image**: Data @NSManaged public **name**: String @NSManaged public **pointCharges**: NSOrderedSet
Μεθόδοι	Public: @nonobjc class fetchRequest() → NSFetchRequest<NSTopology> @objc(insertObject:inPointChargesAtIndex:) @NSManaged **insertIntoPointCharges**(_ value: NSPointCharge, at idx: Int) @objc(removeObjectFromPointChargesAtIndex:) @NSManaged **removeFromPointCharges**(at idx: Int) @objc(insertPointCharges:atIndexes:) @NSManaged **insertIntoPointCharges**(_ values: [NSPointCharge], at indexes: NSIndexSet) @objc(removePointChargesAtIndexes:) @NSManaged **removeFromPointCharges**(at indexes: NSIndexSet) @objc(replaceObjectInPointChargesAtIndex:withObject:) @NSManaged **replacePointCharges**(at idx: Int, with value: NSPointCharge) @objc(replacePointChargesAtIndexes:withPointCharges:) @NSManaged **replacePointCharges**(at indexes: NSIndexSet, with values: [NSPointCharge]) @objc(addPointChargesObject:) @NSManaged **addToPointCharges**(_ value: NSPointCharge) @objc(removePointChargesObject:)

	@NSManaged **removeFromPointCharges**(_ value: NSPointCharge) @objc(addPointCharges:) @NSManaged **addToPointCharges**(_ values: NSOrderedSet) @objc(removePointCharges:) @NSManaged **removeFromPointCharges**(_ values: NSOrderedSet)

PointChargeModel

Μεταβλητές	Public: **position**: SIMD3<Float> **value**: Float **multiplier**: Float
Μεθόδοι	Public: **getPosition()** → SIMD3<Float> **setPosition(**position: SIMD3<Float>**)** **getValue()** → Float **setValue(**value: Float**)** **getMultiplier()** → Float **setMultiplier(**multiplier: Float**)**

TopologyModel

Μεταβλητές	Public: public **id**: ObjectIdentifier public **pointCharges**: [PointChargeModel]

	public **image**: UIImage public **name**: String public **descr**: String
Μεθόδοι	Public: **addPointChargeModel**(model: PointChargeModel) **getPointCharges**() → [PointChargeModel] **setImage**(image: UIImage) **getImage**() → UIImage **setName**(name: String) **getName**() → String **setDescription**(descr: String) **getDescription**() → String

DefaultTopologyModel

Μεταβλητές	Public: **positions**: [SIMD3<Float>] **image**: Data **name**: String **descr**: String
Μεθόδοι	

A.3.3 Οθόνες (View Controllers)

Κύρια οθόνη

ViewController: UIViewController, UIGestureRecognizerDelegate

<table>
<tr><td>Μεταβλητές</td><td>
<table>
<tr><td>Public:</td></tr>
<tr><td>@IBOutlet arView: ARView

(UI View elements)
shutterView: UIView
angleOverview: AnglesOverviewView
angleLabel: UILabel
messagePanel: UIView
messageLabel: UIPaddingLabel
restartExperienceButton: UIButton
stackView: UIStackView
cancelCaptureButton: UIButton
captureButton: UIButton
saveButton: UIButton
newTopoButton: UIButton
addButton: UIButton
coachingOverlay: ARCoachingOverlayView

isRestartAvailable: Bool
menuDict: [String: [UIButton]]
status: Status

selectedARPlaneAnchor: ARPlaneAnchor
currentARPlaneAnchor: ARPlaneAnchor

topology: Topology</td></tr>
</table>
</td></tr>
<tr><td>Μεθόδοι</td><td>
<table>
<tr><td>Public:</td></tr>
<tr><td>(Button Functions)
@objc restartExperience(sender: UIButton)
@objc goBack(sender: UIButton)
@objc captureSnapshot(sender: UIButton)</td></tr>
</table>
</td></tr>
</table>

	@objc **openCaptureMenu**(sender: UIButton) @objc **confirmSeagueToTopoMenu**(sender: UIButton) @objc **performAddition**(sender: UIButton) override **viewDidAppear**(_ animated: Bool) override **prepare**(for segue: UIStoryboardSegue, sender: Any?) **performSegueToTopoMenu**() **performSegueToCapturedImageVC**(image: UIImage) **restartExperience**() **pointChargeInteract**(zoom: Float, showLabel: Bool)
	Private:
	setupARView() **setupTapGestureRecognizer**() **setupLongPressRecognizer**() **resetTracking**()

Επέκταση της κύριας οθόνης για τα στοιχεία του User Interface (UI)

extension ViewController

Μεθόδοι	Public:
	configureAngleOverview() **configureAngleLabel**() **configureMessagePanel**() **configureRestartExperienceButton**() **configureShutterView**() **configureStackView**() **configureButton**() **toggleStackView**(hide: Bool, animated: Bool, animationDuration: Float) **transitionStackViewMenu**(to menu: String) **toggleViewsOnSnapshot**(hide: Bool) **toggleAllSubviews**(of view: UIView, hide: Bool)

Επέκταση της κύριας οθόνης για την υλοποίηση χειρονομιών (Gestures)

extension ViewController

Μεθόδοι	
	Public:
	@obj **handleTap**(recognizer: UITapGestureRecognizer) @obj **handleLongPress**(recognizer: UILongPressGestureRecognizer) override **touchesBegan**(_ touches: Set<UITouch>, with event: UIEvent?) override **touchesMoved**(_ touches: Set<UITouch>, with event: UIEvent?) override **touchesEnded**(_ touches: Set<UITouch>, with event: UIEvent?) **enableRecognizers**(withName name: String) **disableRecognizers**(withName name: String)

Επέκταση της κύριας οθόνης για την υλοποίηση ειδοποιήσεων (Notification Observers)

extension ViewController

Μεθόδοι	
	Public:
	setupObserverMenuDismissal() @obj **recoverAddButton**(notification: Notification) **setupObserverNewTopo**() @obj **addTopology**(notification: Notification) **setupObserverPointChargeDeletion**() @obj **removePointCharge**(notification: Notification) **setupObserverCoulomb**() @obj **updateCoulombValue**(notification: Notification) **setupObserverCapturedImage**() @obj **saveTopologyToCoreData**(notification: Notification) **setupObserverNewSelectedPointChargeObject** @obj **updateAnglesOverviewView**(notification: Notification) **setupObserverNewSelectedForceValue**() @obj **updateAnglesLabel**(notification: Notification)

Επέκταση της κύριας οθόνης για την υλοποίηση του καθοδηγητή

extension ViewController: ARCoachingOverlayViewDelegate

Μεθόδοι	
	Public:
	coachingOverlayViewWillActivate(_ coachingOverlayView: ARCoachingOverlayView) **coachingOverlayViewDidDeactivate**(_ coachingOverlayView: ARCoachingOverlayView) **coachingOverlayViewDidRequestSessionReset**(_ coachingOverlayView: ARCoachingOverlayView) **setupCoachingOverlay**() **setAutoActivation**(auto: Bool) **setGoal**()

Επέκταση της κύριας οθόνης για τη χρήση του AR Session Delegate

extension ViewController: ARSessionDelegate

Μεθόδοι	
	Public:
	session(_ session: ARSession, cameraDidChangeTrackingState camera: ARCamera) **session**(_ session: ARSession, didUpdate frame: ARFrame)

Οθόνη επεξεργασίας φορτίου

CoulombMenuVC: UIViewController

Μεταβλητές	
	Public:
	coulombMenuView: UIView **vStackView**: UIStackView

	hStackView: UIStackView **tabView**: UIView **text**: UILabel **sliderView**: UIView **slider**: UISlider **btns**: [UIButton] **trashButton**: UIButton **step**: Float **initialCoulombValue**: Float **viewController**: ViewController
Μεθόδοι	Public: override **viewDidLoad**() **configureTabView**() **configureStackView_V**() **configureTextLabel**(coulombValue: Float) **configureStackView_H**() **configureSlider**(coulombValue: Float) **configureButtons**() **configureTrashButton**() @obj **sliderButtonAction**(sender: UIButton) @obj **performDeletion**(sender: UIButton) @obj **sliderValueDidChange**(_ sender: UISlider!) override **touchesBegan**(_ touches: Set<UITouch>, with event: UIEvent?) Private: **updateSlider**(constant: Float) **textUpdate**(sliderValue: Float) **notifyObserver**(withKey key: String, value: Float) **notifyObserver**(withKey key: String)

Οθόνη για το μενού επιλογής τοπολογίας

BottomTopoMenuVC: UIViewController

Μεταβλητές	Public: --- @IBOutlet **bottomTopoMenuView**: UIView! **selectedTopo**: TopologyModel **stackView**: UIStackView **scrollView**:UIScrollView **previewView**: UIView **previewImageView**: UIImageView **previewDetailsView**: UIView **previewTitleTextView**: UITextView **previewDetailsTextView**: UITextView **selectTopoButton**: UIButton **deleteTopoButton**: UIButton **dismissMenuButton**: UIButton
Μεθόδοι	Public: --- override **viewDidLoad**() **configureScrollView**() **configureStackView**() **configurePreviewView**() **configurePreviewImageView**() **configureSelectTopoButton**() **configureDissmissMenuButton**() **configureDeleteTopoButton**() **configurePreviewDetailsView()** **configurePreviewTitleTextView**() **configurePreviewDetailsTextView**() **updatePreview**() @obj **buttonLoadTopoAction**(sender: UIButton!) @obj **buttonDeleteTopoAction**(sender: UIButton!) @obj **buttonSelectTopoAction**(sender: UIButton!) @obj **dissmissMenu**(sender: UIButton)

	Private: **reloadTopoView**() **selectTopo(**button: UIButton) **notifyObserver**(withKey key: String)

Οθόνη στιγμιότυπου τοπολογίας

CapturedImageVC: UIViewController

Μεταβλητές	Public: @IBOutlet **capturedImageView**: UIView **imageView**: UIImageView **stackView**: UIStackView **cancelButton**: UIButton **saveButton**: UIButton **image**: UIImage
Μεθόδοι	Public: @obj **cancelSaveTopology**(sender: UIButton) @obj **saveTopology**(sender: UIButton) **topoSavedFeedback**(valid: Bool, message: String) @obj **imageFeedback**(_ image: UIImage, didFinishSavingWithError error: NSError?, contextInfo: UnsafeRawPointer) override **viewDidLoad**() **configureImageView**() **configureStackView**() **configureButton**(for btn: UIButton, title: String)

A.3.4 Κλάσεις στοιχείων της αναπαράστασης

Η κλάση για την αναπαράσταση τοπολογιών στη σκηνή

Topology

Μεταβλητές	Public: **viewController**: ViewController **topoAnchorEntity**: AnchorEntity **selectedPositions**: [SIMD3<Float>] **pointCharges**: [PointChargeClass] **netForces**: [NetForce] **forces**: [Force] **distanceIndicators**: [DistanceIndicator] **labels**: [Entity] **pointChargeEntityTemplate**: Entity
Μεθόδοι	Public: **pinToScene**(viewController: ViewController, topoAnchor: AnchorEntity) **placeTopology**(topoModel: TopologyModel) **clearTopology**() **toggleTopology**(show: Bool)

Επέκταση της κλάσης Topology για τη διαχείριση των δυνάμεων

extension Topology

Μεθόδοι	Public: **addAllForces**() **showForces**(for pointChargeObj: PointChargeClass)

	updateForces() **updateForces**(for pointCharge: PointChargeClass) **reloadAllForces**() **clearAllForces**() **toggleAllForces**()

Επέκταση της κλάσης Topology για τη διαχείριση των φορτίων

extension Topology

Μεθόδοι	
	Public:
	addPointChargeWithRandomPosition() **add**(pointCharge p: PointChargeModel) **addPointCharge**(to pos: SIMD3<Float>) **removeAllPointCharges**() **removePointCharge**() **togglePointCharges**(show: Bool) **toggleCoulombLabels**(show: Bool)
	Private:
	randomPosition() → SIMD3<Float>

Επέκταση της κλάσης Topology για τη διαχείριση των δεικτών απόστασης

extension Topology

Μεθόδοι	
	Public:
	addDistanceIndicators() **showDistanceIndicators**(for pointChargeObj: PointChargeClass) **updateDistanceIndicators**() **updateDistanceIndicators**(for pointCharge: PointChargeClass) **toggleDistanceIndicators**(show: Bool) **clearDistanceIndicators**()

	reloadDistanceIndicators()

Η κλάση για την αναπαράσταση των φορτίων

PointChargeClass

Μεταβλητές	Public: static **count**: Int static **pointChargeRadius**: Float static **multiplier**: Float **id**: Int **entity**: Entity **topology**: Topology **label**: Entity **value**: Float **netForce**: NetForce **forcesOnOthers**: [SingleForce] **distanceIndicators**: [DistanceIndicator]
Μεθόδοι	Public: Static **==** (lhs: PointChargeClass, rhs: PointChargeClass) → Bool **getPositionX**() → Float **getPositionY**() → Float **getPositionZ**() → Float

Η κλάση για την αναπαράσταση των οντοτήτων των φορτίων

PointChargeEntity: Entity, HasCollision

Η κλάση PointChargeEntity δεν περιέχει ιδιότητες ή μεθόδους.

