ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Διπλωματική Εργασία

ΕΦΑΡΜΟΓΗ ΕΠΑΥΞΗΜΕΝΗΣ ΠΡΑΓΜΑΤΙΚΟΤΗΤΑΣ ΓΙΑ ΔΙΑΔΡΑΣΤΙΚΟΥΣ ΓΕΩΜΕΤΡΙΚΟΥΣ ΜΕΤΑΣΧΗΜΑΤΙΣΜΟΥΣ ΤΡΙΣΔΙΑΣΤΑΤΩΝ ΜΟΝΤΕΛΩΝ CAD

Άγγελος Ε. Μαρινάκης

Εξεταστική Επιτροπή:
Δρ. Αικατερίνη Μανιά (Επιβλέπουσα)
Δρ. Βασίλειος Σαμολαδάς
Δρ. Αριστομένης Αντωνιάδης

Χανιά, Οκτώβριος 2018

Με την ολοκλήρωση της παρούσας διπλωματική εργασίας, θα ήθελα να ευχαριστήσω την Αναπ. Καθηγήτρια Αικατερίνη Μανιά για την ανάθεση και την άμεση υποστήριξη που μου παρείχε, χωρίς την οποία η εκπόνηση της εργασίας δεν θα ήταν εφικτή. Επιπλέον ευχαριστώ θερμά τον Καθηγητή Αριστομένη Αντωνιάδη για την καθοδήγηση και άμεση βοήθειά του καθ' όλη τη διάρκεια της εργασίας. Ιδιαίτερες ευχαριστίες οφείλω στις συμφοιτήτριες Αιμιλία Κανιαδάκη και Ιωάννα Πατεράκη για τη συμβολή τους στην παρούσα εργασία. Τέλος, ένα μεγάλο ευχαριστώ οφείλω στους φίλους μου και ιδιαίτερα στην οικογένειά μου για τη συνεχή υποστήριξη που μου παρείχε καθ' όλη τη διάρκεια των σπουδών μου.

ΠΕΡΙΕΧΟΜΕΝΑ

1. ΕΙΣΑΓΩΓΗ

1.1 Αντικείμενο και στόχος της εργασίας

Η επαυξημένη πραγματικότητα χρησιμοποιείται με αυξανόμενο ρυθμό τα τελευταία χρόνια και λόγω των ευκολιών που προσφέρει, χρησιμοποιείται σε πολλούς κλάδους όπως η ιατρική, η εκπαίδευση, η αυτοκινητοβιομηχανία, κ.λπ.. Σκοπός της παρούσας εργασίας είναι η υλοποίηση μίας εκπαιδευτικής εφαρμογής για φορητές συσκευές, η οποία χρησιμοποιεί την τεχνολογία της επαυξημένης πραγματικότητας ως βασικό εργαλείο της και διαχειρίζεται τρισδιάστατα γεωμετρικά μοντέλα παραγόμενα από λογισμικό CAD. Πιο συγκεκριμένα, η εφαρμογή αφορά την αναγνώριση στερεών μοντέλων και την εισαγωγή αυτών σε ένα περιβάλλον επεξεργασίας. Το περιβάλλον αυτό, μέσα από μία σειρά γεωμετρικών μετασχηματισμών προσφέρει στο χρήστη τη δυνατότητα καλύτερης αντίληψης και εποπτείας του μοντέλου. Βασικές λειτουργίες είναι η μετακίνηση, η περιστροφή και η μεγέθυνση του στερεού. Το μεγαλύτερο και πιο σημαντικό κομμάτι της εργασίας αφιερώθηκε στην υλοποίηση μίας τομής του στερεού από ένα επίπεδο σε έναν από τους βασικούς άξονες x, y και z. Η τομή υλοποιείται μέσω μίας σειράς αλγορίθμων υπολογιστικής γεωμετρίας που εφαρμόζονται στο δίκτυο γεωμετρίας του εξεταζόμενου μοντέλου. Η εφαρμογή προσφέρει επιπλέον λειτουργίες όπως ο υπολογισμός της επιφάνειας, του όγκου και του βάρους του στερεού που εξετάζεται καθώς και πρόσβαση σε βασικές πληροφορίες που σχετίζονται με αυτό, όπως οι διαστάσεις του. Ο χρήστης της εφαρμογής έχει επιπλέον τη δυνατότητα αποθήκευσης του μοντέλου τοπικά στη συσκευή του για offline επεξεργασία. Τέλος, στις περισσότερες περιπτώσεις είναι δυνατή η προβολή του μηχανολογικού σχεδίου του εκάστοτε στερεού.

1.2 Δομή Εργασίας

Το **κεφάλαιο 1** παρουσιάζει συνοπτικά το αντικείμενο της παρούσας εργασίας.

Στο **κεφάλαιο 2** αναφέρονται συνοπτικά οι ορισμοί για τα βασικά αντικείμενα της εργασίας.

Στο **κεφάλαιο 3** επεξηγείται η δομή της μηχανής γραφικών που χρησιμοποιήθηκε για την υλοποίηση της εργασίας. Δίνεται μεγαλύτερη έμφαση στις απαραίτητες γνώσεις που χρειάστηκαν κατά τη διάρκεια υλοποίησής της.

Στο **κεφάλαιο 4** επεξηγείται ο τρόπος προβολής και αναγνώρισης τρισδιάστατων μοντέλων με τη χρήση της κάμερας της φορητής συσκευής και της τεχνολογίας της επαυξημένης πραγματικότητας.

Στο **κεφάλαιο 5** παρουσιάζεται αναλυτικά η διαδικασία που εφαρμόστηκε για την υλοποίηση μίας τομής στερεού. Αναφέρονται αναλυτικά όλοι οι αλγόριθμοι που χρησιμοποιήθηκαν και πιθανές τροποποιήσεις αυτών για την ορθή λειτουργία τους στο περιβάλλον της Unity.

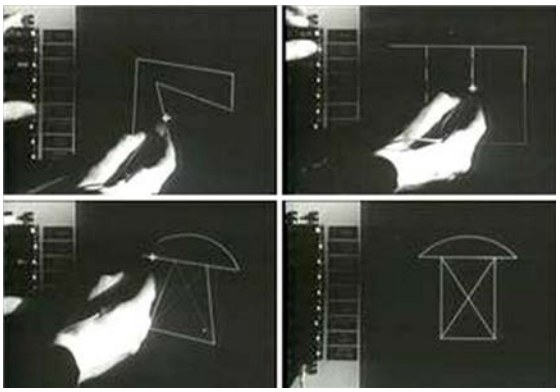
Στο **κεφάλαιο 6** παρουσιάζονται βασικές λειτουργίες της εφαρμογής οι οποίες υλοποιούνται με τον κατάλληλο μετασχηματισμό της κάμερας του περιβάλλοντος ανάπτυξης.

Τέλος, στο **κεφάλαιο 7** παρουσιάζεται το κύριο περιβάλλον επεξεργασίας της εφαρμογής το οποίο αποτελείται από μία γραφική διεπαφή χρήστη και το μοντέλο προς επεξεργασία.

2. ΣΤΑΘΜΗ ΤΩΝ ΓΝΩΣΕΩΝ

2.1 Computer-Aided Design

Ο όρος CAD (Computer Aided Design) [1] αναφέρεται στη χρήση υπολογιστικών συστημάτων με σκοπό την παραγωγή, διαφοροποίηση ή βελτίωση ενός σχεδίου. Τα λογισμικά CAD διευκολύνουν την εργασία του σχεδιαστή, προσφέροντας μεγαλύτερη ακρίβεια και αυτοματοποίηση των διαδικασιών σχεδίασης. Συχνά χρησιμοποιούνται για την παραγωγή δισδιάστατων ή τρισδιάστατων διαγραμμάτων, για τα οποία μπορεί να γίνει εξαγωγή στη μορφή ηλεκτρονικών αρχείων προς εκτύπωση ή για άλλους βιομηχανικούς σκοπούς.



Σχήμα 2.1: Παρουσιάση του Sketchpad από τον Ivan Sutherland

Οι σχεδιαστές χρησιμοποιούν τους υπολογιστές ως ένα σημαντικό εργαλείο σχεδίασης ήδη από τα μέσα της δεκαετίας του 1950. Οι κατασκευαστές των πρώτων υπολογιστών δημιούργησαν μικροπρογράμματα τα οποία βοηθούσαν τους προγραμματιστές στην αποσφαλμάτωση προγραμμάτων με τη χρήση διαγραμμάτων ροής. Έτσι, ανακαλύφθηκε ότι ήταν δυνατή η δημιουργία ηλεκτρονικών συμβόλων και γεωμετρικών σχημάτων για την κατασκευή διαγραμμάτων για απλά κυκλώματα ή διαγράμματα ροής. Η πρώτη εμφάνιση λογισμικού CAD οφείλεται στο Μηχανικό **Pierre Bezier**, ο οποίος ανέπτυξε το πρόγραμμα **UNISURF** την περίοδο μεταξύ 1966-1968 για τη διευκόλυνση της σχεδίασης εξαρτημάτων στην αυτοκινητοβιομηχανία. Το πρόγραμμά του χρησιμοποιήθηκε ευρέως από τις επόμενες γενιές. Η καινοτομία που επέφερε δραστική αλλαγή, ήταν το γραφικό περιβάλλον που διέθετε το λογισμικό **SKETCHPAD**, το οποίο σχεδιάστηκε από τον **Ivan Sutherland** στα εργαστήρια του MIT κατά τη διάρκεια της διδακτορικής διατριβής του. Ο σχεδιαστής μπορούσε πλέον να αλληλεπιδράσει με τον υπολογιστή (βλ. σχήμα 2.1) και να εισάγει σε αυτόν σχέδια με τη χρήση ενός light pen σε μία οθόνη CRT.