Η κλάση για την γενικότερη αναπαράσταση των δυνάμεων

Force

Μεταβλητές	Public: static **total**: Int static **metersPerNewton**: Float **forceId**: Int **topology**: Topology **magnitude**: Float **previousMagnitude**: Float **angle**: Float **pivotEntity**: Entity **arrowEntity**: Entity **label**: Entity **selected**: Bool **type**: ForceType (Computed Properties) **X_Force_Component**: Float **Y_Force_Component**: Float **length**: Float **color**: UIColor
Μεθόδοι	Public: static **createArrowModel**(on pointChargeObj: PointChargeClass, **magnitude**: Float, name: String) ⟶ Entity static **createSingleForce**(from otherPointChargeObj: PointChargeClass, to pointChargeObj: PointChargeClass) ⟶ SingleForce static **createNetForce**(for pointChargeObj: PointChargeClass) ⟶ NetForce **updateArrowModel**() **updateForceMagnitude**() **updateForceAngle**() **updateForceArrowOrientation**() Private:

	calculateForceComponent(x_component_flag: Bool) ⟶ Float updateLabel()

Η κλάση για την αναπαράσταση των συνιστωσών δυνάμεων

SingleForce: Force

Μεταβλητές	Public: static **singleForcesTotal**: Int **singleForceId**: Int **sourcePointCharge**: PointChargeClass **targetPointCharge**: PointChargeClass (Computed) **attraction**: Bool
Μεθόδοι	Public: static **createForce**(from otherPointChargeObj: PointChargeClass, to pointChargeObj: PointChargeClass) ⟶ SingleForce **updateForce**() override **updateForceMagnitude**() override **updateForceAngle**() override **updateForceArrowOrientation**() Private: **coulombsLaw**() ⟶ Float **twoPointsDistance**(x1: Float, x2: Float, y1: Float, y2: Float) ⟶ Float

Η κλάση για την αναπαράσταση των συνισταμένων δυνάμεων

NetForce: Force

Μεταβλητές	Public: static **netForcesTotal**: Int **netForceId**: Int **pointChargeObj**: PointChargeClass **forces**: [SingleForce]
Μεθόδοι	Public: static **createForce**(for pointChargeObj: PointChargeClass) ⟶ NetForce **updateForce**() **calculateNetForce**() override **updateForceArrowOrientation**() override **updateForceAngle**() override **updateForceMagnitude**() Private: **netForceMagnitude**(fx: Float, fy: Float) ⟶ Float **netForceAnle**(fx: Float, fy: Float) ⟶ Float

Η κλάση για την αναπαράσταση των δεικτών απόστασης

DistanceIndicator

Μεταβλητές	Public: static **count**: Int

	static **lineMaterial**: UnlitMaterial **id**: Int **sourcePointCharge**: PointChargeClass **targetPointCharge**: PointChargeClass **entity**: Entity **sourceLine**: Entity **targetLine**: Entity **label**: Entity **topology**: Topology **previousDistance**: Float (Computed) **distance**: Float
Μεθόδοι	Public: **updateIndicator**() **toggle**(show: Bool) Private: **updatePosition**() **updateOrientation**() **updateLabel**() **updateLines**() **twoPointsDistance**(x1: Float, x2: Float, y1: Float, y2: Float) → Float

A.3.5 Κλάσεις όψεων (Views)

AnglesOverviewView: UIView

Μεταβλητές	Public: **FORCE_ARROW_TAIL_LENGTH**: Float

	NETFORCE_ARROW_TAIL_LENGTH: Float **ARROW_TAIL_WIDTH**: Float **HEAD_LENGTH**: Float **HEAD_WIDTH**: Float **ARC_RADIUS**: Float **ARC_COLOR**: UIColor **NETFORCE_COLOR**: UIColor **SELECTED_FORCE_COLOR**: UIColor
Μεθόδοι	Public: override **draw**(_ rect: CGRect) Private: **drawForces**(center: CGPoint) **drawForce**(center: CGPoint, angle: Float, forceType: ForceType, color: UIColor, selected: Bool) **drawAngleArc**(center: CGPoint, angle: Float)

@IBDesignable UIPaddingLabel: UILabel

Μεταβλητές	Public: @IBInspectable **topInset**: CGFloat @IBInspectable **bottomInset**: CGFloat @IBInspectable **leftInset**: CGFloat @IBInspectable **rightInset**: CGFloat (Computed) override **intrinsicContentSize**: CGSize
Μεθόδοι	Public**:**

	override **drawText**(in rect: CGRect)

A.3.6 Βοηθητικές κλάσεις

PersistenceService

Μεταβλητές	Public: static **context**: NSManagedObjectContext (Computed) static **persistentContainer**: NSPersistentContainer
Μεθόδοι	Public: static **saveContext**()

Status

Μεταβλητές	Public: enum **MessageType**: case **trackingStateEscalation** case **planeEstimation**

	case **contentPlacement** case **focusSquare** static **all**: [MessageType] **viewController**: ViewController Private: **displayDuration**: TimeInterval **messageHidetimer**: Timer **timers**: [MessageType: Timer]
Μεθόδοι	Public: **showMessage**(_ text: String, autoHide: Bool) **scheduleMessage**(_ text: String, inSeconds seconds: TimeInterval, messageType: MessageType) **cancelScheduledMessage**(for messageType: MessageType) **cancelAllScheduledMessages**() **showTrackingQualityInfo**(for trackingState: ARCamera.TrackingState, autoHide: Bool) **escalateFeedback**(for trackingState: ARCamera.TrackingState, inSeconds seconds: TimeInterval) Private: **setMessageHidden**(_ hide: Bool, animated: Bool)

TopologyStore

Μεταβλητές	Public: static **sharedInstance**: TopologyStore **savedTopologies**: [TopologyModel]

Μεθόδοι	Public: **totalTopologies**() **saveDefaultTopologiesToCoreData**() **deleteDefaultTopologiesFromCoreData**() **deleteTopologyFromCoreData**(topology: NSTopology) **deleteSavedTopologyFromCoreData**(topology: TopologyModel) **reloadSavedTopologies**() Private: **eraseTopology**(topology: TopologyModel) **eraseTopologies**() **loadSavedTopologies**()

Alert

Μεταβλητές	
Μεθόδοι	Public: static **showDeletionConfirmation**(on vc: ViewController) static **showPointChargesLimitReached**(on vc: ViewController) static **showSeagueToTopoMenuConfirmation**(on vc: ViewController) Private: static **showTripleConfirmationAlert**(on vc: ViewController, title: String, message: String) static **showConfirmationAlert**(on vc: ViewController, title: String, message: String) static **showBasicAlert**(on vc: ViewController, title: String, message: String)

EntityStore

Μεταβλητές	Public: **shared**: EntityStore **textMaterial**: SimpleMaterial **textOcclussionMaterial**: OcclusionMaterial
Μεθόδοι	Public: **load_PointChargeEntity**() → Entity **update_PointChargeModel**(on pointChargeEntity: Entity, radius: Float, color: UIColor) **update_PointChargeEntity**(pointChargeEntity: Entity, radius: Float, color: UIColor) **load_TextEntity**(on entity: Entity, inside topology: Topology, name: String, position: SIMD3<Float>, occlusion: Bool) → Entity **update_TextEntity**(textEntity: Entity, stringValue: String, fontSize: Float) **update_AllTextOrientation**(in topology: Topology) **load_ArrowBody_Entity**(pointEntity: Entity, magnitude: Float) → Entity **update_ArrowBody_Entity**(arrowBodyEntity: Entity, magnitude: Float) **load_ArrowHead**(on arrowEntity: Entity, magnitude: Float) **update_ArrowHead**(on arrowEntity: Entity, magnitude: Float) **load_PlacementIndicator**(side: Float, imageAssetPath: String) → AnchorEntity **toggle_PlacementIndicator**(anchor: AnchorEntity, show: Bool) **update_PlacementIndicator_ModelComponent**(on piAnchor: AnchorEntity, transform: Transform) **load_DistanceIndicator**() → Entity **load_DistanceLine_Model**(length: Float) → ModelComponent **update_DistanceLine_Length**(entity: Entity, length: Float) **update_DistanceLines**(sourceLine: Entity, targetLine: Entity, length: Float) Private: **load_PointChargeModel**(radius: Float, color: UIColor) → ModelComponent **load_ArrowBody_ModelComponent**(length: Float) → ModelComponent **update_ArrowBody_Entity_Length**(arrowBodyEntity: Entity, length: Float) **load_PlacementIndicator_ModelComponent**(side: Float, imageAssetPath: String) → ModelComponent

Παράρτημα Β: Κομμάτια κώδικα υλοποίησης

Β.1 Αρχικοποίηση ARSession αντικειμένου

Ο κώδικας που αποσκοπεί στην αρχικοποίηση και εκκίνηση του *ARSession* ήταν προτιμότερο να συμπεριληφθεί σε μία μέθοδο ώστε να είναι εύκολο να επαναληφθεί σε κάθε επανεκκίνηση της εμπειρίας καλώντας απλώς τη μέθοδο. Στη μέθοδο συμπεριλαμβάνονται κι άλλες ενέργειες όπως η προσθήκη ενός τρισδιάστατου μοντέλου δείκτη τοποθέτησης (Placement Indicator), και η αρχικοποίηση και παραμετροποίηση των ανιχνευτών χειρονομιών χρήστη (Gesture Recognizers) που χρησιμοποιούνται.

```
// MARK: - Private Setup startup Functions
private func setupARView() {
    self.arView.automaticallyConfigureSession = false
    let config = ARWorldTrackingConfiguration()
    config.planeDetection = [.horizontal, .vertical]
    config.environmentTexturing = .automatic
    self.arView.session.run(config, options: [.resetTracking, .removeExistingAnchors])

    // Add Placement Indicator Anchor
    self.arView.scene.addAnchor(placementIndicator)

    // First tap gesture recognizer, will be deleted after Topology is added
    self.setupTapGestureRecognizer()

    // Long Press Recognizer to enable parameters interaction with Point Charge (min
press 1 sec)
    self.setupLongPressRecognizer()
}
```

Β.2 Η υλοποίηση μεθόδων για τη χειρονομία απλού πατήματος

Αφού δημιουργήσουμε το *UITapGestureRecognizer* αντικείμενο, του δίνουμε ένα ξεχωριστό όνομα (“First Point Recognizer”) και το προσθέτουμε στην όψη τύπου *ARView* της εφαρμογής μας, την *arView*, καθώς τα πατήματα που θέλουμε να ανιχνεύει θα συμβαίνουν μέσα σε αυτή.

```
private func setupTapGestureRecognizer() {
    let firstPointTapRecognizer = UITapGestureRecognizer(
                                target: self,
                                action: #selector(handleTap(recognizer:))
                                )
    firstPointTapRecognizer.name = "First Point Recognizer"
    self.arView.addGestureRecognizer(firstPointTapRecognizer)
    firstPointTapRecognizer.isEnabled = false
}
```

Για την υλοποίηση της λογικής που επιθυμούμε, περάσαμε ως παράμετρο στο αντικείμενο ανίχνευσης μία μέθοδο, την *handleTap*. Στη μέθοδο αυτή δημιουργείται ένα *AnchorEntity* αντικείμενο, το οποίο αντιπροσωπεύει τη βάση της τοπολογίας που θα τοποθετηθεί στη σκηνή και περιέχει τις πληροφορίες προσανατολισμού της. Πάνω σε αυτό το αντικείμενο θα δημιουργηθεί στη συνέχεια ένα αντικείμενο *Topology*. Επειδή θέλουμε η τοπολογία που θα επιλεχτεί από το χρήστη να τοποθετηθεί στη θέση του δείκτη τοποθέτησης διατηρώντας την περιστροφή και τον προσανατολισμό του, δώσαμε στο αντικείμενο *AnchorEntity* τη μήτρα μετασχηματισμού του αντικειμένου δείκτη τοποθέτησης. Είναι σημαντικό, πριν πραγματοποιηθεί η μετάβαση στο μενού, να ενεργοποιηθεί το αντικείμενο *UITapGestureRecognizer*, καθώς ο στόχος του επιτεύχθηκε και δεν πρέπει να συνεχιστεί η ανίχνευση ομοίων πατημάτων.

```
@objc func handleTap(recognizer: UITapGestureRecognizer) {
    /// Create new Anchor Entity for Topology
    let anchor = AnchorEntity()
    anchor.name = "Topology"
    anchor.transform = placementIndicator.transform

    /// Disable the Placement Indicator, which also stops
    /// updating indicator's transform
    EntityStore.shared.toggle_PlacementIndicator(
                    anchor: placementIndicator,
                    show: false
                    )

    /// Remove gesture recognizer needed for
    /// the First Tap -> Topology AnchorPlacement
    self.disableRecognizers(withName: "First Point Recognizer")
```

```
    /// Create a Topology Instance with the added anchor as topoAnchor
    self.topology = Topology()
    self.topology.pinToScene(viewController: self, topoAnchor: anchor)

    /// Open the bottom Coulomb Topology menu to choose topology
    self.performSegueToTopoMenu()
}
```