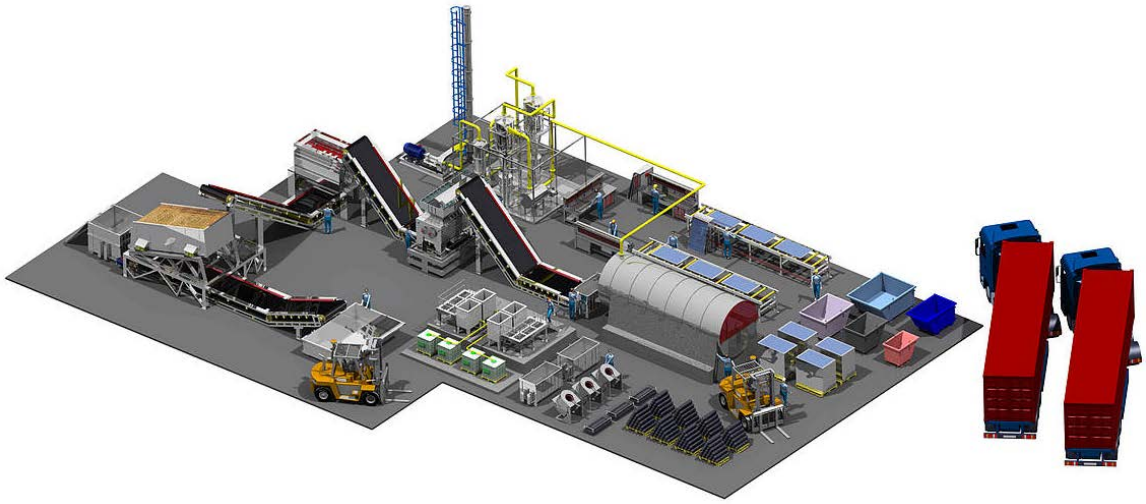
Τα πρώτα συστήματα CAD [2] "έτρεχαν" σε ένα μεγάλο κεντρικό υπολογιστή που τροφοδοτούσε ομάδες γραφικών τερματικών. Σταδιακά όμως, και καθώς ο λόγος κόστος/απόδοση έπεφτε συνεχώς, δημιουργούνταν όλο και μικρότεροι σε μέγεθος υπολογιστές που παρείχαν την αναγκαία ισχύ για γραφική υποστήριξη πολλαπλών χρηστών. Έτσι, δημιουργήθηκε ένα ρεύμα προς τους μίνι-υπολογιστές, τους supermicros και τα workstations. Σήμερα οι γραφικές δυνατότητες που παρέχουν ακόμη και τα PCs έχουν ως φυσικό επακόλουθο να έχει αλλάξει και η φιλοσοφία ανάπτυξης ενός τμήματος CAD. Οι σημερινές τάσεις που απαντώνται σχετίζονται με το μέγεθος της εταιρείας-χρήστη: Οι μεγάλοι χρήστες έχουν ένα διανεμημένο περιβάλλον, όπου οι κεντρικοί υπολογιστές χρησιμοποιούνται για καταχώρηση των σχεδίων, ελεγχόμενη πρόσβαση σε ακριβούς σχεδιογράφους (plotters) και πολύπλοκα πακέτα ανάλυσης. Οι μικρότεροι και μεμονωμένοι χρήστες χρησιμοποιούν workstations και προσωπικούς ηλεκτρονικούς υπολογιστές (PCs) με ενσωματωμένες γραφικές δυνατότητες.

Τα πρώτα συστήματα ακολούθησαν την ανάπτυξη του στερεού μοντέλου από την δυσδιάστατη γεωμετρία του. Ο χρήστης όριζε την ακριβή γεωμετρία του αντικειμένου και κάθε αλλαγή στις διαστάσεις και τη γεωμετρία ήταν χρονοβόρα και το στερεό μοντέλο ήταν μια πρόσθετη εφαρμογή που μπορούσε να ενεργοποιήσει. Η σημερινή τάση στα συστήματα

μηχανολογικής σχεδίασης είναι τα παραμετρικά μοντέλα με τη χρήση μορφολογικών χαρακτηριστικών. Η παραμετρική μοντελοποίηση εμφανίσθηκε το 1987 από την εταιρεία Parametric Technology (σύστημα Pro/ENGINEER) και από τότε όλοι οι προμηθευτές προσπαθούν να παρουσιάσουν ένα αντίστοιχο προϊόν. Τον Ιανουάριο του 1994 υπήρχαν επτά αντίστοιχα συστήματα και η τάση είναι να αυξάνονται συνέχεια. Σήμερα όλοι οι προμηθευτές συστημάτων CAD για μηχανολογική σχεδίαση παρέχουν κάποιο αντίστοιχο προϊόν. Όλα όμως τα εμπορικά συστήματα στηρίζονται σε περιορισμένο αριθμό πυρήνων, για τη δημιουργία του συστήματος. Οι πυρήνες αυτοί είναι, ACIS από την Spatial Technology, Parasolid από την EDS, Granite/1 από την PTC. Μερικά από τα σημερινά εμπορικά συστήματα φαίνονται στον επόμενο πίνακα 2.1 ενώ στο σχήμα 2.2 παρουσιάζεται το μοντέλο μίας γραμμής ανακύκλωσης σχεδιασμένο μέσα από το περιβάλλον της πλατφόρμας σχεδίασης CATIA (Το συγκεκριμένο μοντέλο κέρδισε το διαγωνισμό POTY2k15 που διοργανώθηκε από την εταιρεία λογισμικών Dassault Systèmes και ανέδειξε την ομάδα του Πολυτεχνείου Κρήτης στην πρώτη θέση ανάμεσα από 432 υποψηφιότητες από όλον τον κόσμο).

ΕΤΑΙΡΕΙΑ	ΠΡΟΪΟΝ	ΠΥΡΗΝΑΣ
Parametric Technology	Pro/ENGINEER - Wildfire	PTC
EDS	Unigraphics	Parasolid
EDS-Intergraph	Solid Edge	Parasolid
Bentley	Microstation Modeler	Parasolid
AUTODESK	Mechanical Desktop, Inventor	ACIS
DASSAULT-Solidworks	Solidworks	Parasolid
DASSAULT	CATIA 5	Parasolid
HP	HP Designer	ACIS

Πίνακας 2.1: Συστήματα CAD & CAD/CAM



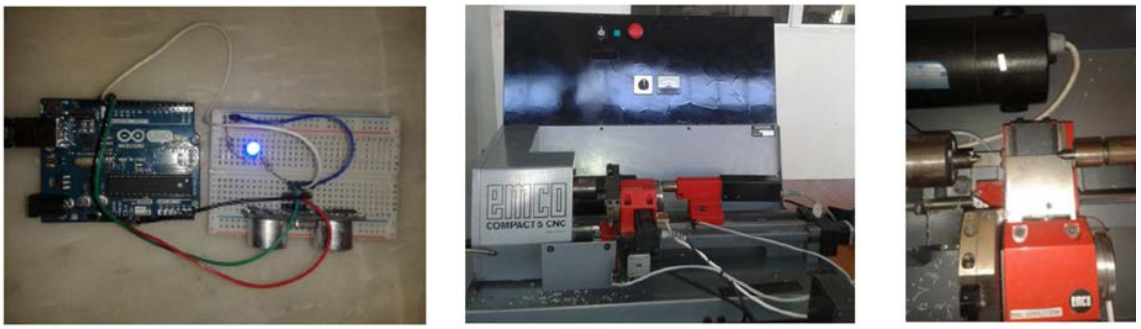
Σχήμα 2.2: Μοντέλο CAD χωροθέτησης γραμμής ανακύκλωσης (Πολυτεχνείο Κρήτης)

Η συνεχής εξέλιξη που παρατηρείται στους τομείς των γραφικών υπολογιστών και των συστημάτων CAD απαιτεί την επίλυση πολύπλοκων γεωμετρικών προβλημάτων, αποστολή που έχει αναλάβει η **υπολογιστική γεωμετρία** [3]. Η υπολογιστική γεωμετρία είναι ένας κλάδος της επιστήμης υπολογιστών, ο οποίος αφιερώνεται στη μελέτη αλγορίθμων, οι οποίοι διαχειρίζονται γεωμετρικά προβλήματα. Τέτοια προβλήματα παρουσιάζονται επίσης στη ρομποτική, τη μηχανική όραση, στη σχεδίαση ενσωματωμένων κυκλωμάτων, κ.λπ., καθιστώντας την ένα πολύτιμο εργαλείο της επιστήμης. Κάποια γεωμετρικά προ-

βλήματα που θα μελετηθούν στην παρούσα εργασία, αφορούν τον διαχωρισμό ενός πολυγώνου σε υποπολύγωνα συγκεκριμένων ιδιοτήτων καθώς και την τριγωνοποίηση του, τα οποία θα αναφερθούν αναλυτικά στο κεφάλαιο 5.

2.2 Επαυξημένη πραγματικότητα

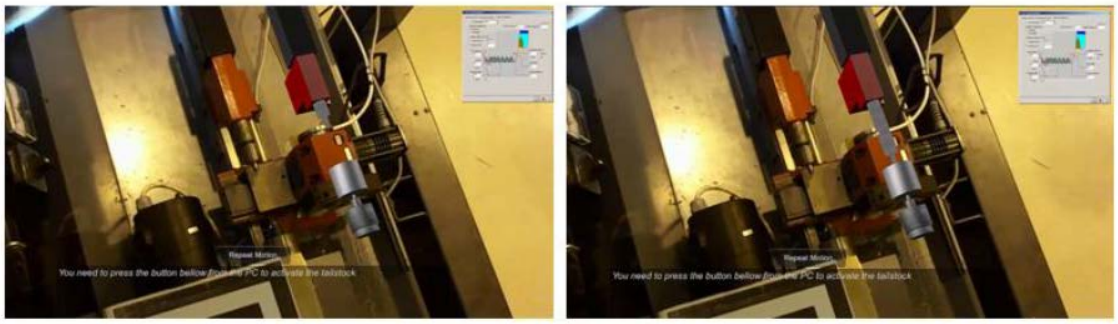
Με τον όρο επαυξημένη πραγματικότητα ονομάζεται η προβολή ψηφιακών εικόνων, βίντεο ή τρισδιάστατων μοντέλων στον πραγματικό κόσμο. Δηλαδή, η επαυξημένη πραγματικότητα διαφοροποιεί την αντίληψη ενός ατόμου για τον πραγματικό κόσμο, προσθέτοντας ψηφιακό περιεχόμενο σε αυτόν. Συχνά ο όρος ταυτίζεται λανθασμένα με την εικονική πραγματικότητα η οποία αποτελεί πλήρη αντικατάσταση του πραγματικού κόσμου με κάποιον εικονικό κόσμο. Η επαυξημένη πραγματικότητα έχει πολλές εφαρμογές στα ηλεκτρονικά παιχνίδια καθώς και σε ιατρικές και άλλες βιομηχανικές εφαρμογές. Ο **Τζίμας Ε.** [4] διερεύνησε τη χρήση της επαυξημένης πραγματικότητας στον κατασκευαστικό κλάδο, αναπτύσσοντας δύο εφαρμογές οι οποίες υποστηρίζουν τη διαδικασία προετοιμασίας εργαλειομηχανών. Η πρώτη εφαρμογή σκοπεύει στη δημιουργία ενός εικονικού οδηγού, ο οποίος κατευθύνει τον χρήστη στον τρόπο στήριξης ενός κυλινδρικού τεμαχίου στον τόρνο EMCO CNC (βλ. σχήμα 2.3).