Β.3 Η υλοποίηση των μεθόδων για την μετακίνηση των φορτίων

Β.3.1 Πρώτη επαφή (touchesBegan)

```
override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
       // Check if the touch happened on a point of charge entity
       guard let location = touches.first?.location(in: self.arView) else {return}
       guard let hitEntity = self.arView.entity(at: location) else {return}

       if hitEntity.name == "pointCharge" {
              // Set the entity as the tracked one
              trackedEntity = hitEntity

              // Deselect the previous selected force by deselecting them all first
              selectedPointChargeObj.netForce?.selected = false
              selectedPointChargeObj.netForce?.forces.forEach{ f in
                  f.selected = false
              }

              // Find the tapped PointCharge Object
              let tappedPointCharge: PointChargeClass =
              self.topology.pointCharges.first {
                  (p) -> Bool in
                      p.entity.id == trackedEntity.id
              }!

              // If the tappedPointCharge is the selectedPointCharge, select
              // its netForce
              if (tappedPointCharge.entity.id == selectedPointChargeObj.entity.id) {
                  selectedPointChargeObj.netForce?.selected = true
                  selectedForceObj = selectedPointChargeObj.netForce!
                  if let angle = selectedPointChargeObj.netForce?.angle {
                      let selectedForceAngleDegrees =
                          selectedForceAngleFloatValue.radiansToDegrees
                      if abs(angle.radiansToDegrees - selectedForceAngleDegrees) >= 1 {
                          selectedForceAngleFloatValue = angle
                      }
                  }
              } else {
                  // Else, find and select the right force on the selected Point Charge
                  let selectedForce = tappedPointCharge.forcesOnOthers.first {
                      (force) -> Bool in
                          (selectedPointChargeObj.netForce?.forces.contains(where: {
                              (f) -> Bool in
                                  force.forceId == f.forceId
                          }))!
                  }
                  selectedForce?.selected = true
```

```
                selectedForceObj = selectedForce!
                if let angle = selectedForce?.angle {
                    let selectedForceAngleDegrees =
                        selectedForceAngleFloatValue.radiansToDegrees
                    if abs(angle.radiansToDegrees - selectedForceAngleDegrees) >= 1 {
                        selectedForceAngleFloatValue = angle
                    }
                }
            }

            // Toggle all forces off
            self.topology.toggleAllForces(show: false)

            // Toggle all point of charges labels off
            self.topology.toggleCoulombLabels(show: false)
        }
}
```

B.3.2 Το σημείο αφής μετακινείται (touchesMoved)

```
override func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent?) {
    if trackedEntity.name == "pointCharge" {
        self.topology.updateForces(for: selectedPointChargeObj)
        self.topology.updateDistanceIndicators(for: selectedPointChargeObj)
        let angle = selectedForceObj.angle
        let selectedForceAngleDegrees = selectedForceAngleFloatValue.radiansToDegrees
        if abs(angle.radiansToDegrees - selectedForceAngleDegrees ) >= 1 {
            selectedForceAngleFloatValue = angle
        }
    }
}
```

B.3.3 Τερματισμός υπάρχουσας επαφής (touchesEnded)

```
override func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?) {
        /// Enable Forces again
        self.topology.showForces(for: selectedPointChargeObj)
        /// Show all Coulomb Labels
        self.topology.toggleCoulombLabels(show: true)
        /// Update all forces magnitude,directions and all Distance Indicators
        self.topology.updateForces(for: selectedPointChargeObj)
        self.topology.updateDistanceIndicators(for: selectedPointChargeObj)

        // Align the points of charge with axis distance < 0.02 meters
        // First get their (x, z)coordinates
        let x = trackedEntity.position.x
```

```
        let z = trackedEntity.position.z

        // Loop through the scene anchors to find our 'Topology' AnchorEntity
        arView.scene.anchors.forEach{ anchor in
            if anchor.name == "Topology" {
                /// Loop through its children (pointChargeEntities) and check
                /// their (x, z)differences
                anchor.children.forEach{ child in
                    if child.position.x != x && child.position.z != z{
                        if abs(child.position.x - x) < 0.02 {
                            trackedEntity.position.x = child.position.x
                        }
                        if abs(child.position.z - z) < 0.02 {
                            trackedEntity.position.z = child.position.z
                        }
                    }
                }
            }
        }
}
```

Β.4 Υλοποίηση μεθόδων ειδοποιήσεων

Β.4.1 Ειδοποίηση: Επιλογή νέας τοπολογίας από το μενού επιλογής

Μέθοδος ορισμού

```
func setupObserverNewTopo() {
    let notifName = Notification.Name(rawValue: topoNotificationKey)
    NotificationCenter.default.addObserver(
        self,
        selector: #selector(addTopology(notification:)),
        name: notifName,
        object: nil
    )
}
```

Κλήση

```
let notifName = Notification.Name(rawValue: topoNotificationKey)
let valueDict: [String: TopologyModel] = ["updatedValue": selectedTopo!]
NotificationCenter.default.post(name: notifName, object: nil, userInfo: valueDict)
```

Β.4.2 Ειδοποίηση: Επιλογή φορτίου

Μέθοδος ορισμού

```
func setupObserverNewSelectedPointChargeObject() {
    let notifName = Notification.Name(rawValue: newSelectedPointChargeNotificationKey)
    NotificationCenter.default.addObserver(
        self,
        selector: #selector(updateAnglesOverviewView(notification:)),
        name: notifName,
        object: nil
    )
}
```

Κλήση

```
let notifName = Notification.Name(rawValue: newSelectedPointChargeNotificationKey)
NotificationCenter.default.post(name: notifName, object: nil)
```

Β.4.3 Ειδοποίηση: Αλλαγή τιμής φορτίου από το μενού επεξεργασίας φορτίου

Μέθοδος ορισμού

```
func setupObserverCoulomb() {
    let notifName = Notification.Name(rawValue: cbNotificationKey)
    NotificationCenter.default.addObserver(
        self,
        selector: #selector(updateCoulombValue(notification:)),
        name: notifName,
        object: nil
    )
}
```

Κλήση

```
let notifName = Notification.Name(rawValue: key)
let valueDict: [String: Float] = ["updatedValue": value]
NotificationCenter.default.post(name: notifName, object: nil, userInfo: valueDict)
```

Β.4.4 Ειδοποίηση: Διαγραφή φορτίου από το μενού επεξεργασίας φορτίου

Μέθοδος ορισμού

```
func setupObserverPointChargeDeletion() {
    let notifName = Notification.Name(rawValue: removalNotificationKey)
    NotificationCenter.default.addObserver(
        self,
        selector: #selector(removePointCharge(notification: )),
        name: notifName,
        object: nil
    )
}
```

Κλήση

```
let notifName = Notification.Name(rawValue: key)
NotificationCenter.default.post(name: notifName, object: nil, userInfo: nil)
```

Β.4.5 Ειδοποίηση: Κλείσιμο όψης μενού επεξεργασίας φορτίου

Μέθοδος ορισμού

```
func setupObserverMenuDismissal() {
```

```
    let notifName = Notification.Name(rawValue: dismissalNotificationKey)
    NotificationCenter.default.addObserver(
        self,
        selector: #selector(recoverAddButton(notification:)),
        name: notifName,
        object: nil
    )
}
```

Κλήση

```
let notifName = Notification.Name(rawValue: key)
NotificationCenter.default.post(name: notifName, object: nil, userInfo: nil)
```

Β.4.6 Ειδοποίηση: Κλείσιμο όψης στιγμιότυπου τοπολογίας

Μέθοδος ορισμού

```
func setupObserverCapturedImage() {
    let notifName = Notification.Name(rawValue: photoTakenNotificationKey)
    NotificationCenter.default.addObserver(
        self,
        selector: #selector(saveTopologyToCoreData(notification:)),
        name: notifName,
        object: nil
    )
}
```

Κλήση

```
let notifName = Notification.Name(rawValue: photoTakenNotificationKey)
let valueDict: [String: Data] = ["imageData": self.image!.pngData()!]
NotificationCenter.default.post(name: notifName, object: nil, userInfo: valueDict)
```

Β.4.7 Ειδοποίηση: Επιλογή δύναμης

Μέθοδος ορισμού

```
func setupObserverNewSelectedForceValue() {
    let notifName = Notification.Name(rawValue: newSelectedForceValueNotificationKey)
    NotificationCenter.default.addObserver(
        self,
        selector: #selector(updateAnglesLabel(notification:)),
        name: notifName,
        object: nil
    )
```

```
}
```

Κλήση

```
let notifName = Notification.Name(rawValue: newSelectedForceValueNotificationKey)
NotificationCenter.default.post(name: notifName, object: nil)
```

Β.5 Μέθοδοι δημιουργίας και ενημέρωσης τρισδιάστατων μοντέλων

Β.5.1 Μοντέλο δείκτη τοποθέτησης

Δημιουργία (2 μέθοδοι)

```
func load_PlacementIndicator(side: Float = 0.1, imageAssetPath: String =
"Placement_Indicator_LightYellow") -> AnchorEntity {
    let piEntity: AnchorEntity = AnchorEntity()
    piEntity.name = "PlacementIndicator"
    piEntity.isEnabled = false
    let model = load_PlacementIndicator_ModelComponent(
                    side: side,
                    imageAssetPath: imageAssetPath
                )
    piEntity.components.set(model)
    return piEntity
}

private func load_PlacementIndicator_ModelComponent(side: Float, imageAssetPath:
String) -> ModelComponent {
    var material: UnlitMaterial = UnlitMaterial()
    material.baseColor = try! MaterialColorParameter.texture(
                            TextureResource.load(named: imageAssetPath)
                        )
    material.tintColor = UIColor.white.withAlphaComponent(0.9)

    /// Plane for placement indicator
    let mesh: MeshResource = .generatePlane(width: side, depth: side)
    let model: ModelComponent = .init(mesh: mesh, materials: [material])
    return model
}
```

Ενημέρωση (2 μέθοδοι)

```
func update_PlacementIndicator_ModelComponent(on piAnchor: AnchorEntity, side: Float,
imageAssetPath: String) {
    let model = load_PlacementIndicator_ModelComponent(
                    side: side,
                    imageAssetPath: imageAssetPath
                )
    piAnchor.components.set(model)
}

func update_PlacementIndicator_Transform(on piAnchor: AnchorEntity, transform:
Transform) {
```

```
    piAnchor.transform = transform
    piAnchor.position.y += 0.01
}
```

B.5.2 Μοντέλο φορτίου

Δημιουργία (2 μέθοδοι)

```
func load_PointCharge() -> Entity {
    let pointChargeEntity: PointChargeEntity = PointChargeEntity()
    pointChargeEntity.name = "pointCharge"

    self.update_PointCharge_ModelComponent(on: pointChargeEntity)
    return pointChargeEntity
}

private func load_PointCharge_ModelComponent(radius: Float, color: UIColor) ->
ModelComponent {
       let material: SimpleMaterial = {
              var mat = SimpleMaterial()
              mat.metallic = MaterialScalarParameter(floatLiteral: 0.5)
              mat.roughness = MaterialScalarParameter(floatLiteral: 0.5)
              mat.tintColor = color

              return mat
       }()

       // Create a sphere shaped model with the above materials
       let model: ModelComponent = ModelComponent(mesh: .generateSphere(radius:
       radius),  materials: [material])

       return model
}
```

Ενημέρωση (2 μέθοδοι)

```
func update_PointCharge(pointChargeEntity: Entity, radius: Float, color: UIColor) {
    self.update_PointCharge_ModelComponent(
              on: pointChargeEntity,
              radius: radius,
              color: color
         )
}

func update_PointCharge_ModelComponent(on pointChargeEntity: Entity, radius: Float =
PointChargeClass.pointChargeRadius, color: UIColor = UIColor.red) {
    // Add ModelComponent
```

```
    let model = load_PointCharge_ModelComponent(radius: radius, color: color)
    pointChargeEntity.components.set(model)
    // Add CollisionComponent
    /// Make the collision sphere object radius little bigger than the model itself
    pointChargeEntity.components.set(CollisionComponent(shapes:
[.generateSphere(radius: radius * 1.2)]))
}
```

B.5.3 Μοντέλο δείκτη απόστασης

Δημιουργία (3 μέθοδοι)

```
func load_DistanceIndicator() -> Entity {
    let distanceEntity: Entity = Entity()
    distanceEntity.name = "Distance Indicator"

    return distanceEntity
}

func load_DistanceLine(on indicator: Entity) -> Entity {
    let distanceEntity: Entity = Entity()
    distanceEntity.name = "Distance Line"
    indicator.addChild(distanceEntity)

    return distanceEntity
}

func load_DistanceLine_ModelComponent(length: Float) -> ModelComponent {
    let mesh: MeshResource = .generateBox(
                                width: length,
                                height: 0.0005,
                                depth: 0.0005
                            )
    return ModelComponent(mesh: mesh, materials: [DistanceIndicator.lineMaterial])
}
```

Ενημέρωση (2 μέθοδοι)

```
func update_DistanceLine_Length(entity: Entity, length: Float) {
    let model = load_DistanceLine_ModelComponent(length: length)
    entity.components.set(model)
}

func update_DistanceLines(sourceLine: Entity, targetLine: Entity, length: Float) {
    /// Line length = distance/2. Also, keep 0.02m to give distance label some space.
    let lineLength: Float = length/2 - 0.015
```

```
    /// So set position.x diff to (line)length/2, which technically is
    /// (paramter=distance)length/4
    let xdif = length/4 + 0.0075
    sourceLine.setPosition(SIMD3<Float>(-xdif, 0, 0), relativeTo: sourceLine.parent)
    targetLine.setPosition(SIMD3<Float>(xdif, 0, 0), relativeTo: targetLine.parent)

    /// Actucally, set line length to distance/2, minus the space we
    /// gave for the  distance label
    self.update_DistanceLine_Length(entity: sourceLine, length: lineLength)
    self.update_DistanceLine_Length(entity: targetLine, length: lineLength)
}
```

B.5.4 Μοντέλο σώματος διανύσματος δύναμης

Δημιουργία (2 μέθοδοι)

```
func load_ArrowBody(pointEntity: Entity, magnitude: Float) -> Entity {
    let bodyEntity: Entity = Entity()
    pointEntity.addChild(bodyEntity)
    bodyEntity.name = "Arrow Body Entity"

    return bodyEntity
}

private func load_ArrowBody_ModelComponent(length: Float) -> ModelComponent{
    let material: SimpleMaterial = {
        var mat = SimpleMaterial()
        mat.metallic = MaterialScalarParameter(floatLiteral: 0.2)
        mat.roughness = MaterialScalarParameter(floatLiteral: 0.1)
        mat.tintColor = UIColor.white

        return mat
    }()
    let model: ModelComponent = ModelComponent(
                                    mesh: .generateBox(
                                        width: 0.001,
                                        height: 0.001,
                                        depth: length),
                                    materials: [material]
                                )
    return model
}
```

Ενημέρωση (2 μέθοδοι)

```
func update_ArrowBody(arrowBodyEntity: Entity, magnitude: Float) {
    /// Get the actual length in meters
```

```
    let length = magnitude * Force.metersPerNewton

    /// Arrow Body center needs to be <arrow-length>/2 + <pointCharge-Radius>
    /// meters away from Pivot Entity (Arrow's Parent)
    let pos = length/2 + 0.02
    arrowBodyEntity.setPosition(
                        SIMD3<Float>(0, 0, pos),
                        relativeTo: arrowBodyEntity.parent
                    )

    /// Update Arrow Body Length
    self.update_ArrowBody_Length(arrowBodyEntity: arrowBodyEntity, length: length)
}

private func update_ArrowBody_Length(arrowBodyEntity: Entity, length: Float) {
    let model = load_ArrowBody_ModelComponent(length: length)
    arrowBodyEntity.components.set(model)
}
```

Β.5.5 Μοντέλο κεφαλής διανύσματος δύναμης

Δημιουργία (1 μέθοδος)

```
func load_ArrowHead(on arrowEntity: Entity, magnitude: Float) {
    let arrowHeadAnchor = try! ArrowHead.load_ArrowHead()
    let arrowHeadEntity = arrowHeadAnchor.arrowHead! as Entity
    /// <Set name>
    arrowHeadEntity.name = "Arrow Head Entity"
    /// <Add Head to Body>
    arrowEntity.addChild(arrowHeadEntity)
    /// <Scale the model (0.1)>
    arrowHeadEntity.setScale(SIMD3<Float>(0.1,0.1,0.1), relativeTo: arrowHeadEntity)
    /// <Positioning>
    /// Get actual length in meters
    let length = magnitude * Force.metersPerNewton
    /// Set Position relatively to arrow entity (distance = half its length)
    /// CAREFUL: TO BE EXACT FOR THIS ARROW HEAD MODEL, REMOVE <----0.005m---->
    arrowHeadEntity.setPosition(SIMD3<Float>(0, 0, length/2 - 0.005), relativeTo:
arrowEntity)
}
```

Ενημέρωση (1 μέθοδος)

```
func update_ArrowHead(on arrowEntity: Entity, magnitude: Float) {
    /// Find arrow Head Entity and update
    arrowEntity.children.forEach{ entity in
        if entity.name == "Arrow Head Entity" {
```

```
            let arrowHeadEntity = entity
            /// - Tag: Positioning
            /// Get actual length in meters
            let length = magnitude * Force.metersPerNewton
            /// Set Position relatively to arrow entity (distance = half its length)
            /// CAREFUL: TO BE EXACT FOR THIS ARROW HEAD MODEL,
            /// REMOVE <----0.005m---->
            arrowHeadEntity.setPosition(SIMD3<Float>(0, 0, length/2 - 0.005),
relativeTo: arrowEntity)
        }
    }
}
```

B.5.6 Μοντέλο επιγραφής

Δημιουργία (1 μέθοδος)

```
func load_TextEntity(on entity: Entity, inside topology: Topology, name: String,
position: SIMD3<Float>, occlusion: Bool = false) -> Entity {
    let textPivotEntity: Entity = Entity()
    textPivotEntity.name  = "Label Entity"
    textPivotEntity.setParent(entity)
    textPivotEntity.setPosition(position, relativeTo: entity)
    let textEntity: Entity = Entity()
    textEntity.name = name
    textEntity.setParent(textPivotEntity)

    /// Add label to Topology's labels[]
    topology.labels.append(textPivotEntity)

    return textPivotEntity
}
```

Ενημέρωση (2 μέθοδοι)

```
func update_TextEntity(textEntity: Entity, stringValue: String, fontSize: Float =
0.012) {
    let defaultFont = "TimesNewRomanPSMT"
    var materialsArray: [Material] = [EntityStore.shared.textMaterial]
    /// Get the Label Entity, by accessing the Label Pivot Entity's child
    let text = textEntity.children[0]
    text.name == "Force Label" ?
materialsArray.append(EntityStore.shared.textOcclussionMaterial) : nil
    let mesh = .generateText(
                    stringValue,
                    extrusionDepth: 0.001,
```

```
                    font: UIFont(name: defaultFont, size: CGFloat(fontSize))!,
                    containerFrame: .zero,
                    alignment: .center,
                    lineBreakMode: .byCharWrapping
                    )
    let model: ModelComponent = ModelComponent(mesh: mesh, materials: materialsArray)
    text.components.set(model)

    /// Update the xdif from the pivot (xdif: width of text Mesh / 2)
    let pos = SIMD3<Float>(-model.mesh.bounds.boundingRadius, 0, 0)
    text.setPosition(pos, relativeTo: textEntity)
}

func update_AllTextOrientation(in topology: Topology) {
    topology.labels.forEach{ label in
        label.look(
            at: cameraTransform.translation,
            from: label.position(relativeTo: nil),
            relativeTo: nil
        )
        label.setOrientation(
            simd_quatf(angle: 180, axis: SIMD3<Float>(0, 1, 0)),
            relativeTo: label
        )
    }
}
```

Β.6 Μέθοδοι διαχείρισης της κλάσης Topology

Β.6.1 Τοποθέτηση τοπολογίας στη σκηνή (pinToScene)

```
//  Add the parent viewController and an ARAnchorEntity
func pinToScene(viewController: ViewController, topoAnchor: AnchorEntity) {
    self.viewController = viewController

    /// Add the Anchor Entity to the scene (where the user tapped)
    self.topoAnchorEntity = topoAnchor
    self.viewController?.arView.scene.addAnchor(self.topoAnchorEntity)

    /// Create the Points of Charge Observers (value change or deletion)
    self.viewController?.setupObserverCoulomb()
    self.viewController?.setupObserverPointChargeDeletion()
}
```

Β.6.2 Εμφάνιση των μοντέλων της τοπολογίας (placeTopology)

```
func placeTopology(topoModel: TopologyModel) {
    /// Clear the topology, if there was one
    self.clearTopology()

    /// Set new selectedPositions
    self.selectedPositions = {
        var positions: [SIMD3<Float>] = []
        topoModel.pointCharges.forEach{ p in
            positions.append(p.position)
        }
        return positions
    }()

    /// Create PointCharges in the selected Positions
    for pointChargeModel in topoModel.pointCharges {
        self.add(pointCharge: pointChargeModel)
    }

    /// Add all forces to all the pointCharge Objects
    self.reloadAllForces()
    /// Add all distance indicators
    self.reloadDistanceIndicators()

    /// Show Forces and Distance Indicators only for selected pointCharge
    self.showForces(for: selectedPointChargeObj)
    self.showDistaneIndicators(for: selectedPointChargeObj)
}
```

Β.6.3 Διαγραφή της τρέχουσας τοπολογίας (clearTopology)

```
func clearTopology() {
    /// Clear all Forces and Distance Indicators
    self.clearAllForces()
    self.clearDistanceIndicators()

    /// Clear all PointCharges
    self.removeAllPointCharges()

    /// Clear selectedPositions
    self.selectedPositions.removeAll()

    /// Clear all Labels
    self.labels.removeAll()

    /// Set selected objects to none
    longPressedEntity = Entity()
    trackedEntity = Entity()
    selectedPointChargeObj = PointChargeClass(on: Entity(), inside: self, withValue:
0)
}
```

Β.6.4 Προσθήκη φορτίου στο αντικείμενο τοπολογίας της σκηνής (add)

```
func add(pointCharge p: PointChargeModel) {
    /// Load the PointChargeEntity by cloning it
    let point = self.pointChargeEntityTemplate?.clone(recursive: true)
    /// Add it to the AnchorEntity of the Topology
    self.topoAnchorEntity.addChild(point!)

    /// Create a position based on PointChargeModel coordinates
    let pos: SIMD3<Float> = p.position
    /// Set its position relative to the Anchor Entity
    point?.setPosition(pos, relativeTo: self.topoAnchorEntity)

    /// Create new PointChargeClass Object and append it to pointCharges[]
    let newPointChargeObj = PointChargeClass(
        on: point!,
        inside: self,
        withValue: p.value
    )
    self.pointCharges.append(newPointChargeObj)

    /// Set selectedEntity (longPressedEntity)
    selectedPointChargeObj = newPointChargeObj
```

```
    longPressedEntity = point!

    /// Install gestures, careful to set its ".cancelTouchesInView" to
    /// false cause it cancels touches gestures  other than
    /// the installed below (I do that in Topology Placement)
    self.viewController?.arView.installGestures(
        [.translation, .rotation],
         for: point as! HasCollision
    )

    /// Enable the pointCharge LongPress Recognizer
    /// Careful that the above installedGesutres for translation and rotation
    /// disable the rest recognizers so for them set ".cancelsTouchesInView"
    /// to false, so that LongPress Recognizer is active
    self.viewController?.enableRecognizers(withName: "Long Press Recognizer")
}
```

Β.6.5 Προσθήκη φορτίου σε τυχαία θέση στη σκηνή (randomPosition)

```
/// Calculate a random position
private func randomPosition() -> SIMD3<Float>{
    let randPos: SIMD3<Float>

    var xmax: Float = -10000
    var xmin: Float = 10000
    var zmax: Float = -10000
    var zmin: Float = 10000

    /// If there are not enough ( less than 2 ) pointCharges select a random position
    if self.pointCharges.count < 2 {
        if self.selectedPositions.isEmpty {
            /// If there are no selectedPositions, no topo was selected from the menu.
            /// So, no pointCharge is present.
            /// So set a random position in a radius = 0.1 around the Anchor
            /// Entity of the topology (around the center)

            xmin = -0.1
            xmax = 0.1
            zmin = -0.1
            zmax = 0.1
        } else {
            /// Else if there are selectedPositions, only 1 pointCharge
            /// is being present (caused we are in case  less than 2)
            /// So, create a radius of 0.1 around it
            let selectedPosition = self.selectedPositions[0]

            xmin = selectedPosition.x - 0.1
            xmax = selectedPosition.x + 0.1
```

```
            zmin = selectedPosition.z - 0.1
            zmax = selectedPosition.z + 0.1
        }

    /// Else select one between the current pointCharges' positions
    } else {
        self.pointCharges.forEach{ pointChargeObj in
            let pos = pointChargeObj.entity.position

            if pos.x > xmax { xmax = pos.x }
            if pos.x < xmin { xmin = pos.x }
            if pos.z > zmax { zmax = pos.z }
            if pos.z < zmin { zmin = pos.z }
        }
    }

    let randPos_x = Float.random(in: xmin ..< xmax)
    let randPos_z = Float.random(in: zmin ..< zmax)
    randPos = SIMD3<Float>(randPos_x, 0, randPos_z)

    return randPos
}
```

B.6.6 Αφαίρεση επιλεγμένου φορτίου από την τρέχουσα τοπολογία (removePointCharge)

```
func removePointCharge() {
    /// First, find the selectedPointChargeObj and remove it from pointCharges[]
    for i in 0..<self.pointCharges.count {
        if selectedPointChargeObj == self.pointCharges[i] {
            self.pointCharges.remove(at: i)
            break
        }
    }
    /// And then remove the longPressedEntity (selected Entity)
    longPressedEntity.removeFromParent()

    /// Finally, calculate again All Forces and Distance Indicators
    self.reloadAllForces()
    self.reloadDistanceIndicators()

    /// Set a new random selecrtedPointChargeObj …
    if self.pointCharges.count > 0 {
        selectedPointChargeObj = self.pointCharges.last!
        longPressedEntity = selectedPointChargeObj.entity
    }
    /// ... and show the forces relative to it
```

```
    self.showForces(for: selectedPointChargeObj)
    self.showDistaneIndicators(for: selectedPointChargeObj)
}
```

Β.7 Κλάσεις των μεγεθών μίας τοπολογίας

Β.7.1 Η κλάση PointChargeClass

Ένα αντικείμενο PointChargeClass περιέχει τις παρακάτω ιδιότητες:

- **id**: Int
  Μοναδικός αριθμός ταυτοποίησης του αντικειμένου.

- **entity**: Entity
  Το αντικείμενο *Entity* στο οποίο βρίσκεται το τρισδιάστατο μοντέλο του φορτίου.