Σχήμα 2.3: Εφαρμογή επαυξημένης πραγματικότητας σε CNC τόρνο (ΕΜΠ)

Για την ορθή λειτουργία της εφαρμογής απαιτείται η εισαγωγή σε αυτήν του ύψους του κυλινδρικού τεμαχίου που πρόκειται να στηριχθεί. Η εισαγωγή του ύψους γίνεται είτε χειροκίνητα από το χρήστη σε συγκεκριμένο πεδίο, είτε με μέτρηση του κυλινδρικού τεμαχίου από μία διάταξη που φέρει έναν αισθητήρα μέτρησης μήκους και συνδέεται με την εφαρμογή μέσω μίας πλακέτας Arduino. Η εφαρμογή ξεκινά τη λειτουργία της μετά από αναγνώριση μίας εικόνας - στόχου που βρίσκεται στον πραγματικό κόσμο. Έτσι, μέσω γραπτών οδηγιών, εικόνων και κινούμενων αντικείμενων που προβάλλονται στην κάμερα, ο χρήστης κατευθύνεται βήμα-βήμα στην τοποθέτηση του κυλινδρικού τεμαχίου στον τόρνο. Στη συνέχεια, παρουσιάζονται μερικά στιγμιότυπα από τη διαδικασία καθοδήγησης (βλ. σχήμα 2.4, 2.5).

Σχήμα 2.4: Ένδειξη τοποθέτησης



Σχήμα 2.5: Ολοκλήρωση επίδειξης και εμφάνιση υπόδειξης χρήσης του λογισμικού του τόρνου

Η δεύτερη εφαρμογή στοχεύει επίσης στη δημιουργία ενός εικονικού οδηγού, αυτή τη φορά για τη στήριξη συγκεκριμένων τεμαχίων στην τράπεζα κέντρου κατεργασιών για τη μετέπειτα επεξεργασία τους. Βασικό πλεονέκτημα της εφαρμογής αυτής είναι η συνεισφορά που μπορεί να έχει στην εκπαίδευση προσωπικού ή φοιτητών, αντικαθιστώντας έτσι εγχειρίδια χρήσης ή προφορικές οδηγίες. Η εικόνα - στόχος για την εφαρμογή παραμένει όμοια με την αντίστοιχη της πρώτης εφαρμογής. Σημειώνεται, ότι η υπέρθεση των εικονικών μοντέλων που προβάλλονται, γίνεται αναφορικά με τη θέση και τον προσανατολισμό της εικόνας – στόχου.

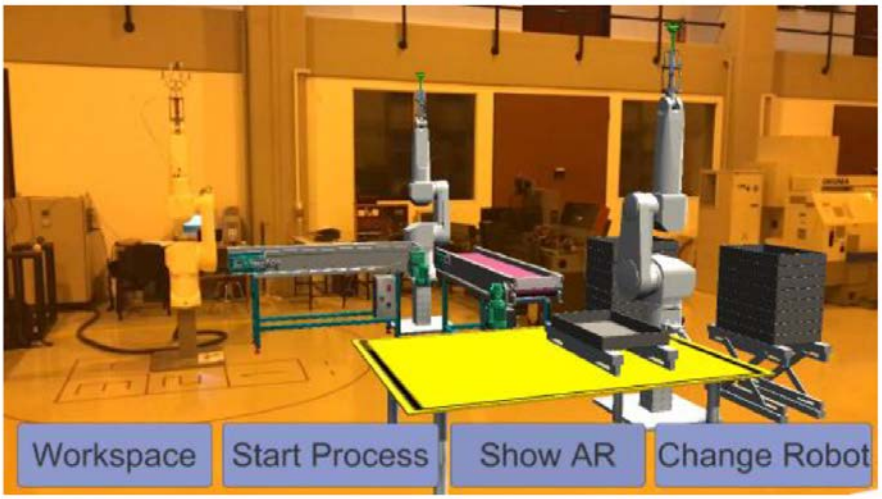
Σχήμα 2.6: Επιλογή αντικειμένου προς κατεργασία

Η εφαρμογή ξεκινά και ο χρήστης επιλέγει μεταξύ δύο αντικείμενων που θα τοποθετήσει στην τράπεζα κατεργασίας, όπως φαίνεται στο σχήμα 2.6. Στη συνέχεια, μέσω κατάλληλης λίστας, επιλέγεται ο προσανατολισμός για τον οποίο θα καταδειχθεί η στήριξη του τεμαχίου. Αναλόγως τον προσανατολισμό που επιλέχθηκε, προτείνεται και διαφορετικός τρόπος συγκράτησης του αντικειμένου στην τράπεζα κατεργασίας. Οι πιθανοί τρόποι συγκράτησης είναι η χρήση τσοκ, ιδιοσυσκευών σύσφιξης καθώς και η χρήση μέγγενης, όπως παρουσιάζονται στο σχήμα 2.7.

Σχήμα 2.7: Ενδείξεις τρόπου συγκράτησης τεμαχίου

Ο **Κόκκας Α.** [5] με τη χρήση της επαυξημένης πραγματικότητας σχεδίασε μία μηχανουργική διάταξη και συγκεκριμένα ένα Ευέλικτο Σύστημα Κατεργασιών (FMS), δηλαδή ένα σύστημα που αποτελείται από ψηφιακά καθοδηγούμενες εργαλειομηχανές, συνδεμένες μεταξύ τους μέσω ενός κεντρικού συστήματος ελέγχου. Στο σύστημα αυτό παρέχονται κατεργασμένα και ακατέργαστα τεμάχια και ο σκοπός του είναι η βελτιστοποίηση της αποδοτικότητας κάθε εργαλειομηχανής ξεχωριστά. Η εφαρμογή που αναπτύχθηκε αποσκοπεί στην υπέρθεση των στοιχείων που λείπουν από ένα πλήρες “κύτταρο” FMS, έτσι ώστε να γίνει αποτύπωση των πιθανών εναλλακτικών διατάξεων και στη συνέχεια να κα-

θοριστεί η βέλτιστη τελική χωροθέτηση. Επιπλέον, επιχειρείται η λειτουργική σύνδεση και προσομοίωση της συνεργασίας των εικονικών μοντέλων με τις πραγματικές εργαλειομηχανές και ρομποτικούς βραχίονες του εργαστηρίου, δηλαδή ένας ψηφιακά καθοδηγούμενος τόρνος EMCO και ένα ρομπότ Staubli. Η προσομοίωση της παραγωγικής διαδικασίας με κινούμενα μέρη, προσφέρει τη δυνατότητα εκτενούς ανάλυσης της διάταξης των μηχανών έτσι ώστε αυτή να αποτελέσει τη βέλτιστη και ασφαλέστερη λύση. Ο χρήστης επαυξημένης πραγματικότητας μπορεί να αξιολογήσει μία διάταξη με την περιήγησή του στον μελλοντικό χώρο παραγωγής, λαμβάνοντας υπόψη παράγοντες που δεν μπορούν να αξιολογηθούν με διαφορετικό τρόπο. Στο σχήμα 2.8 παρουσιάζεται ένα στιγμιότυπο της εφαρμογής όπου παρατηρείται η διάταξη των εικονικών μοντέλων σε σχέση με τις πραγματικές μηχανές στο χώρο του εργαστηρίου.



Σχήμα 2.8: Στιγμιότυπο εφαρμογής όπου παρουσιάζεται μία πιθανή διάταξη εικονικών μοντέλων σε σχέση με πραγματικές μηχανές (ΕΜΠ)

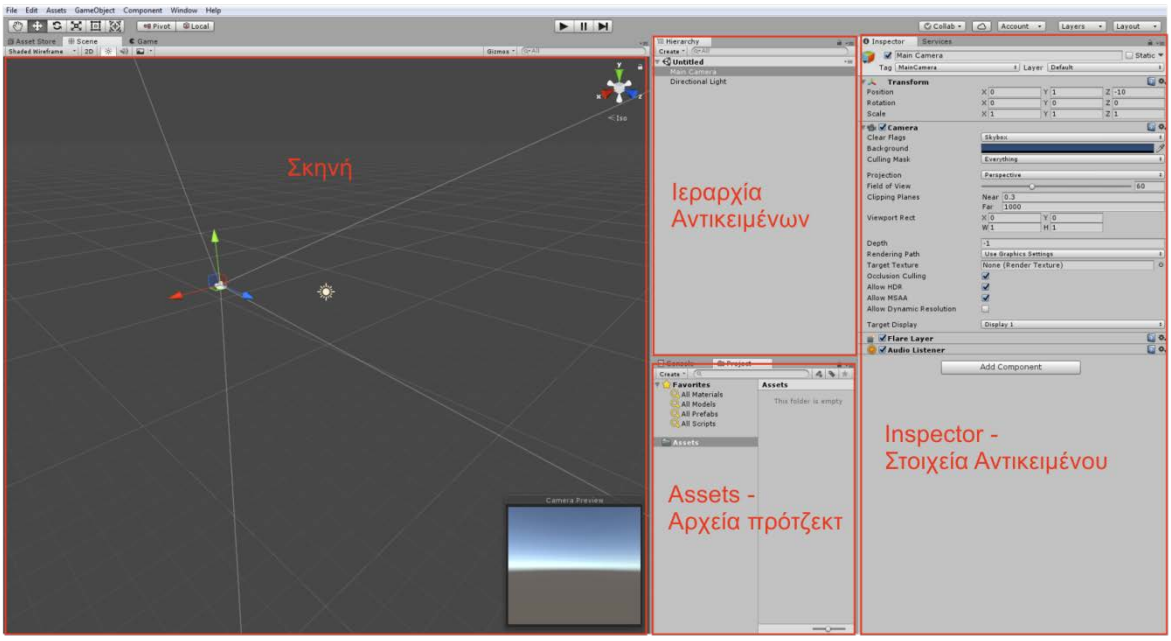
Η εφαρμογή βασίστηκε σε ένα συγκεκριμένο σενάριο παραγωγής, το οποίο περιλαμβάνει την κατεργασία ενός κυλινδρικού τεμαχίου. Τα εικονικά μοντέλα προγραμματίστηκαν έτσι ώστε να λειτουργούν σε διαφορετικά χρονικά διαστήματα, τα οποία ακολουθούν μία συγκεκριμένη σειρά, δίνοντας την εντύπωση στον χρήστη ότι συνεργάζονται με τις πραγματικές μηχανές. Η τελική εφαρμογή προσφέρει τη δυνατότητα επιλογής μεταξύ διαφορετικών ρομπότ και διαφορετικών διατάξεων για προσομοίωση.

3. Η ΜΗΧΑΝΗ ΓΡΑΦΙΚΩΝ UNITY

Η εφαρμογή της παρούσας εργασίας υλοποιήθηκε με τη χρήση της μηχανής τρισδιάστατων γραφικών Unity. Η μηχανή αυτή έχει ένα τεράστιο εύρος δυνατοτήτων και χρησιμοποιείται συχνά για την ανάπτυξη παιχνιδιών απευθυνόμενα σε κάθε είδους πλατφόρμα, αλλά και εφαρμογών του βιομηχανικού ή εκπαιδευτικού τομέα. Αυτή η μηχανή γραφικών θεωρήθηκε ιδανική, καθώς παρέχει μία πληθώρα χρήσιμων εργαλείων ανάπτυξης, ένα εύχρηστο περιβάλλον εργασίας, μία μεγάλη και ενεργή κοινότητα χρηστών καθώς και εύκολη πρόσβαση σε γεωμετρικά στοιχεία των μοντέλων, η οποία είναι υψίστης σημασίας για την υλοποίηση της συγκεκριμένης εφαρμογής. Στο κεφάλαιο αυτό θα περιγραφεί το περιβάλλον της μηχανής αυτής και οι βασικές λειτουργίες που παρέχει, δίνοντας έμφαση στις πιο σημαντικές από αυτές για την υλοποίηση της παρούσας εφαρμογής.