- **topology**: Topology
  Το αντικείμενο Topology της τρέχουσας τοπολογίας στην οποία ανήκει το φορτίο.

- **label**: Entity
  Το αντικείμενο *Entity* στο οποίο βρίσκεται το τρισδιάστατο μοντέλο της επιγραφής της τιμής του φορτίου.

- **value**: Float
  Η τιμή του φορτίου σε Coulomb.

- **netForce**: NetForce
  Το αντικείμενο *NetForce* που αντιπροσωπεύει τη συνισταμένη δύναμη που δέχεται το φορτίο.

- **forcesOnOthers**: [SingleForce]
  Μία λίστα από αντικείμενα *SingleForce* που αντιπροσωπεύουν τις δυνάμεις που ασκεί το φορτίο στα υπόλοιπα φορτία της τοπολογίας.

- **distanceIndicators**: [DistanceIndicator]
  Μία λίστα από αντικείμενα *DistanceIndicator* που αντιπροσωπεύουν τους δείκτες απόστασης του φορτίου με κάθε άλλο φορτίο.

Εκτός από τις παραπάνω ιδιότητες, η κλάση *PointChargeClass* περιέχει και τρεις στατικές μεταβλητές. Αυτές αντιπροσωπεύουν το συνολικό αριθμό φορτίων, το μήκος της ακτίνας με βάση την οποία σχεδιάζεται το τρισδιάστατο μοντέλο του φορτίου (*pointChargeRadius*) και τον πολλαπλασιαστή της τιμής του φορτίου ο οποίος δίνει στο φορτίο τιμή της κλίμακας μC (μίκρο κουλόμπ). Παρακάτω αναλύουμε τις κλάσεις των περιεχομένων μίας τοπολογίας.

Β.7.2 Η κλάση Force

Ένα αντικείμενο κλάσης Force περιέχει τις παρακάτω ιδιότητες:

- **forceId**: Int
  Ο μοναδικός αριθμός ταυτοποίησης του αντικειμένου.

- **topology**: Topology
  Το αντικείμενο τοπολογίας στο οποίο ανήκει το αντικείμενο.

- **magnitude**: Float
  Το μέτρο της δύναμης.

- **previousMagnitude**: Float
  Το προηγούμενο μέτρο της δύναμης

- **angle**: Float
  Το μέτρο της γωνίας σε ακτίνια.

- **arrowEntity**: Entity
  Το αντικείμενο *Entity* το οποίο αντιπροσωπεύει το τρισδιάστατο μοντέλο του βέλους του διανύσματος της δύναμης.

- **pivotEntity**: Entity
  Το αντικείμενο *Entity* στο οποίο τοποθετείται το *arrowEntity*. Το χρησιμοποιούμε για να περιστρέφουμε το *arrowEntity* χωρίς να περιστρέφουμε το ίδιο.

- **label**: Entity
  Το αντικείμενο *Entity* το οποίο αντιπροσωπεύει το τρισδιάστατο μοντέλο της επιγραφής τιμής της δύναμης.

- **selected**: Bool
  Μεταβλητή που δείχνει αν η δύναμη είναι η επιλεγμένη που προβάλλεται στην όψη επισκόπησης γωνίας.

- **type**: ForceType
  Ο τύπος της δύναμης (*single* ή *net*).

Επιπλέον, περιέχει μεταβλητές που εξαρτώνται από τις τιμές των παραπάνω ιδιοτήτων και ενημερώνονται σε κάθε αλλαγή αυτών των ιδιοτήτων.

- **X_Force_Component**: Float
  Η συνιστώσα της δύναμης στον οριζόντιο άξονα. Ενημερώνεται κατάλληλα κάθε φορά που αλλάζει η τιμή του μέτρου της δύναμης και της γωνίας.

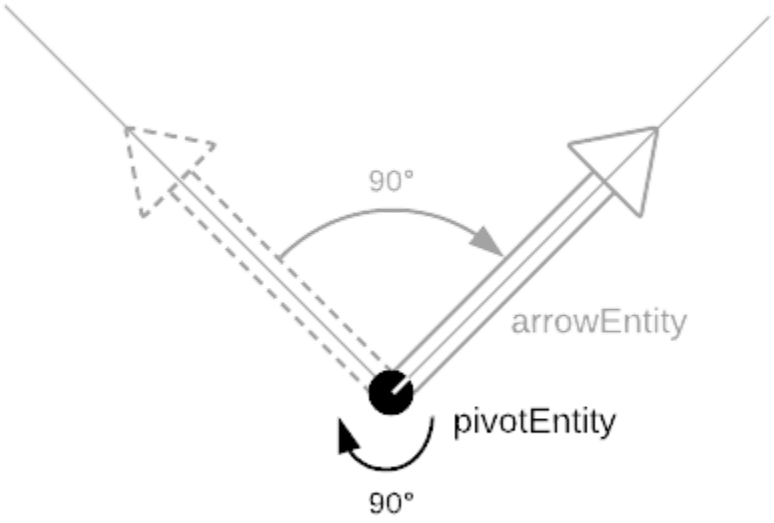
- **Y_Force_Component**: Float
  Η συνιστώσα της δύναμης στον κατακόρυφο άξονα. Ενημερώνεται κατάλληλα κάθε φορά που αλλάζει η τιμή του μέτρου της δύναμης και της γωνίας.

- **length**: Float
  Το μήκος του τρισδιάστατου μοντέλου του διανύσματος της δύναμης. Ενημερώνεται κατάλληλα κάθε φορά που αλλάζει η τιμή του μέτρου της δύναμης.

- **color**: UIColor
  Το χρώμα της δύναμης στην όψη επισκόπησης γωνίας. Ενημερώνεται κατάλληλα κάθε φορά που αλλάζει η τιμή *selected*.

Κατά την αρχικοποίηση ενός αντικειμένου *Force*, αυτό παίρνει τιμές για τον τύπο, την τοπολογία, το αντικείμενο *Entity* για το μοντέλο του διανύσματος δύναμης και για το μέτρο και τη γωνία της δύναμης. Τις τιμές αυτές τις παίρνει το αντικείμενο ως παραμέτρους στη μέθοδο αρχικοποίησης. Επιπλέον, δημιουργούμε ένα αντικείμενο *Entity* για την οντότητα περιστροφής *pivotEntity*. Τέλος, κατά την αρχικοποίηση καλούμε τη μέθοδο *load_TextEntity* για να δημιουργήσουμε μία οντότητα (*Entity*) για την επιγραφή της τιμής και το αναθέτουμε στην ιδιότητα *label*.

Οι μέθοδοι της κλάσης *Force* περιέχει αφορούν τη δημιουργία ενός τρισδιάστατου μοντέλου για το διάνυσμα της δύναμης καθώς και για την ενημέρωσή του, χρησιμοποιώντας τις μεθόδους *load_ArrowBody*, *update_ArrowBody*, *load_ArrowHead* της κλάσης *EntityStore*. Σε περίπτωση που αλλάξει η θέση, τιμή ή γωνία της δύναμης, καλείται η μέθοδος *updateArrowModel*, όπου ενημερώνουμε το μοντέλο του σώματος όσον το μήκος του σε σχέση με την οντότητα *pivotEntity* που βρίσκεται στην ίδια θέση στο χώρο με το φορτίο. Επειδή το μοντέλο του σώματος *arrowEntity* είναι τοποθετημένο σχετικά με το αντικείμενο *pivotEntity*, αρκεί να περιστρέψουμε το *pivotEntity* για να αλλάξουμε τη γωνία του μοντέλου του σώματος.



Σχήμα Β.1: Για να στρίψουμε το μοντέλο του βέλους *arrowEntity* αρκεί να περιστρέψουμε το σημείο και αντικείμενο *pivotEntity.*

Το μοντέλο της κεφαλής παραμένει πάντα στην ίδια θέση και γωνία σε σχέση με το μοντέλο του σώματος. Αν το σώμα αλλάξει μήκος επειδή η δύναμη αυξήθηκε ή μειώθηκε, τότε υπολογίζεται η απόσταση της κεφαλής από το κέντρο του σώματος όπου πρέπει να είναι το μισό του μήκος, και τοποθετείται κατάλληλα.

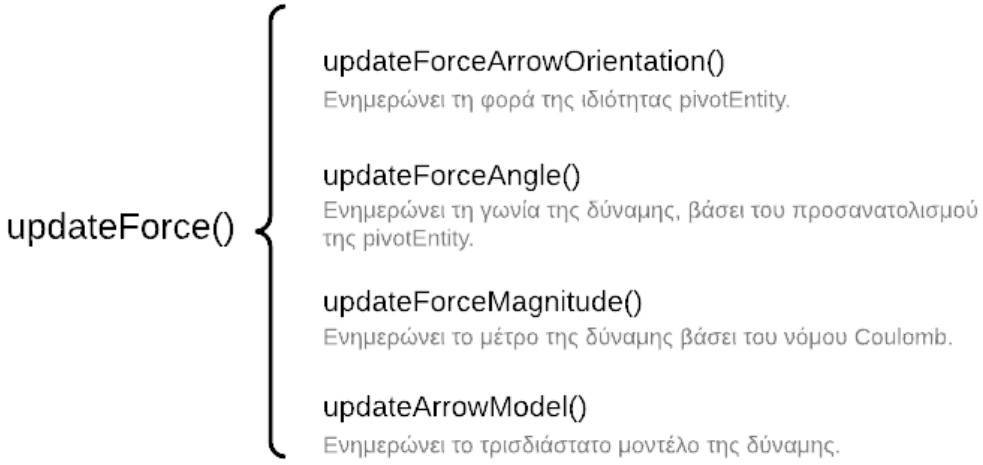
Βέβαια, εκτός από το μοντέλο του διανύσματος, κατά τη κλήση της μεθόδου *updateArrowModel* καλούμε και τη μέθοδο *updateLabel*, στην οποία ενημερώνουμε την ιδιότητα *label* με τη κλήση της μεθόδου *EntityStore.update_TextEntity*, προβάλλοντας έτσι την τρέχουσα τιμή της δύναμης στο μοντέλο επιγραφής της. Τέλος, για τον υπολογισμό των συνιστωσών της δύναμης δημιουργήσαμε τη μέθοδο *calculateForceComponent*, η οποία υπολογίζει και επιστρέφει το μέτρο της οριζόντιας ή κάθετης συνιστώσας με βάση το μέτρο και τη γωνία της δύναμης.

Β.7.3 Η κλάση SingleForce

Εκτός από τις ιδιότητες που κληρονομεί από την κλάση *Force*, η κλάση *SingleForce* περιέχει ακόμα τις ιδιότητες:

- **sourcePointCharge:** το φορτίο-πηγή που ασκεί τη δύναμη
- **targetPointCharge:** το φορτίο-στόχος στο οποίο ασκείται η δύναμη
- **attraction:** μία λογική μεταβλητή που ορίζει αν η δύναμη είναι ελκτική ή απωθητική και εξαρτάται από τις τιμές των φορτίων (κάθε φορά που αυτές αλλάζουν, ενημερώνεται κατάλληλα)

Κάθε φορά που δημιουργείται ένα αντικείμενο της κλάσης *SingleForce*, το προσθέτουμε στη λίστα *forcesOnOthers*[] του φορτίου που αποτελεί την πηγή της δύναμης (*sourcePointCharge*). Για την ενημέρωση του αντικειμένου *SingleForce*, δημιουργήσαμε τη μέθοδο *updateForce*, η οποία με τη σειρά της καλεί τις απαραίτητες μεθόδους για την ενημέρωση των στοιχείων της δύναμης.



Σχήμα Β.2: Η ενημέρωση μίας δύναμης περιλαμβάνει την ενημέρωση της γωνίας, του μέτρου και του μοντέλου της.

Β.7.4 Η κλάση NetForce

Εκτός από τις ιδιότητες που κληρονομεί από την κλάση *Force*, η κλάση *NetForce* περιέχει ακόμα τις ιδιότητες:

- **pointChargeObj:** το φορτίο στο οποίο ασκείται η συνισταμένη δύναμη
- **forces[]:** μία λίστα από τις συνιστώσες δυνάμεις (αντικείμενα *SingleForce*) από τις οποίες προκύπτει η συνισταμένη δύναμη

Για τον υπολογισμό της συνισταμένης δύναμης, δηλαδή για το μέτρο και τη γωνία της, δημιουργήσαμε τη μέθοδο *calculateNetForce*. Για κάθε αντικείμενο *SingleForce* στη λίστα *forces* προσθέτουμε τις οριζόντιες και τις κάθετες συνιστώσες, έως ότου έχουμε μία συνολική οριζόντια και μία συνολική κάθετη. Με βάση αυτές τις δύο, βρίσκουμε τη συνισταμένη και τη γωνία τους με τις μεθόδους *netForceMagnitude* και *netForceAngle* αντίστοιχα.

Β.7.5 Η κλάση DistanceIndicator

Η κλάση περιέχει τις παρακάτω ιδιότητες:

- **sourcePointCharge**: PointChargeClass
  Το πρώτο από τα δύο φορτία, το οποίο θεωρείται ως πηγή του δείκτη.

- **targetPointCharge**: PointChargeClass
  Το δεύτερο φορτίο, το οποίο θεωρείται ως προορισμός του δείκτη.

- **topology**: Topology
  Η τοπολογία στην οποία ανήκουν τα δύο φορτία και ο δείκτης.