3.1 Δομή ενός πρότζεκτ στη Unity

Μία εφαρμογή που αναπτύσσεται στο περιβάλλον της Unity ονομάζεται πρότζεκτ (**Project**). Ένα πρότζεκτ αποτελείται από μία ή περισσότερες σκηνές (**Scenes**), οι οποίες περιέχουν ένα ή περισσότερα αντικείμενα γνωστά ως **GameObjects.** Τα αντικείμενα βρίσκονται σε μία ιεραρχία στο πρότζεκτ και οποιοδήποτε από αυτά μπορεί να οριστεί ως γονέας κάποιου άλλου. Η κατάλληλη τοποθέτηση των αντικειμένων στη σκηνή δημιουργεί το γραφικό περιβάλλον της εφαρμογής που αναπτύσσεται. Κάθε αντικείμενο αποτελείται από ένα ή περισσότερα στοιχεία (**Components**) που βρίσκονται στον **Inspector** του αντικειμένου. Τα στοιχεία αυτά καθορίζουν τις ιδιότητες του αντικειμένου, την αλληλεπίδρασή του με το περιβάλλον της εφαρμογής και τη συμπεριφορά που θα έχει κατά τη διάρκεια εκτέλεσής της. Όλα τα αρχεία που μπορεί να χρειαστούν για την υλοποίηση του πρότζεκτ (π.χ. αρχεία ήχου, εικόνες, αρχεία τρισδιάστατων μοντέλων, κ.λπ.) αποθηκεύονται στον φάκελο **Assets.** Στο σχήμα 3.1 παρουσιάζεται το περιβάλλον της μηχανής γραφικών κατά τη δημιουργία ενός νέου πρότζεκτ.

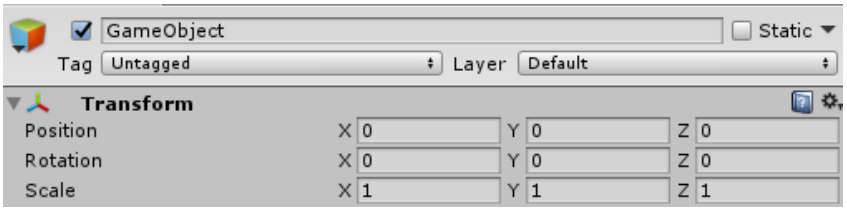


Σχήμα 3.1: Το περιβάλλον ανάπτυξης της Unity

3.2 Στοιχεία αντικειμένων (Components)

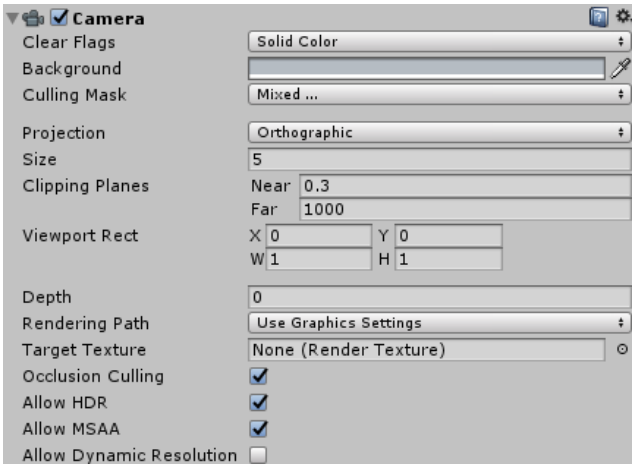
Στην ενότητα αυτή θα αναφερθούν μερικά από τα πιο βασικά στοιχεία που μπορεί να περιέχονται σε κάποιο αντικείμενο. Τα στοιχεία αυτά εμφανίζονται στον Inspector στο περιβάλλον ανάπτυξης κατά την επιλογή ενός αντικειμένου.

3.2.1 Transform



Το πιο βασικό στοιχείο κάθε αντικειμένου είναι το στοιχείο Transform. Δημιουργείται αυτόματα κατά τη δημιουργία ενός νέου αντικειμένου και βρίσκεται υποχρεωτικά σε αυτό. Το στοιχείο αυτό καθορίζει τη θέση του αντικειμένου, την περιστροφή του καθώς και τις διαστάσεις του ως προς τους άξονες x, y και z. Σημειώνεται ότι, εάν το αντικείμενο έχει κάποιο γονέα, τότε οι τιμές του στοιχείου Transform υπολογίζονται σε σχέση με τις αντίστοιχες τιμές του γονέα του.

3.2.2 Camera

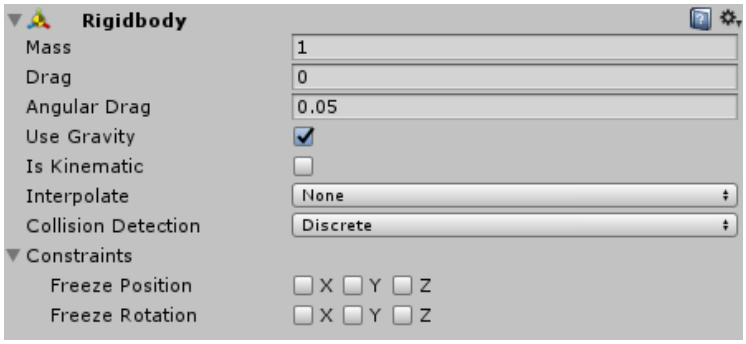


Ως κάμερα ορίζεται η συσκευή που προβάλει στο χρήστη το περιβάλλον της σκηνής. Το στοιχείο της κάμερας τοποθετείται στο αντικείμενο για να αποκτήσει αυτή την ιδιότητα. Το στοιχείο αυτό προσφέρει ένα αριθμό από παραμέτρους τις οποίες ο χρήστης μεταβάλλει ανάλογα με τις ανάγκες της εφαρμογής που αναπτύσσει. Σε μία σκηνή μπορεί να υπάρχει απεριόριστος αριθμός από κάμερες. Κάθε στιγμή, μόνο μία θέαση μπορεί να είναι ορατή κατά την εκτέλεση της εφαρμογής. Η κατάλληλη διαχείριση και εναλλαγή καμερών με την ενεργοποίηση και απενεργοποίησή τους, δίνει την αίσθηση μίας πιο πλούσιας εφαρμογής και προσφέρει μία μοναδική εμπειρία στο χρήστη.

3.2.3 Mesh

Η προβολή τρισδιάστατων μοντέλων στη σκηνή της Unity γίνεται μέσω δικτύων γεωμετρίας (**Meshes**). Αυτά σχεδιάζονται συνήθως με τη βοήθεια ειδικών προγραμμάτων μοντελοποίησης, όπως τα 3ds Max, Blender, Autodesk Inventor ή Cinema 4D. Η Unity υποστηρίζει συγκεκριμένους τύπους αρχείων γεωμετρίας, οι οποίοι είναι .obj, .fbx, .max, .blend, .dae και .dxf. Κατά τη δημιουργία ενός τρισδιάστατου αντικειμένου στη σκηνή της Unity, δημιουργούνται αυτόματα σε αυτό τα στοιχεία **MeshFilter** και **MeshRenderer**. Το πρώτο από αυτά είναι υπεύθυνο για το σχηματισμό της γεωμετρίας του μοντέλου, ενώ το δεύτερο για την προβολή του στη σκηνή και τις ιδιότητες εμφάνισής του (φωτισμός, αντανακλάσεις, σκιές, χρώμα, κ.λπ.).

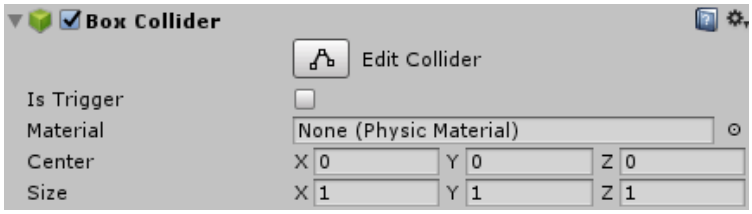
3.2.4 Rigidbody



Η εισαγωγή του στοιχείου Rigidbody σε ένα αντικείμενο, αυτόματα ορίζει το αντικείμενο αυτό υπό την επίδραση της μηχανής φυσικής που προσφέρει η Unity. Η μηχανή φυσικής είναι ένα λογισμικό που παρέχει μία προσεγγιστική αναπαράσταση των νόμων φυσικής, δημιουρ-

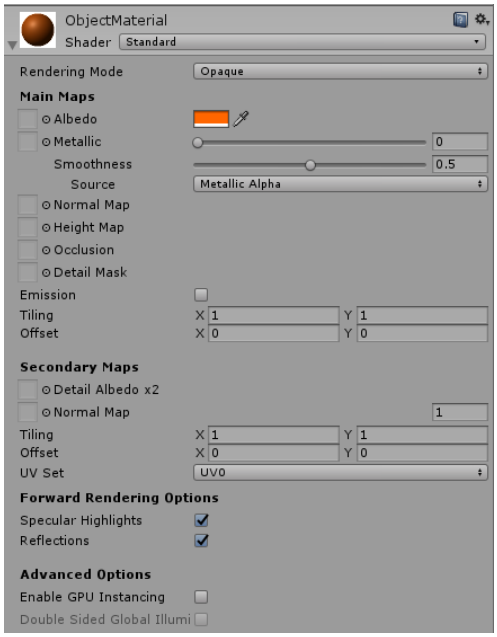
γώντας έτσι ρεαλιστικές συμπεριφορές κατά την αλληλεπίδραση αντικειμένων του περιβάλλοντος. Αυτό σημαίνει ότι το αντικείμενο βρίσκεται αρχικά υπό την επίδραση της βαρύτητας και ότι ανταποκρίνεται σε συγκρούσεις με άλλα αντικείμενα. Αλλαγή των παραμέτρων που προσφέρει το στοιχείο Rigidbody, έχει ως αποτέλεσμα διαφορετική συμπεριφορά η οποία τροποποιείται σύμφωνα με τις ανάγκες του χρήστη. Στην εφαρμογή της παρούσας εργασίας η χρήση του στοιχείου αυτού δεν ήταν απαραίτητη καθώς δεν προβλέπεται η δημιουργία ενός ρεαλιστικού περιβάλλοντος που ανταποκρίνεται σε νόμους φυσικής.