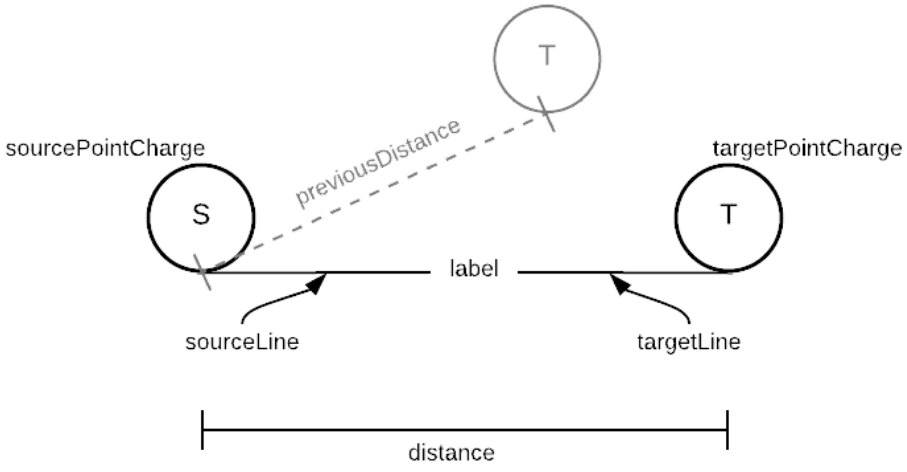
- **entity**: Entity
  Το αντικείμενο *Entity* πάνω στο οποίο τοποθετείται ο δείκτης.

- **sourceLine**: Entity
  Το αντικείμενο *Entity* που αντιπροσωπεύει το τρισδιάστατο μοντέλο του ευθύγραμμου τμήματος από την μεριά του φορτίου-πηγή.

- **targetLine**: Entity
  Το αντικείμενο *Entity* που αντιπροσωπεύει το τρισδιάστατο μοντέλο του ευθύγραμμου τμήματος από την μεριά του φορτίου-προορισμού.

- **label**: Entity
  Το αντικείμενο *Entity* που αντιπροσωπεύει το τρισδιάστατο μοντέλο επιγραφής της τιμής του μήκους του δείκτη, δηλαδή της απόστασης των δύο φορτίων.

Επιπλέον, περιέχεται και μία εξαρτημένη μεταβλητή με όνομα *distance*, η οποία αντιπροσωπεύει την απόσταση των δύο φορτίων και ενημερώνεται κάθε φορά που κάποιο από τα φορτία αλλάζει θέση στο χώρο. Τέλος, περιέχεται και η ιδιότητα *previousDistance* η οποία αναφέρεται στην προηγούμενη τιμή της απόστασης μεταξύ των φορτίων, και ανανεώνεται κάθε φορά που ανανεώνεται η μεταβλητή *distance*.



Σχήμα Β.3: Ο δείκτης απόστασης μεταξύ των φορτίων S και T και οι ιδιότητές του.

Κατά την αρχικοποίηση ενός αντικειμένου *DistanceIndicator* αναθέτουμε στις ιδιότητες *sourcePointCharge*, *targetPointCharge* και *topology* τις αντίστοιχες παραμέτρους που περάσανε στην κατασκευαστική μέθοδο κατά την κλήση.

Στη συνέχεια δημιουργούμε ένα αντικείμενο *Entity* για την τοποθέτηση του δείκτη με τη μέθοδο *EntityStore.load_DistanceIndicator* και το αναθέτουμε στην ιδιότητα *entity*. Έπειτα, δημιουργούμε ένα μοντέλο επιγραφής με τη μέθοδο *EntityStore.load_TextEntity* πάνω στο αντικείμενο *entity*. Ομοίως, δημιουργούμε πάνω στο αντικείμενο *entity* τα δύο μοντέλα ευθύγραμμων τμημάτων *sourceLine* και *targetLine* καλώντας κατάλληλα τη μέθοδο *EntityStore.load_DistanceLine*. Η δημιουργία αυτών των τεσσάρων οντοτήτων καθιστά ένα συνολικό τρισδιάστατο μοντέλο για το δείκτη απόστασης, το οποίο βρίσκεται πάνω στην ιδιότητα *entity*.

Μέθοδοι

Μετά τη δημιουργία του μοντέλου, επιβεβαιώνουμε ότι είναι ενημερωμένο καλώντας τις μεθόδους *updateLabel* και *updateLines*. Οι δύο αυτές μεθόδοι καλούν τις μεθόδους *EntityStore.update_TextEntity* και *EntityStore.update_DistanceLines* αντίστοιχα με τις κατάλληλες παραμέτρους, ώστε το μοντέλο του δείκτη να ενημερωθεί για την τρέχουσα θέση, γωνία και διάστασή του.

Όταν τα φορτία στην τοπολογία αλλάζουν, μεταξύ πολλών μεθόδων καλείται και η μέθοδος *Topology.updateDistanceIndicators*. Η μέθοδος αυτή καλεί με τη σειρά της τη μέθοδο *updateIndicator* για κάθε δείκτη απόστασης μεταξύ των φορτίων. Η μέθοδος *updateIndicator* αναλαμβάνει να ενημερώσει τη θέση, τον προσανατολισμό, τα μοντέλα των ευθύγραμμων τμημάτων και της επιγραφής. Η ενημέρωση της θέσης γίνεται εύκολα με τον υπολογισμό του ενδιάμεσου μεταξύ δύο σημείων στο χώρο, τις θέσεις των δύο φορτίων. Η ενημέρωση του προσανατολισμού γίνεται με χρήση της μεθόδου *look* της κλάσης *Entity*. Η μέθοδος αυτή προσανατολίζει το αντικείμενο *Entity* στο οποίο καλείται προς μία τοποθεσία στο χώρο. Είναι εύκολο να εξάγουμε την τοποθεσία του φορτίου προορισμού όπως φαίνεται στον κώδικα παρακάτω.

```
private func updateOrientation() {
    self.entity.look(
        at: targetPointCharge.entity.position,
        from: self.entity.position, relativeTo: self.entity.parent
    )
    /// Place it on the surface that the PointCharges sit on
    let ydif = -(1.5 * PointChargeClass.pointChargeRadius)
    self.entity.setPosition(SIMD3<Float>(0, ydif, 0), relativeTo: self.entity)

    self.entity.setOrientation(
        simd_quatf(angle: Int(270).degreesToRadians(),
        axis: SIMD3<Float>(0, 1.0, 0)),
```

```
        relativeTo: self.entity
    )
}
```

Στη συνέχεια, ενημερώνονται τα μοντέλα των ευθύγραμμων τμημάτων καλώντας τη μέθοδο *EntityStore.update_DistanceLines*, περνώντας ως παραμέτρους τα δύο φορτία και το μήκος της απόστασής τους. Τέλος, αν η απόσταση των φορτίων άλλαξε περισσότερο από 0.01 μέτρα, τότε ενημερώνουμε και το μοντέλο επιγραφής της απόστασης καλώντας τη μέθοδο *EntityStore.update_TextEntity* και περνώντας ως παραμέτρους την ιδιότητα *label* και τη νέα τιμή της απόστασης. Ο λόγος που κάνουμε τον έλεγχο των 0.01 μέτρων είναι ότι δε θέλουμε η εφαρμογή να δημιουργεί συνεχώς ένα τρισδιάστατο μοντέλο για την απόσταση καθώς τιμές κάτω του εκατοστού δεν έχουν σημασία και η εφαρμογή επιβαρύνεται σε μεγάλο βαθμό.

Β.8 Επισκόπηση γωνιών

Β.8.1 Η κλάση AnglesOverview

Όπως και κάθε άλλη κλάση όψης, η *AnglesOverview* πρέπει να σχεδιαστεί για να εμφανιστεί στην οθόνη με την μέθοδο *draw*. Για να ορίσουμε εμείς τον τρόπο με τον οποίο συμπεριφέρεται η όψη κατά το σχεδιασμό της, παρακάμπτουμε τη μέθοδο *draw* με μία δική μας εκδοχή.

```
override func draw(_ rect: CGRect) {
    // Obtain the center point of the pie chart
    let center = CGPoint(
        x: bounds.width / 2,
        y: bounds.height / 2
    )
    if selectedPointChargeObj.netForce != nil {
        if selectedPointChargeObj.netForce!.forces.count > 0 {
            drawForces(center: center)
        }
    }
}
```

Κάθε φορά που καλείται η μέθοδος *draw* για να σχεδιαστεί μία ενημερωμένη έκδοση της επισκόπησης των γωνιών, ελέγχουμε πρώτα αν το επιλεγμένο φορτίο δέχεται δυνάμεις. Αν δέχεται δυνάμεις, καλούμε τη μέθοδο *drawForces* που δημιουργήσαμε στην κλάση *AnglesOverview* η οποία σχεδιάζει τα διανύσματα δυνάμεων και τη γωνία της επιλεγμένης δύναμης.

Η μέθοδος *drawForces* καλεί τη μέθοδο *drawForce* για κάθε δύναμη που δέχεται το επιλεγμένο φορτίο. Η *drawForce* υπολογίζει τις διαστάσεις για τα δισδιάστατα γραφικά βέλη των δυνάμεων και στη συνέχεια τα σχεδιάζει χρησιμοποιώντας την κλάση *UIBezierPath*, η οποία προσφέρει μεγάλη ελευθερία στο σχεδιασμό δισδιάστατων γραφικών.

```
private func drawForces(center: CGPoint) {
    let netForce = selectedPointChargeObj.netForce!
    drawForce(
        center: center,
        angle: netForce.angle,
        forceType: netForce.type,
        color: netForce.color,
        selected: netForce.selected
    )

    netForce.forces.forEach{ f in
        drawForce(
            center: center,
            angle: f.angle,
            forceType: f.type,
            color: f.color,
```

```
            selected: f.selected
        )
    }
}

private func drawForce(center: CGPoint, angle: Float, forceType: ForceType, color:
UIColor, selected: Bool) {
    /**
    - Regulated Angle:
    To ensure that the angle representation and the
    vectors go the same way as the 3d models
    */

    let regulatedAngle = 360.degreesToRadians() - angle
    // Get the tail length
    let tailLength: Float = forceType == ForceType.single ? FORCE_ARROW_TAIL_LENGTH :
NETFORCE_ARROW_TAIL_LENGTH

    // Find the end point based on the angle
    let end_x_Point = CGFloat(center.x) + CGFloat( tailLength * cos(regulatedAngle) )
    let end_y_Point = CGFloat(center.y) + CGFloat( tailLength * sin(regulatedAngle) )

    // Create the arrow path
    let arrowPath = UIBezierPath.arrow(
        from: center,
        to: CGPoint(x: end_x_Point, y: end_y_Point),
        tailWidth: CGFloat(ARROW_TAIL_WIDTH),
        headWidth: CGFloat(HEAD_WIDTH),
        headLength: CGFloat(HEAD_LENGTH)
    )
    // Fill with color
    color.setFill()
    arrowPath.fill()

    // If force is the selected one, draw Arc
    if selected {
        drawAngleArc(center: center, angle: regulatedAngle)
    }
}
```

Για το σχεδιασμό ενός βέλους, δημιουργήσαμε μία επέκταση της κλάσης *UIBezierPath*, στην οποία ορίσαμε μία νέα μέθοδο κλάσης, την *arrow*. Η μέθοδος αυτή δέχεται ως παραμέτρους τις συντεταγμένες για την αρχή του βέλους και την κορυφή του, το πλάτος του σώματος και της κεφαλής του βέλους, καθώς και το μήκος της κεφαλής του. Με αυτά τα δεδομένα, σχεδιάζει ένα βέλος τύπου *UIBezierPath* και το επιστρέφει [39].

```
// Extension of UIBezierPath to draw Arrow shape path
extension UIBezierPath {
    class func arrow(
        from start: CGPoint,
```

```
        to end: CGPoint,
        tailWidth: CGFloat,
        headWidth: CGFloat,
        headLength: CGFloat
    ) -> Self {
        let length = hypot(end.x - start.x, end.y - start.y)
        let tailLength = length - headLength

        func p(_ x: CGFloat, _ y: CGFloat) -> CGPoint { return CGPoint(x: x, y: y) }
        let points: [CGPoint] = [
            p(0, tailWidth / 2),
            p(tailLength, tailWidth / 2),
            p(tailLength, headWidth / 2),
            p(length, 0),
            p(tailLength, -headWidth / 2),
            p(tailLength, -tailWidth / 2),
            p(0, -tailWidth / 2)
        ]
        let cosine = (end.x - start.x) / length
        let sine = (end.y - start.y) / length
        let transform = CGAffineTransform(
            a: cosine,
            b: sine,
            c: -sine,
            d: cosine,
            tx: start.x,
            ty: start.y
        )
        let path = CGMutablePath()
        path.addLines(between: points, transform: transform)
        path.closeSubpath()
        return self.init(cgPath: path)
    }
}
```