3.2.5 Collider



Το στοιχείο **Collider** καθορίζει τα όρια του αντικειμένου για την ανίχνευση συγκρούσεων. Ένας collider είναι αόρατος και δεν έχει κάποια σχέση με το δίκτυο γεωμετρίας του αντικειμένου. Υπάρχουν διάφορα είδη από colliders, μερικά από τα οποία είναι οι **Box Collider**, **Sphere Collider** και **Capsule Collider**. Η χρήση επιπλέον ορίων για την ανίχνευση συγκρούσεων συμβαίνει επειδή συνήθως η γεωμετρία των αντικειμένων είναι αρκετά πολύπλοκη, γεγονός το οποίο θα οδηγούσε σε μεγάλη σπατάλη πόρων. Στις περισσότερες περιπτώσεις, η απλή γεωμετρία των στοιχείων collider είναι αρκετή για να προσεγγίζει ικανοποιητικά την πραγματική γεωμετρία του αντικειμένου. Σε αντίθετες περιπτώσεις χρησιμοποιείται το στοιχείο **Mesh Collider**, το οποίο αντιστοιχεί ακριβώς στη πραγματική γεωμετρία του μοντέλου. Η περιορισμένη χρήση του στοιχείου Mesh Collider αρκετές φορές επιφέρει καλύτερα αποτελέσματα χωρίς να δημιουργεί προβλήματα έλλειψης υπολογιστικών πόρων. Σημειώνεται ότι είναι δυνατή η χρήση περισσότερων του ενός Collider σε ένα αντικείμενο, σε περιπτώσεις όπου η χρήση του ενός δεν δημιουργεί την επιθυμητή γεωμετρία.

3.2.6 Materials - Shaders



Η εμφάνιση ενός αντικειμένου στη σκηνή εξαρτάται σε μεγάλο βαθμό από τα στοιχεία **Material** και **Shader** που βρίσκονται σε αυτό. Τα στοιχεία αυτά λειτουργούν σε συνδυασμό, δηλαδή το στοιχείο Material είναι υπεύθυνο για το χρώμα και την υφή που θα έχει ένα αντικείμενο, ενώ το στοιχείο Shader περιέχει μαθηματικούς υπολογισμούς που αφορούν τον υπολογισμό του χρώματος σε κάθε pixel του αντικειμένου, βασισμένο στον εκάστοτε φωτισμό της σκηνής και στις ρυθμίσεις του στοιχείου Material. Κατά την εισαγωγή ενός στοιχείου Material ορίζεται ποιο Shader θα χρησιμοποιηθεί. Συνεπώς, αναλόγως την επιλογή του Shader, οι ρυθμίσεις για το στοιχείο Material είναι διαφορετικές. Για τα περισσότερα αντικείμενα της σκηνής, το Shader που χρησιμοποιείται είναι το **Standard Shader.** Το Shader αυτό παρέχει μία πληθώρα ρυθμίσεων το οποίο το καθιστά κατάλληλο για τη ρεαλιστική απεικόνιση των περισσότερων αντικειμένων. Για εξειδικευμένες περιπτώσεις χρησιμοποιούνται διαφορετικά Shaders τα οποία προσδίδουν ιδιαίτερα εφέ στο αντικείμενο. Για παράδειγμα, στην παρούσα ε-

φαρμογή χρησιμοποιήθηκε το Shader **Silhouete-Outlined Diffuse** [12] για τον τονισμό του περιγράμματος ενός αντικειμένου κατά την επιλογή του.

3.2.7 Scripts

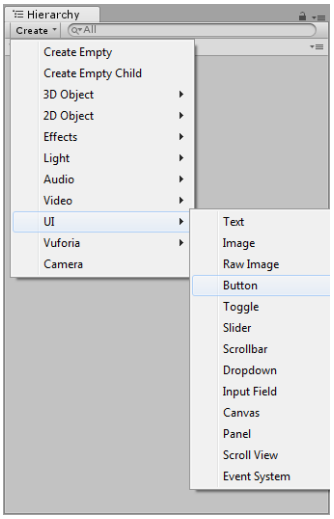
Το πιο βασικό στοιχείο το οποίο καθορίζει τη συμπεριφορά ενός αντικειμένου, είναι το στοιχείο Script. Τα scripts αποτελούν ουσιαστικά κλάσεις στις οποίες περιέχονται κομμάτια κώδικα τα οποία συνήθως συντάσσει ο χρήστης. Οποιαδήποτε συμπεριφορά και έλεγχος αντικειμένου ή στοιχείου, μπορεί να υλοποιηθεί μέσω κώδικα. Οι υποστηριζόμενες γλώσσες στο περιβάλλον της Unity είναι η **C#** και η **Javascript**. Η βασική σύνδεση ενός script με τη Unity γίνεται μέσω της κλάσης **MonoBehaviour**. Για κάθε κλάση – script που δημιουργείται, δηλώνεται αυτόματα ότι η κλάση αυτή κληρονομεί τη MonoBehaviour. Αυτό έχει ως αποτέλεσμα την αυτόματη δημιουργία των συναρτήσεων **Start()** και **Update()** σε κάποιο script.

Η συνάρτηση Start() χρησιμοποιείται για την αρχικοποίηση διαφόρων μεταβλητών στο script. Η κλήση της συνάρτησης αυτής γίνεται μόνο μία φορά κατά την εκτέλεση της εφαρμογής και συγκεκριμένα όταν το script ενεργοποιείται για πρώτη φορά. Η Unity εξασφαλίζει ότι όλες οι συναρτήσεις Start() θα έχουν κληθεί πριν από οποιαδήποτε εκτέλεση κάποιας συνάρτησης Update().

Η συνάρτηση Update() καλείται σε κάθε frame της εφαρμογής – παιχνιδιού. Ο αριθμός των frames ανά δευτερόλεπτο σε μία εφαρμογή ονομάζεται **Frame Rate** ή **Frames Per Second** (**FPS**). Ως FPS ορίζεται ο αριθμός των εικόνων που προβάλλονται στην οθόνη κάθε δευτερόλεπτο. Επομένως, η συνάρτηση Update() χρησιμοποιείται για την υλοποίηση συμπεριφορών στις οποίες είναι απαραίτητος ο συνεχής έλεγχος κάποιων καταστάσεων. Για παράδειγμα, η μετακίνηση ενός παίκτη με τη χρήση εισόδου από το πληκτρολόγιο, απαιτεί συνεχώς να ελέγχεται αν κάποιο από τα κουμπιά μετακίνησης έχει πατηθεί. Εάν ο έλεγχος αυτός γίνεται στη συνάρτηση Update(), το αποτέλεσμα θα είναι μία ομαλή εκτέλεση του παιχνιδιού, όπως είναι επιθυμητό.

Η κλάση MonoBehaviour προσφέρει επιπλέον κάποιες σημαντικές συναρτήσεις που μπορούν να χρησιμοποιηθούν, όμως δεν δηλώνονται αυτόματα κατά τη δημιουργία των script. Μερικές από αυτές είναι οι **Awake()**, **FixedUpdate()**, **LateUpdate()**, **OnGUI()**, **OnEnable()** και **OnDisable()**. Εκτός από τις παραπάνω συναρτήσεις, ο χρήστης μπορεί να δημιουργήσει τις δικές του, τις οποίες θα πρέπει να καλέσει ξεχωριστά.

3.3 Γραφική διεπαφή χρήστη



Η δημιουργία μίας γραφικής διεπαφής χρήστης (**Graphical User Interface**) μπορεί πολύ εύκολα να υλοποιηθεί μέσω των αντικειμένων τύπου UI που προσφέρει η Unity. Τα αντικείμενα αυτά βρίσκονται στην ιεραρχία αντικειμένων ως παιδιά ενός αντικειμένου στο οποίο βρίσκεται το στοιχείο Canvas. Τα περισσότερα αντικείμενα γραφικής διεπαφής δημιουργούνται εύκολα μέσω της επιλογής Create → UI στην ιεραρχία αντικειμένων. Η αναδιάταξη και προσαρμογή των αντικειμένων αυτών στα όρια της οθόνης, δημιουργεί ένα εύχρηστο περιβάλλον μέσω του οποίου ο χρήστης χειρίζεται την εφαρμογή ή δέχεται ορισμένες σημαντικές πληροφορίες για αυτήν στην πορεία εκτέλεσής της. Σημειώνεται ότι η δημιουργία αντικειμένων γραφικής διεπαφής και ο χειρισμός γεγονότων που έχουν να κά-

νουν με αυτήν, μπορούν να υλοποιηθούν και μέσω κώδικα με τη βοήθεια της συνάρτησης OnGUI().

3.4 Prefabs

Όπως έχει ήδη αναφερθεί κάθε αντικείμενο στη σκηνή μπορεί να αποτελείται από ένα ή περισσότερα στοιχεία τα οποία έχουν ρυθμιστεί σύμφωνα με τις ανάγκες της εφαρμογής. Σε πολλές περιπτώσεις υπάρχει η ανάγκη δημιουργίας ενός αντικειμένου πολλές φορές σε μία σκηνή. Η πιο προφανής λύση θα ήταν η απλή αντιγραφή του αντικειμένου πολλαπλές φορές σε μία σκηνή. Το πρόβλημα στην περίπτωση αυτή είναι ότι όλα τα αντίγραφα θα ήταν ανεξάρτητα μεταξύ τους, οπότε μία αλλαγή σε κάποιο από αυτά δεν θα εφαρμοζόταν και σε όλα τα αντίγραφά του. Για το λόγο αυτό η Unity προσφέρει τον τύπο **Prefab,** ο οποίος επιτρέπει την αποθήκευση ενός GameObject μαζί με όλα τα στοιχεία του στον φάκελο αρχείων του πρότζεκτ. Έτσι, οποιαδήποτε αλλαγή συμβαίνει σε ένα αντικείμενο που έχει αποθηκευτεί ως prefab, εφαρμόζεται και σε όλα τα αντίγραφά του.

4. Η ΕΠΑΥΞΗΜΕΝΗ ΠΡΑΓΜΑΤΙΚΟΤΗΤΑ ΣΤΗΝ ΕΦΑΡΜΟΓΗ

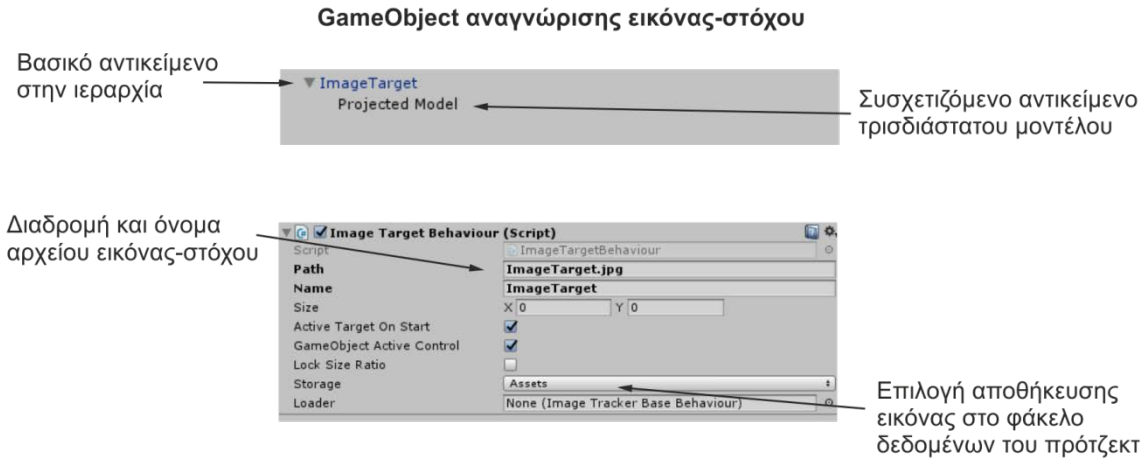
Ένα μικρό αλλά ιδιαίτερα σημαντικό κομμάτι της εφαρμογής αποτελεί η σχέση της με την τεχνολογία της επαυξημένης πραγματικότητας. Η εισαγωγή οποιουδήποτε μοντέλου στο κύριο περιβάλλον της εφαρμογής γίνεται μέσω εύρεσης και επιλογής της προβολής αυτού από μία εικόνα-στόχο (**Marker**) που το αντιπροσωπεύει. Συγκεκριμένα ο χρήστης μέσω της κάμερας του κινητού ή του τάμπλετ βρίσκει την εικόνα-στόχο που επιθυμεί και στη συνέχεια προβάλλεται πάνω από αυτήν το τρισδιάστατο μοντέλο που της αντιστοιχεί. Οι εικόνες-στόχοι βρίσκονται όλες στην 3η έκδοση του βιβλίου Μηχανολογικό Σχέδιο του Καθηγητή Αριστομένη Αντωνιάδη της Σχολής Μηχανικών Παραγωγής και Διοίκησης του Πολυτεχνείου Κρήτης. Στο κεφάλαιο αυτό θα εξηγηθεί ο τρόπος αναγνώρισης, επιλογής και εισαγωγής των μοντέλων στο κύριο περιβάλλον της εφαρμογής.

4.1 Αναγνώριση και προβολή μοντέλου

Η ενσωμάτωση της τεχνολογίας επαυξημένης πραγματικότητας στο περιβάλλον της Unity γίνεται, όπως έχει ήδη αναφερθεί, με τη χρήση της βιβλιοθήκης EasyAR. Η βιβλιοθήκη αυτή παρέχει έναν εύκολο τρόπο αναγνώρισης εικόνων-στόχων και υπέρθεσης τρισδιάστατων μοντέλων. Συγκεκριμένα παρέχει την υλοποίηση διαφόρων λειτουργιών επαυξημένης πραγματικότητας μέσω μίας συλλογής από Prefabs. Για τους σκοπούς της παρούσας εφαρμογής χρησιμοποιήθηκαν τα Prefabs **EasyAR_Startup** και **ImageTarget**.

Το prefab **EasyAR_Startup** περιέχει όλη τη βασική λειτουργικότητα που απαιτείται σε μία εφαρμογή επαυξημένης πραγματικότητας. Δηλαδή την αρχικοποίηση της κάμερας και τη δυνατότητα αναγνώρισης στόχων (**Targets**) και την αποτελεσματική παρακολούθηση αυτών (**Tracking**). Επιπλέον δίνεται η δυνατότητα αλλαγής κάποιων ρυθμίσεων παρακολούθησης, όπως η προτίμηση καλύτερης ποιότητας έναντι απόδοσης (**Quality over Performance**) ή η επιλογή του αριθμού ταυτόχρονων στόχων που προβάλλονται στη σκηνή (**Simultaneuous Targets**). Για την ενεργοποίηση λειτουργίας των δυνατοτήτων αυτών απαιτείται η δημιουργία ενός λογαριασμού στην ιστοσελίδα της βιβλιοθήκης (https://www.easyar.com/) και στη συνέχεια η σύνδεση ενός μοναδικού κλειδιού πιστοποίησης με το project στη Unity.

Η προσθήκη ενός prefab **ImageTarget** στη σκηνή αφορά τη δημιουργία μίας νέας εικόνας-στόχου και τη σύνδεσή της με ένα τρισδιάστατο αντικείμενο. Στο σχήμα 4.1 παρουσιάζονται οι βασικές ρυθμίσεις για ένα ImageTarget.



Σχήμα 4.1: Ρυθμίσεις ενός αντικείμενου ImageTarget

Στο σχήμα 4.1 παρατηρείται ότι η διαδρομή του αρχείου εικόνας-στόχου αποτελείται μόνο από το όνομα του αρχείου. Αυτό συμβαίνει επειδή η επιλογή Assets ως σημείο αποθήκευσης, ορίζει ως προκαθορισμένη θέση των αρχείων το φάκελο **StreamingAssets** που βρίσκεται στο φάκελο δεδομένων (Assets) του πρότζεκτ. Κατά τη διάρκεια εκτέλεσης της εφαρμογής, το αντικείμενο είναι αρχικά απενεργοποιημένο στη σκηνή της Unity. Η ενεργοποίησή του συμβαίνει μόλις αναγνωριστεί μέσω της κάμερας η εικόνα-στόχος που έχει οριστεί σε αυτό, προκαλώντας την ταυτόχρονη υπέρθεση του συσχετιζόμενου τρισδιάστατου μοντέλου. Σημειώνεται ότι το τρισδιάστατο μοντέλο που προβάλλεται κατά την αναγνώριση της εικόνας που έχει οριστεί, ορίζεται ως παιδί του αντικειμένου ImageTarget. Αυτό καθιστά τη λειτουργία αναγνώρισης ιδιαίτερα απλή, όμως παρουσιάζεται ένα σημαντικό πρόβλημα το οποίο περιγράφεται παρακάτω.

Η εφαρμογή που υλοποιήθηκε αποτελείται από ένα μεγάλο αριθμό αντικειμένων ImageTargets και κατά συνέπεια από ένα μεγάλο αριθμό τρισδιάστατων μοντέλων συσχετιζόμενα με αυτά. Ο μεγάλος όγκος τρισδιάστατων μοντέλων έχει ως αποτέλεσμα τη σημαντική αύξηση του μεγέθους της εφαρμογής, κάτι το οποίο δεν είναι επιθυμητό σε μία εφαρμογή που προορίζεται για φορητές συσκευές. Για το λόγο αυτό, τα τρισδιάστατα μοντέλα αποθηκεύονται online υπό μορφή .obj και συγκεκριμένα στην δα http://www.antoniadis.gr/. Αυτό σημαίνει ότι στην αρχή της εκτέλεσης της εφαρμογής, κανένα αντικείμενο ImageTarget δεν έχει κάποιο παιδί – τρισδιάστατο μοντέλο συσχετιζόμενο με αυτό. Συνεπώς κάθε μοντέλο θα πρέπει να εισάγεται κατά τη διάρκεια εκτέλεσης της εφαρμογής όταν αυτό είναι απαραίτητο και να ορίζεται ως παιδί του αντίστοιχου ImageTarget στο οποίο ανήκει. Η διαδικασία αυτή περιγράφεται στη συνέχεια.

Όπως αναφέρθηκε νωρίτερα, τα ImageTargets είναι αρχικά απενεργοποιημένα και ενεργοποιούνται μόνο κατά την αναγνώριση της εικόνας-στόχου τους με αποτέλεσμα την προβολή του τρισδιάστατου μοντέλου τους. Στη συγκεκριμένη περίπτωση το αντίστοιχο μοντέλο δεν υπάρχει. Επομένως κατά την ενεργοποίηση κάποιου ImageTarget στέλνεται ένα αίτημα λήψης του αρχείου της γεωμετρίας αντίστοιχου μοντέλου (αρχείο .obj) από ένα online url, όπως παρουσιάζεται στον κώδικα του πίνακα 4.1.

Αίτημα λήψης αρχείου τρισδιάστατου μοντέλου από ένα ImageTarget

```
    void Update(){
        if (wasEnabled && !objectCreated && gameControl.CheckInternetAvailability() ) {
            gameControl.DownloadRequest("http://www.antoniadis.gr/models/"+this.name+".obj",this.
gameObject,this.name);
            objectCreated = true;
            Destroy (this);
        }
    }
```

Πίνακας 4.1: Κώδικας δημιουργίας αιτήματος λήψης αρχείου

Ο κώδικας του πίνακα 4.1 τοποθετείται σε κάθε αντικείμενο ImageTarget μέσω αντίστοιχου script. Το κάθε αντικείμενο από αυτά έχει διαφορετική ονομασία και συγκεκριμένα ένα όνομα-κωδικό του μοντέλου στο οποίο αντιστοιχεί. Με τη λογική αυτή, το αιτούμενο αρχείο προς λήψη αντιστοιχεί στο όνομα-κωδικό του μοντέλου, αποκλείοντας έτσι τη λήψη λανθασμένου αρχείου. Για να πραγματοποιηθεί επιτυχώς ένα αίτημα, απαιτείται να υπάρχει σύνδεση στο διαδίκτυο. Σε αντίθετη περίπτωση προβάλλεται αντίστοιχο μήνυμα σφάλματος στην οθόνη της εφαρμογής. Εφόσον ένα αίτημα πραγματοποιηθεί, ο κώδικας του πί-

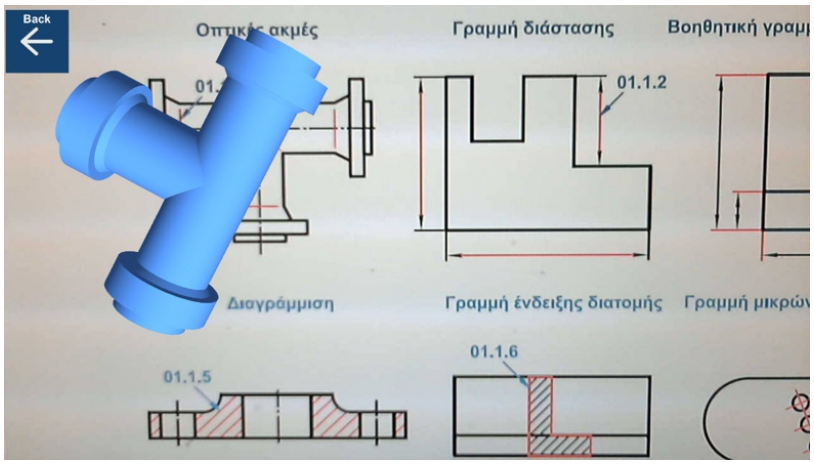
νακα 4.1 αφαιρείται από το αντίστοιχο ImageTarget, έτσι ώστε να αυξηθεί η απόδοση της εφαρμογής μειώνοντας, την εκτέλεση τμημάτων κώδικα που βρίσκονται στη συνάρτηση Update(). Η λήψη του αρχείου πραγματοποιείται μέσω της κλάσης **UnityWebRequest**. Κατά τη διάρκεια λήψης του μοντέλου προβάλλεται ταυτόχρονα μία μπάρα προόδου η οποία κρατά ενήμερο το χρήστη. Η εκτέλεση του αιτήματος και η ενημέρωση της μπάρας προόδου παρουσιάζονται στον κώδικα του πίνακα 4.2.