Η μέθοδος *drawForce*, αφού σχεδιάσει το βέλος του διανύσματος της δύναμης, σχεδιάζει και τη γωνία της δύναμης καλώντας τη μέθοδο *drawAngleArc*, αν η δύναμη αυτή φυσικά είναι η επιλεγμένη. Η μέθοδος *drawAngleArc* σχεδιάζει μία γωνία χρησιμοποιώντας τη προκαθορισμένη μέθοδο της κλάσης *UIBezierPath* και τη γωνία της δύναμης για την οποία σχεδιάζεται.

```
private func drawAngleArc(center: CGPoint, angle: Float) {
    let startAngle: CGFloat = 0
    let forceAngle: CGFloat = CGFloat(angle)
    // Create arc path starting from 0 angle
    let arcPath = UIBezierPath(
        arcCenter: center,
        radius: CGFloat(ARC_RADIUS),
        startAngle: startAngle,
```

```
        endAngle: forceAngle,
        clockwise: false
    )
    // Add line to center
    arcPath.addLine(to: center)

    // Fill color
    ARC_COLOR.setFill()
    arcPath.fill()
}
```

B.8.2 Χρώματα και διαστάσεις των γραφικών διανυσμάτων και γωνιών

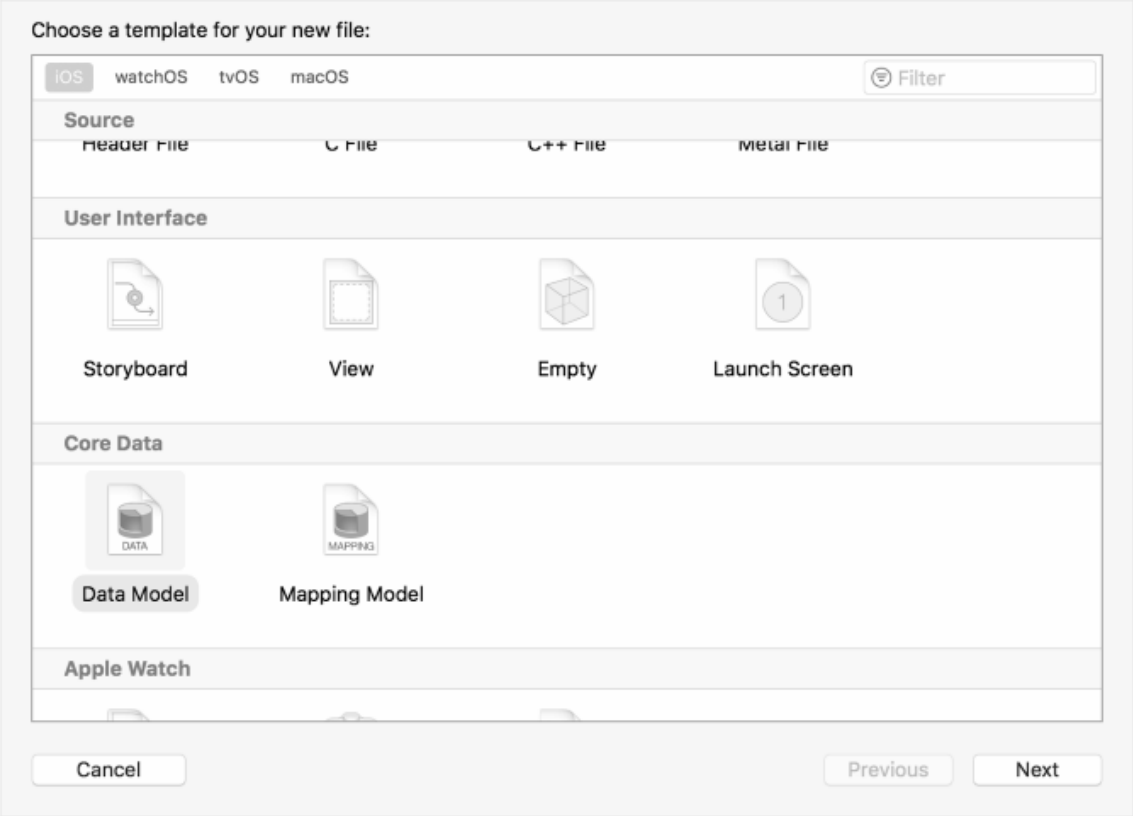
Για τις διαστάσεις των παραπάνω σχεδίων καθώς και το χρωματισμό τους, ορίσαμε τις παρακάτω σταθερές στη κλάση *AnglesOverview*. Οι σταθερές αυτές αναγράφονται και στα παραπάνω κομμάτια κώδικα που περιγράφουν τη δημιουργία των γραφικών.

```
let FORCE_ARROW_TAIL_LENGTH: Float = 30
let NETFORCE_ARROW_TAIL_LENGTH: Float = 50
let ARROW_TAIL_WIDTH: Float = 2
let HEAD_LENGTH: Float = 10
let HEAD_WIDTH: Float = 8
let ARC_RADIUS: Float = 10
let ARC_COLOR: UIColor = UIColor.orange
let NETFORCE_COLOR: UIColor = UIColor.white
let FORCE_COLOR: UIColor = UIColor.white
let SELECTED_FORCE_COLOR: UIColor = UIColor.yellow
```

Οι αριθμοί που αναγράφονται είναι σε σύστημα πόντων UIKit της οθόνης κι όχι σε μονάδες pixel. Παρατηρούμε ότι το μήκος του σώματος του βέλους για τη συνισταμένη δύναμη είναι μεγαλύτερο από το μήκους του σώματος του βέλους για κάθε συνιστώσα (50 πόντοι έναντι 30). Επιλέξαμε τη διαφορά μήκους ώστε η συνισταμένη δύναμη να ξεχωρίζει. Όποια δύναμη επιλέξει ο χρήστης, συνισταμένη ή συνιστώσα, τότε αυτή σχεδιάζεται με κίτρινο χρώμα.

B.9 Το εργαλείο Core Data και η υλοποίησή του στην εφαρμογή μας

Αρχικά έπρεπε να δημιουργήσουμε ένα νέο Core Data μοντέλο στο υπάρχον πρότζεκτ μας. Αυτό είναι εύκολο μέσα από το XCode ακολουθώντας το μονοπάτι *File* > *New* > *File* και επιλέγοντας το “Data Model” από τα iOS templates.



Εικόνα Β.1: Το μενού επιλογής template στο Xcode.

Στη συνέχεια, ονομάσαμε το νέο αρχείο τύπου *.xcdatamodeld* "Topologies" και πατήσαμε "Create" για τη δημιουργία του. Στο αρχείο "Topologies.xcdatamodeld" προσθέσαμε τα μοντέλα τοπολογίας και φορτίου ως οντότητες (*Entities*) πατώντας το κουμπί "Add Entity" και ονομαζοντάς τες κατάλληλα "NSTopology" και "NSPointCharge" αντίστοιχα. Κάθε κλάση οντότητας περιέχει ιδιότητες (attributes), τις οποίες έπρεπε να προσθέσουμε με βάση το σκοπό του μοντέλου μας. Στην κλάση οντότητα *NSTopology* προσθέσαμε τρεις ιδιότητες:

- **name**: String → το όνομα της τοπολογίας
- **descr**: String → το κείμενο περιγραφής της τοπολογίας
- **Image**: Binary Data → το στιγμιότυπο της τοπολογίας

Στην κλάση οντότητα *NSPointCharge* προσθέσαμε πέντε ιδιότητες:

- **value**: Float → η τιμή του φορτίου
- **multiplier**: Float → ο πολλαπλασιαστής της τιμής του φορτίου

- **posX**: Float → η τιμή της θέσης του φορτίου στον x άξονα
- **posY**: Float → η τιμή της θέσης του φορτίου στον y άξονα
- **posZ**: Float → η τιμή της θέσης του φορτίου στον z άξονα

Φυσικά δε θα ήταν εφικτό να περιγράψουμε εξ ολοκλήρου μία τοπολογία αν δεν προσθέταμε τη σχέση μεταξύ των κλάσεων *NSTopology* και *NSPointCharge*. Παρακάτω φαίνεται η επιφάνεια διεπαφής του Core Data framework στο XCode.

Εικόνα Β.2: Η διεπαφή με το Core Data που προσφέρει το Xcode στον προγραμματιστή.

Μόλις αποθηκεύσαμε το αρχείο "Topologies.xcdatamodeld" δημιουργήθηκαν αυτόματα τέσσερα αρχεία τα οποία περιέχουν τον ορισμό των παραπάνω κλάσεων οντοτήτων καθώς και τις ιδιότητές τους. Τα πρώτα δύο αρχεία ονομάζονται "NSPointCharge+CoreDataClass.swift" και "NSPointCharge+CoreDataProperties.swift" και περιέχουν τον ορισμό της κλάσης *NSPointCharge* και τις ιδιότητές της αντίστοιχα. Παρακάτω παραθέτουμε τα βασικά κομμάτια κώδικα του κάθε αρχείου.

- **NSPointCharge+CoreDataClass.swift**

```
@objc(NSPointCharge)
public class NSPointCharge: NSManagedObject {}
```

- **NSPointCharge+CoreDataProperties.swift**

```
extension NSPointCharge {
    @nonobjc public class func fetchRequest() -> NSFetchRequest<NSPointCharge> {
        return NSFetchRequest<NSPointCharge>(entityName: "NSPointCharge")
    }

    @NSManaged public var value: Float
    @NSManaged public var multiplier: Float
    @NSManaged public var posX: Float
    @NSManaged public var posY: Float
    @NSManaged public var posZ: Float
    @NSManaged public var topology: NSTopology?
}
```

Αντίστοιχα, τα άλλα δύο αρχεία ονομάζονται “NSTopology+CoreDataClass.swift” και “NSTopology+CoreDataProperties.swift” και περιέχουν τον ορισμό και τις ιδιότητες της κλάσης *NSTopology*. Επιπλέον, κάθε αντικείμενο *NSTopology* περιέχει μία λίστα τύπου *NSOrderedSet* με τα αντικείμενα *NSPointCharge* με τα οποία σχετίζεται, η οποία ορίζεται μαζί με τις υπόλοιπες ιδιότητες στο αρχείο “NSTopology+CoreDataProperties.swift”.

```
@NSManaged public var pointCharges: NSOrderedSet?
```

Ακόμα, το αρχείο αυτό περιέχει μεθόδους (accessors) για την πρόσβαση στα αντικείμενα *NSPointCharge* της λίστας αυτής.

```
// MARK: Generated accessors for pointCharges
extension NSTopology {

    @objc(insertObject:inPointChargesAtIndex:)
    @NSManaged public func insertIntoPointCharges(_ value: NSPointCharge, at idx: Int)

    @objc(removeObjectFromPointChargesAtIndex:)
    @NSManaged public func removeFromPointCharges(at idx: Int)

    @objc(insertPointCharges:atIndexes:)
    @NSManaged public func insertIntoPointCharges(_ values: [NSPointCharge], at
indexes: NSIndexSet)

    @objc(removePointChargesAtIndexes:)
    @NSManaged public func removeFromPointCharges(at indexes: NSIndexSet)

    @objc(replaceObjectInPointChargesAtIndex:withObject:)
    @NSManaged public func replacePointCharges(at idx: Int, with value: NSPointCharge)

    @objc(replacePointChargesAtIndexes:withPointCharges:)
    @NSManaged public func replacePointCharges(at indexes: NSIndexSet, with values:
[NSPointCharge])

    @objc(addPointChargesObject:)
    @NSManaged public func addToPointCharges(_ value: NSPointCharge)
```

```
    @objc(removePointChargesObject:)
    @NSManaged public func removeFromPointCharges(_ value: NSPointCharge)

    @objc(addPointCharges:)
    @NSManaged public func addToPointCharges(_ values: NSOrderedSet)

    @objc(removePointCharges:)
    @NSManaged public func removeFromPointCharges(_ values: NSOrderedSet)
}
```

Β.10 Η υλοποίηση μηνυμάτων προς το χρήστη με την κλάση Status και την όψη messagePanel

Β.10.1 Η κλάση Status

Η κλάση Status ενημερώνεται συνεχώς για την κατάσταση της εφαρμογής και εξάγει μηνύματα κατάστασης και προτεινόμενων ενεργειών τα οποία προορίζονται για το χρήστη. Για να ξεχωρίζουμε τους διαφορετικούς τύπους μηνυμάτων ορίσαμε μία απαριθμητική μεταβλητή.

```
enum MessageType {

    case trackingStateEscalation
    case planeEstimation
    case contentPlacement

    static var all: [MessageType] = [
        .trackingStateEscalation,
        .planeEstimation,
        .contentPlacement
    ]
}
```

Στην κλάση επίσης δημιουργήσαμε τις εξής μεθόδους που αφορούν τα παραπάνω μηνύματα.

- **showMessage**: Εμφανίζει το μήνυμα το οποίο περνάει ως παράμετρος.
- **scheduleMessage**: Προγραμματίζει την εμφάνιση ενός μηνύματος σε κάποιο χρονικό διάστημα, καθώς και τον τύπο του προγραμματισμένου μηνύματος.
- **cancelScheduledMessage**: Ακυρώνει ένα προγραμματισμένο μήνυμα που περνάει ως παράμετρος.
- **cancelAllScheduledMessages**: Ακυρώνει όλα τα προγραμματισμένα μηνύματα.

Ακόμα, δημιουργήσαμε δύο μεθόδους οι οποίες χρησιμοποιούν τις παραπάνω και εμφανίζουν τον κατάλληλο μήνυμα αναλόγως την κατάσταση της ιχνηλάτησης της εφαρμογής.

- **showTrackingQualityInfo**: Λαμβάνει ως παράμετρο την κατάσταση ιχνηλάτησης του αντικειμένου *ARCamera* της εφαρμογής, και καλεί την *showMessage* με το ανάλογο μήνυμα.
- **escalateFeedback**: Ανάλογα με την κατάσταση ιχνηλάτησης, παρουσιάζει στο χρήστη ένα μήνυμα που του προτείνει τις ενέργειες που πρέπει να κάνει στη συνέχεια.