Εκτέλεση αιτήματος και πρόοδος λήψης

```
UnityWebRequest www = UnityWebRequest.Get(url)
UnityWebRequestAsyncOperation asyncOperation = www.SendWebRequest ();
float prevProgress = -1;
while (!asyncOperation.isDone) {
    if (asyncOperation.progress > prevProgress) {
        prevProgress = asyncOperation.progress;
        progressBar.UpdateProgress (asyncOperation.progress);
    }
}
```

Πίνακας 4.2: Κώδικας εκτέλεσης αιτήματος και ενημέρωση προόδου λήψης

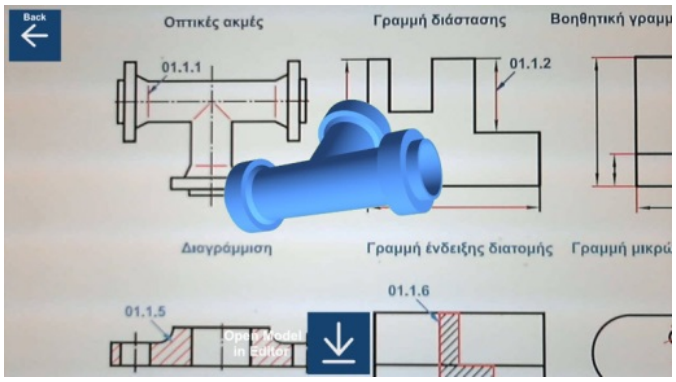
Εφόσον η λήψη του αρχείου γεωμετρίας του μοντέλου είναι επιτυχής, το επόμενο βήμα είναι η εισαγωγή του στο περιβάλλον της Unity και η αναγνώρισή του από αυτό. Η γεωμετρία τρισδιάστατων μοντέλων στο περιβάλλον της Unity αναπαριστάται μέσω της κλάσης **Mesh**. Συνεπώς χρησιμοποιήθηκε ένας ObjImporter [10], ο οποίος διαβάζει ένα αρχείο .obj και επιστρέφει ένα αντικείμενο κλάσης Mesh. Στη συνέχεια δημιουργείται ένα GameObject που αναπαριστά το μοντέλο και αποτελείται από τη γεωμετρία του αντικειμένου της κλάσης Mesh. Το GameObject ορίζεται ως παιδί του ImageTarget που ενεργοποιήθηκε και κατά συνέπεια προβάλλεται πάνω από την εικόνα-στόχο που του αντιστοιχεί. Στο σχήμα 4.2 παρουσιάζεται ένα στιγμιότυπο της εφαρμογής κατά την αναγνώριση και προβολή ενός μοντέλου από μία εικόνα-στόχο.



Σχήμα 4.2: Προβολή τρισδιάστατου μοντέλου πάνω από την αντίστοιχη εικόνα-στόχο

4.2 Επιλογή και μεταφορά μοντέλου στο κύριο περιβάλλον

Μετακινώντας την κάμερα πάνω από μία εικόνα-στόχο που βρίσκεται στον πραγματικό κόσμο, ο χρήστης έχει πλέον τη δυνατότητα να δει το τρισδιάστατο μοντέλο που προβάλλεται πάνω από αυτήν. Επιπρόσθετα, εφόσον ο χρήστης αποφασίσει ποιο μοντέλο τον ενδιαφέρει, μπορεί να το επιλέξει και στη συνέχεια να το μεταφέρει σε ένα νέο περιβάλλον για περαιτέρω επεξεργασία. Όταν ένα αντικείμενο επιλεχθεί, μεταφέρεται στο κέντρο της κάμερας με μία διακριτική γραμμή στο περίγραμμά του, η οποία και δηλώνει πως αυτό έχει επιλεχθεί. Η επιλογή γίνεται μέσω αφής στο σημείο που βρίσκεται το μοντέλο, εφόσον κάποιο μοντέλο προβάλλεται στην οθόνη. Στο σχήμα 4.3 παρουσιάζεται ένα στιγμιότυπο της εφαρμογής μετά από επιλογή κάποιου μοντέλου.



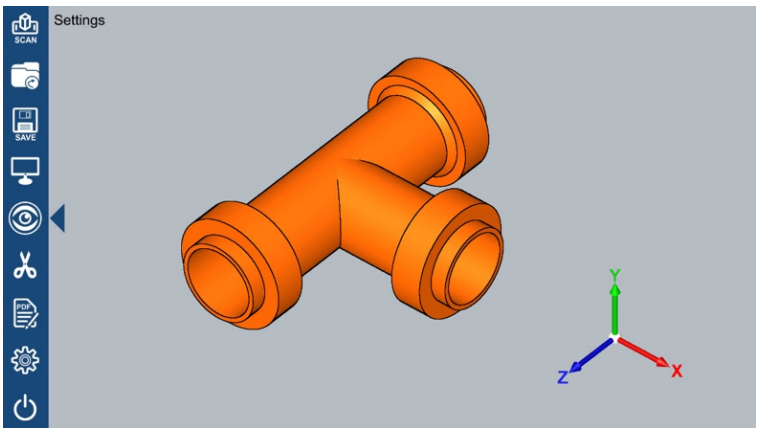
Σχήμα 4.3: Προβολή τρισδιάστατου μοντέλου πάνω από την αντίστοιχη εικόνα-στόχο

Στο στιγμιότυπο του σχήματος 4.3 παρουσιάζονται επιπλέον δύο κουμπιά στην οθόνη. Το κουμπί στο κέντρο της μεταφέρει το επιλεγμένο μοντέλο στο κύριο περιβάλλον της εφαρμογής, ενώ το κουμπί που βρίσκεται πάνω αριστερά επιστρέφει στο κύριο περιβάλλον χωρίς τη μεταφορά κάποιου μοντέλου σε αυτό. Σημειώνεται, ότι το περιβάλλον της κάμερας για την επιλογή και προβολή μοντέλων και το κύριο περιβάλλον της εφαρμογής αποτελούν δύο διαφορετικές σκηνές στο πρότζεκτ της Unity. Επομένως η μεταφορά του μοντέλου απαιτεί τη μεταφορά ενός αντικειμένου από μία σκηνή σε μία άλλη. Η διαδικασία αυτή παρουσιάζεται στον κώδικα του πίνακα 4.3. Το κύριο περιβάλλον μετά την επιτυχή μεταφορά ενός μοντέλου σε αυτό παρουσιάζεται στο σχήμα 4.4. Οι λειτουργίες που παρέχονται θα εξηγηθούν αναλυτικά στο κεφάλαιο 7.

Φόρτωση σκηνής και μεταφορά μοντέλου σε αυτήν

```
IEnumerator LoadEditorAsync(GameObject objectToMove)
{
    AsyncOperation asyncLoad = SceneManager.LoadSceneAsync (1,LoadSceneMode.Additive);

    while (!asyncLoad.isDone)
    {
        yield return null;
    }
    SceneManager.MoveGameObjectToScene (objectToMove, SceneManager.GetSceneByBuildIndex (1));
    SceneManager.UnloadSceneAsync(SceneManager.GetActiveScene());
}
```

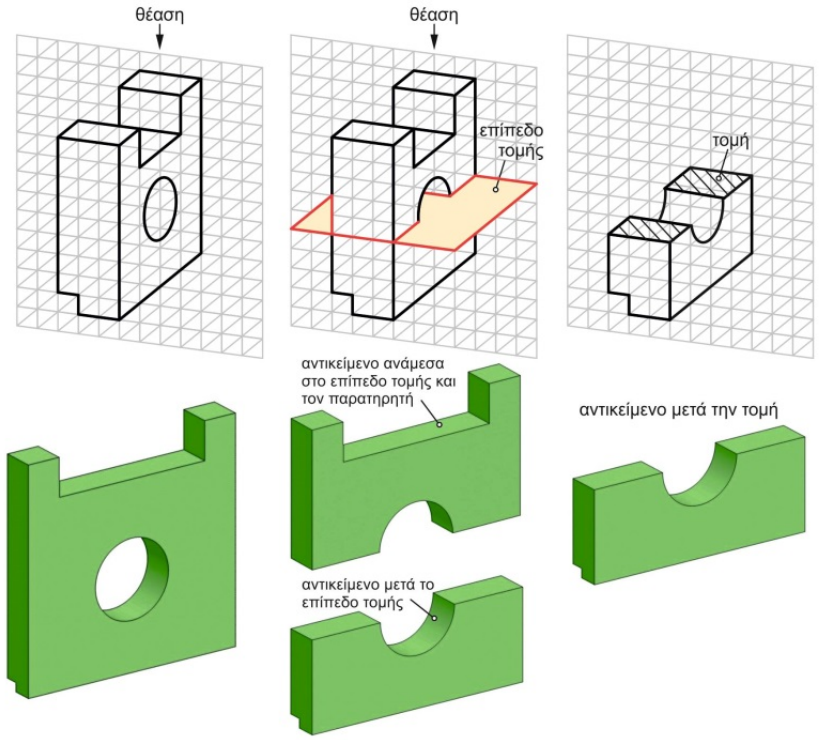
Πίνακας 4.3: Κώδικας φόρτωσης σκηνής και μεταφοράς μοντέλου σε αυτή



Σχήμα 4.4: Το κύριο περιβάλλον της εφαρμογής

5. ΤΟΜΗ ΣΤΕΡΕΟΥ ΑΠΟ ΕΝΑ ΕΠΙΠΕΔΟ

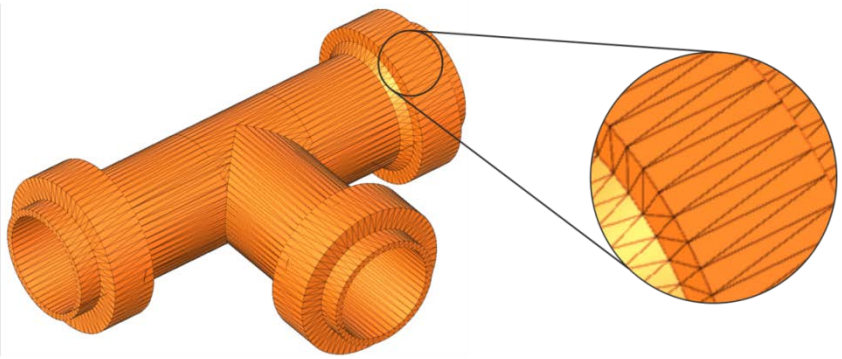
Η πιο σημαντική λειτουργία της εφαρμογής αφορά την τομή ενός στερεού από ένα επίπεδο στο χώρο [6,7,8,9]. Η λειτουργία αυτή χρησιμοποιείται πολύ συχνά σε CAD εφαρμογές και αρκετές φορές είναι επιθυμητές δύο ή περισσότερες διαδοχικές τομές ενός στερεού. Η διαδικασία της τομής ενός τυχαίου στερεού από ένα αντίστοιχα τυχαίο επίπεδο παρουσιάζεται στο σχήμα 5.1. Στο κεφάλαιο αυτό περιγράφεται αναλυτικά η διαδικασία που ακολουθείται για να επιτευχθεί το συγκεκριμένο αποτέλεσμα μέσω μίας σειράς αλγορίθμων υπολογιστικής γεωμετρίας.