Ο τρόπος με τον οποίο επιλέγεται το σωστό μήνυμα είναι μέσω μίας επέκτασης που δημιουργήσαμε στην κλάση *ARCamera.TrackingState*. Στην επέκταση αυτή δημιουργήσαμε δύο μεταβλητές οι οποίες αλλάζουν τιμή ανάλογα με την κατάσταση της AR κάμερας.

```
extension ARCamera.TrackingState {
    var presentationString: String {
        switch self {
            case .notAvailable:
                return "TRACKING UNAVAILABLE"
            case .normal:
                return "TRACKING NORMAL"
            case .limited(.excessiveMotion):
                return "TRACKING LIMITED\nExcessive motion"
            case .limited(.insufficientFeatures):
                return "TRACKING LIMITED\nLow detail"
            case .limited(.initializing):
                return "Initializing"
            case .limited(.relocalizing):
                return "Recovering from interruption"
            @unknown default:
                return "Unknown tracking state."
        }
    }
    var recommendation: String? {
        switch self {
            case .limited(.excessiveMotion):
                return "Try slowing down your movement, or reset the session."
            case .limited(.insufficientFeatures):
                return "Try pointing at a flat surface, or reset the session."
            case .limited(.relocalizing):
                return "Return to the location where you left off or try resetting the
session."
            default:
                return nil
        }
    }
}
```

Επομένως, η εφαρμογή είναι σε θέση ανά πάσα στιγμή να ανιχνεύσει και να ενημερώσει αυτόματα το χρήστη για την κατάσταση της εφαρμογής, με το κατάλληλο μήνυμα. Επιλέγουμε να το κάνουμε αυτό όποτε η κατάσταση της εφαρμογής αλλάζει. Για παράδειγμα, αν η κατάσταση μεταβεί από *.normal* σε *.limited* η εφαρμογή θα καλέσει πρώτα τη μέθοδο *showTrackingQualityInfo* για να ειδοποιήσει το χρήστη για την αλλαγή κατάστασης, ενώ στη συνέχεια θα καλέσει την *escalateFeedback* για να παροτρύνει το χρήστη στις απαραίτητες ενέργειες που χρειάζονται για να επαναφέρει την εφαρμογή σε κατάσταση *.normal*.

Οι παραπάνω μέθοδοι μας δίνουν τη δυνατότητα να γράψουμε και δικά μας μηνύματα προς το χρήστη. Επομένως, αν θέλουμε να προγραμματίσουμε ένα μήνυμα τύπου *contentPlacement* προς το χρήστη σε δύο δευτερόλεπτα που τον παροτρύνει να πατήσει την οθόνη για να τοποθετήσει μία τοπολογία, αρκεί η παρακάτω γραμμή κώδικα:

```
self.status?.scheduleMessage("TAP TO PLACE TOPOLOGY", inSeconds: 2, messageType:
.contentPlacement)
```

B.10.2 Η όψη messagePanel και η αλληλεπίδρασή της με την κλάση Status

Για να παρουσιάσουμε τα μηνύματα που μπορούμε να δημιουργήσουμε με την κλάση *Status*, δημιουργήσαμε την όψη *messagePanel* τύπου *UIView.* Μέσα σε αυτή δημιουργήσαμε μία όψη τύπου *UIPaddingLabel*, η οποία θα προβάλλει μία συμβολοσειρά με το κατάλληλο μήνυμα. Κάθε φορά που καλείται η μέθοδος *showMessage* της κλάσης *Status*, ενημερώνεται και η όψη ώστε το μήνυμα να φτάνει στην οθόνη του χρήστη. Η όψη κρύβεται και εμφανίζεται, αναλόγως με την κατάσταση της ιχνηλάτησης της εφαρμογής, μέσω της μεθόδου *setMessageHidden* της κλάσης *Status*. Ορίζουμε την όψη όταν αρχικοποιείται η κλάση *ViewController*, ορίζοντας τους χωρικούς περιορισμούς καθώς και τις παραμέτρους στοίχισης, γραμματοσειράς και χρώματος.

Βιβλιογραφία - Αναφορές

[1] Megan Tocci, «History and Evolution of Smartphones» [Ηλεκτρονικό]. Available: https://simpletexting.com/where-have-we-come-since-the-first-smartphone/

[2] Kevin Jackson, «A brief history of the smartphone», July 2018 [Ηλεκτρονικό]. Available: https://sciencenode.org/feature/How%20did%20smartphones%20evolve.php

[3] Wikipedia, «Smartphone» [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Smartphone

[4] Wikipedia, «Mobile Operating System» [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Mobile_operating_system

[5] J.kiran kumar, D.Yugandhar, «A Study on Current Mobile Operating Systems», May 2017 [Ηλεκτρονικό]. Available: https://www.ijser.org/researchpaper/A-Study-on-Current-Mobile-Operating-Systems.pdf

[6] Ana Javornik, «The Mainstreaming of Augmented Reality: A Brief History» [Ηλεκτρονικό]. Available: https://hbr.org/2016/10/the-mainstreaming-of-augmented-reality-a-brief-history

[7] Tulane University, «What's the difference between AR and VR» [Ηλεκτρονικό]. Available: https://sopa.tulane.edu/blog/whats-difference-between-ar-and-vr

[8] R. Silva, J. C. Oliveira, G. A. Giraldi, «Introduction to Augmented Reality» [Ηλεκτρονικό]. Available: https://www.lncc.br/~jauvane/papers/RelatorioTecnicoLNCC-2503.pdf

[9] Wikipedia, «Augmented reality» [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Augmented_reality

[10] Ana Javornik, «The Mainstreaming of Augmented Reality: A Brief History» [Ηλεκτρονικό]. Available: https://hbr.org/2016/10/the-mainstreaming-of-augmented-reality-a-brief-history

[11] Apple Developer Documentation, «Xcode Overview» [Ηλεκτρονικό]. Available: https://developer.apple.com/xcode/

[12] Apple Developer Documentation, «SwiftUI Overview» [Ηλεκτρονικό]. Available: https://developer.apple.com/xcode/swiftui/

[13] Apple Developer Documentation, «Augmented Reality» [Ηλεκτρονικό]. Available: https://developer.apple.com/augmented-reality/

[14] Apple Developer Documentation, «ARKit Overview» [Ηλεκτρονικό]. Available: https://developer.apple.com/augmented-reality/arkit/

[15] Daniel Eran Dilger, «Inside Apple's ARKit and Visual Inertial Odometry, new in iOS 11», Oct 2017 [Ηλεκτρονικό]. Available: https://appleinsider.com/articles/17/10/12/inside-apples-arkit-and-visual-inertial-odometry-new-in-ios-11

[16] Apple Developer Documentation, «AR Creation Tools» [Ηλεκτρονικό]. Available: https://developer.apple.com/augmented-reality/tools/

[17] Apple Developer Documentation, «Resources» [Ηλεκτρονικό]. Available: https://developer.apple.com/augmented-reality/resources/

[18] Apple Developer Documentation, «ARView» [Ηλεκτρονικό]. Available: https://developer.apple.com/documentation/realitykit/arview

[19] Apple Developer Documentation, «ARCoachingOverlayView» [Ηλεκτρονικό]. Available: https://developer.apple.com/documentation/arkit/arcoachingoverlayview

[20] Apple Developer Documentation, «ARSession» [Ηλεκτρονικό]. Available: https://developer.apple.com/documentation/arkit/arsession

[21] Apple Developer Documentation, «ARAnchor» [Ηλεκτρονικό]. Available: https://developer.apple.com/documentation/arkit/aranchor

[22] Apple Developer Documentation, «Content Anchors» [Ηλεκτρονικό]. Available: https://developer.apple.com/documentation/arkit/content_anchors

[23] Apple Developer Documentation, «ARPlaneAnchor» [Ηλεκτρονικό]. Available: https://developer.apple.com/documentation/arkit/arplaneanchor

[24] Apple Developer Documentation, «Entity» [Ηλεκτρονικό]. Available: https://developer.apple.com/documentation/realitykit/entity

[25] Apple Developer Documentation, «ModelEntity» [Ηλεκτρονικό]. Available: https://developer.apple.com/documentation/realitykit/modelentity

[26] Apple Developer Documentation, «AnchorEntity» [Ηλεκτρονικό]. Available: https://developer.apple.com/documentation/realitykit/anchorentity

[27] Apple Developer Documentation, «NotificationCenter» [Ηλεκτρονικό]. Available: https://developer.apple.com/documentation/foundation/notificationcenter

[28] Apple Developer Documentation, «Tracking and Visualizing Planes» [Ηλεκτρονικό]. Available:

https://developer.apple.com/documentation/arkit/content_anchors/tracking_and_visualizing_planes

[29] Apple Developer Documentation, «Managing Session Life Cycle and Tracking Quality» [Ηλεκτρονικό]. Available: https://developer.apple.com/documentation/arkit/managing_session_life_cycle_and_tracking_quality

[30] Apple Developer Documentation, «ARWorldTrackingConfiguration» [Ηλεκτρονικό]. Available: https://developer.apple.com/documentation/arkit/arworldtrackingconfiguration

[31] Apple Developer Documentation, «ARHitTestResult» [Ηλεκτρονικό]. Available: https://developer.apple.com/documentation/arkit/arhittestresult

[32] Apple Developer Documentation, «Occluding Virtual Content with People» [Ηλεκτρονικό]. Available: https://developer.apple.com/documentation/arkit/camera_lighting_and_effects/occluding_virtual_content_with_people

[33] Apple Developer Documentation, «Placing Objects and Handling 3D Interaction» [Ηλεκτρονικό]. Available: https://developer.apple.com/documentation/arkit/environmental_analysis/placing_objects_and_handling_3d_interaction

[34] Apple Developer Documentation, «Gestures» [Ηλεκτρονικό]. Available: https://developer.apple.com/documentation/swiftui/gestures

[35] Apple Developer Documentation, «HasCollision» [Ηλεκτρονικό]. Available: https://developer.apple.com/documentation/realitykit/hascollision

[36] Apple Developer Documentation, «Install Gestures» [Ηλεκτρονικό]. Available: https://developer.apple.com/documentation/realitykit/arview/3368185-installgestures

[37] Apple Developer Documentation, «Setting Up a Core Data Stack» [Ηλεκτρονικό]. Available: https://developer.apple.com/documentation/coredata/setting_up_a_core_data_stack

[38] Apple Developer Documentation, «UIView» [Ηλεκτρονικό]. Available: https://developer.apple.com/documentation/uikit/uiview

[39] StackOverflow, «How to draw a directional arrow head» [Ηλεκτρονικό]. Available: https://docs.google.com/document/d/16QbtfE0szGxGxYbiVC7aj1PeIfUuviYJmP1TnY3KO4A/edit#

Βιβλιογραφία Εικόνων και Σχημάτων

1. Εικόνα 1.1: Το Simon Personal Communicator (1994), «History and Evolution of Smartphones» [Ηλεκτρονικό]: https://simpletexting.com/where-have-we-come-since-the-first-smartphone/

2. Σχήμα 1.1: Η σταδιακή αύξηση των έξυπνων συσκευών την τελευταία δεκαετία, «Number of Smartphone users from 2016 to 2021» [Ηλεκτρονικό]: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

3. Σχήμα 1.2: Το ποσοστό της αγοράς για τα λειτουργικά συστήματα των έξυπνων συσκευών, «Mobile Operating System Market Share Worldwide» [Ηλεκτρονικό]: https://gs.statcounter.com/os-market-share/mobile/worldwide

4. Εικόνα 2.1: Τα σημεία ενδιαφέροντος (feature points), Mark Dawson, Jun 2017, «ARKit by Example - Plane Detection and Visualization» [Ηλεκτρονικό]: https://blog.markdaws.net/arkit-by-example-part-2-plane-detection-visualization-10f05876d53

5. Σχήμα 2.4: Το σύστημα οντοτήτων σε μία σκηνή επαυξημένης πραγματικότητας, «Entity» [Ηλεκτρονικό]: https://developer.apple.com/documentation/realitykit/entity

6. Σχήμα 2.5: Τα συστατικά που περιέχονται σε κάθε βασική κατηγορία οντοτήτων, «Entity» [Ηλεκτρονικό]: https://developer.apple.com/documentation/realitykit/entity

7. Εικόνα 2.2: Οπτική αναπαράσταση ενός hit test, Neil Mathew, Feb 2020, «Design a Responsive 3D Cursor for ARKit apps in Unity» [Ηλεκτρονικό]: https://placenote.com/blog/real-world-cursor-with-arkit-hittest/

8. Εικόνα 2.3: «Occluding Virtual Content with People» [Ηλεκτρονικό]: https://developer.apple.com/documentation/arkit/camera_lighting_and_effects/occluding_virtual_content_with_people

9. Εικόνα 2.4: «Occluding Virtual Content with People» [Ηλεκτρονικό]: https://developer.apple.com/documentation/arkit/camera_lighting_and_effects/occluding_virtual_content_with_people

10. Εικόνα 2.5: «Occluding Virtual Content with People» [Ηλεκτρονικό]: https://developer.apple.com/documentation/arkit/camera_lighting_and_effects/occluding_virtual_content_with_people