Σχήμα 5.1: Τομή ενός στερεού σε σχέση με ένα επίπεδο

5.1 Δομή 3D Μοντέλων

Η αναπαράσταση τρισδιάστατων μοντέλων στα γραφικά υπολογιστών γίνεται μέσω δικτύων γεωμετρίας (Meshes). Τα δίκτυα γεωμετρίας είναι μία συλλογή από κορυφές (Vertices), ακμές (Edges) και επιφάνειες (Faces), οι οποίες ορίζουν το σχήμα ενός τρισδιάστατου μοντέλου ενός στερεού. Οι επιφάνειες αποτελούνται συνήθως από τρίγωνα, τετράπλευρα ή απλά κυρτά πολύγωνα. Στο σχήμα 5.2 παρουσιάζεται ένα τριγωνικό δίκτυο γεωμετρίας (Triangle Mesh).



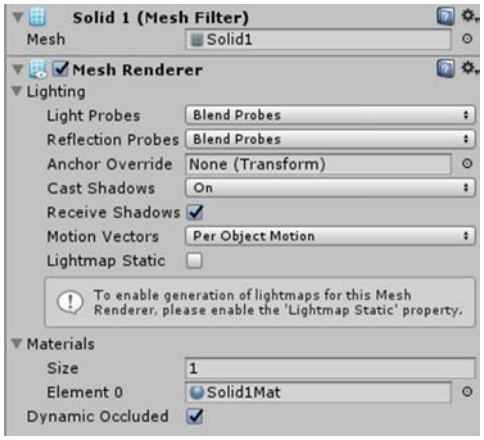
Σχήμα 5.2: Τριγωνικό Δίκτυο Γεωμετρίας που αναπαριστά ένα μοντέλο CAD

Στις περισσότερες περιπτώσεις η γεωμετρία περιορίζεται στην αναπαράσταση της εξωτερικής επιφάνειας του μοντέλου και χωρίς να παρέχεται καμία πληροφορία για το εσωτερικό του. Έτσι, η τομή σε μία τέτοια γεωμετρία επιφανειών έχει ως αποτέλεσμα ανοικτές περιοχές στο επίπεδο τομής, σε αντίθεση με τις επιθυμητές κλειστές πολυγωνικές γραμμές. Στο σχήμα 5.3 παρουσιάζεται η τομή ενός μοντέλου με ένα επίπεδο τομής και οι δύο αντίστοιχες περιπτώσεις αποτελέσματος της τομής, η πρώτη για ένα μοντέλο επιφανειών και η δεύτερη για ένα στερεό μοντέλο. Στην περίπτωση της εφαρμογής που αναπτύχθηκε, τα μοντέλα που εισάγονται είναι μοντέλα επιφανειών αλλά απαιτείται η τομή να προσφέρει κλειστά σχήματα, όπως γίνεται στην τομή στερεών μοντέλων. Κατά συνέπεια είναι απαραίτητο το κλείσιμο των ανοικτών χώρων που δημιουργούνται μετά την τομή του μοντέλου επιφανειών με το επίπεδο τομής.

Σχήμα 5.3: Τομή σε ένα Μοντέλο Επιφανειών και σε ένα Στερεό Μοντέλο

Επισημαίνεται ότι η διαδικασία που χρησιμοποιείται στην εφαρμογή προσομοιώνει την τομή στερεού μοντέλου κλείνοντας τους ανοιχτούς χώρους από την τομή ενός μοντέλου επιφανειών. Δεν πρόκειται δηλαδή για τομή στερεού σώματος με επίπεδο αλλά για ψευδοπαρουσίαση της τομής αυτής, χρησιμοποιώντας τα αποτελέσματα της τομής επιφανειών.

5.1.1 Αναπαράσταση Δικτύων Γεωμετρίας στη Unity



Σχήμα 5.4: Τα στοιχεία Mesh Filter και Mesh Renderer σε μία οντότητα

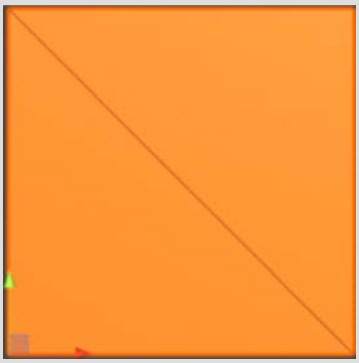
Για την αναπαράσταση ενός αντικειμένου στη σκηνή της Unity, είναι απαραίτητο να υπάρχουν σε αυτό τα στοιχεία **Mesh Filter** και **Mesh Renderer (**βλ. σχήμα 5.4). Η πρόσβαση στο δίκτυο γεωμετρίας ενός μοντέλου γίνεται μέσω της κλάσης Mesh. Το στοιχείο Mesh Filter παίρνει ένα στιγμιότυπο της κλάσης Mesh από τον φάκελο του εκάστοτε project. Το στιγμιότυπο αυτό παρέχεται ουσιαστικά από το μοντέλο που έχει εισαχθεί στον φάκελο αυτό. Ο ρόλος του στοιχείου Mesh Filter είναι να παρέχει τη γεωμετρία σε ένα component τύπου Mesh Renderer, το οποίο και τη σχεδιάζει στη σκηνή.

Η κλάση Mesh περιέχει δύο πίνακες οι οποίοι είναι απαραίτητοι προκειμένου να μπορεί να προσδιοριστεί η γεωμετρία ενός αντικειμένου. Ο πίνακας **Vector3[] vertices** περιέχει τη θέση κάθε κορυφής της γεωμετρίας στον τρισδιάστατο χώρο. Ο πίνακας **int[] triangles** παρέχει πληροφορίες σχετικά με κάθε τρίγωνο της

γεωμετρίας. Συγκεκριμένα, τα τρίγωνα αποθηκεύονται διαδοχικά στον πίνακα και κάθε τρίγωνο καταλαμβάνει τρεις θέσεις, όσες δηλαδή και οι κορυφές του. Κάθε θέση περιέχει έναν ακέραιο ο οποίος συμβολίζει τη θέση της κορυφής στον πίνακα κορυφών vertices. Για παράδειγμα, στον πίνακα 5.1 παρουσιάζεται ένα τμήμα κώδικα που σχεδιάζει τη γεωμετρία ενός απλού τετραγώνου το οποίο αποτελείται από δύο τρίγωνα.

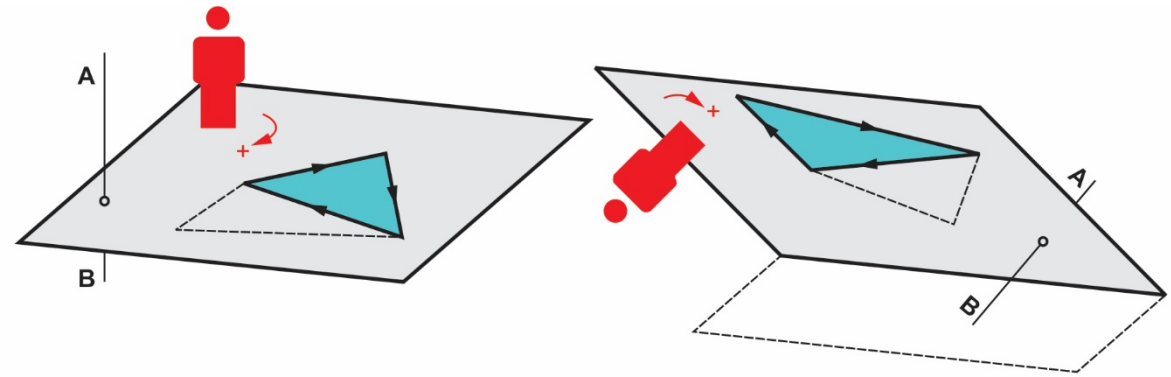
Κώδικας Δημιουργίας Τετραγώνου	Αποτέλεσμα στη σκηνή της Unity

```
void MakeMesh(){
     vertices = new Vector3[] {
         new Vector3 (0, 0, 0),
         new Vector3 (1, 0, 0),
         new Vector3 (0, 1, 0),
         new Vector3 (1, 1, 0),
     };
     triangles = new int[]{ 0, 2, 1, 2, 3, 1};

     mesh.Clear ();
     mesh.vertices = vertices;
     mesh.triangles = triangles;

     mesh.RecalculateNormals ();
}
```



Πίνακας 5.1: Κώδικας και αποτέλεσμα δημιουργίας τετραγώνου στη Unity

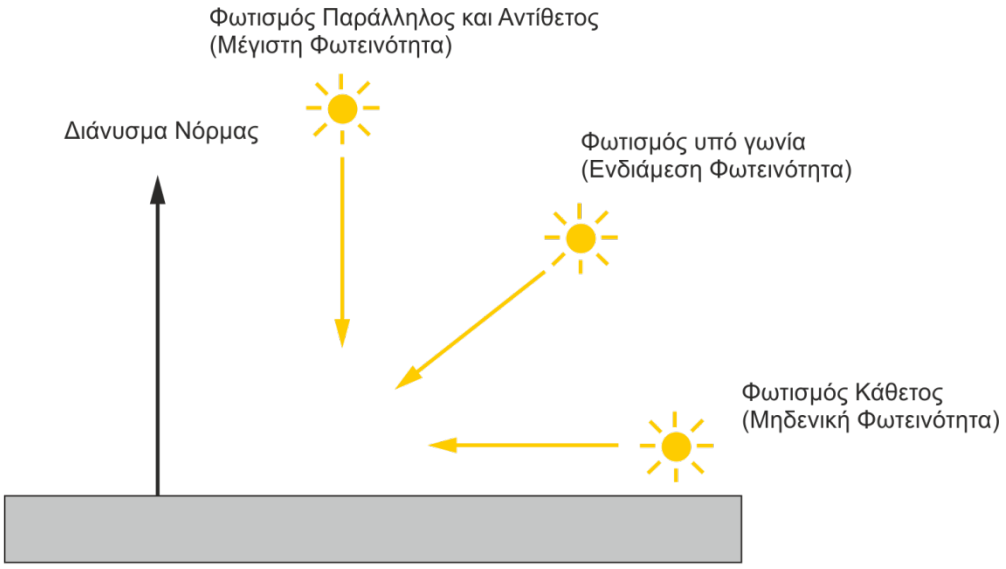
Σημειώνεται ότι τα τρίγωνα σχεδιάζονται με δεξιόστροφη φορά και είναι ορατά από έναν παρατηρητή ο οποίος τα βλέπει από κατάλληλη θέση ώστε να φαίνονται μόνο από αυτήν. Στο σχήμα 5.5 παρουσιάζονται δύο τρίγωνα με αντίθετες φορές και ένας παρατηρητής. Το κάθε τρίγωνο είναι ορατό από τον παρατηρητή μόνο όταν βρίσκεται στον κατάλληλο ημιχώρο. Συνεπώς απαιτείται προσοχή κατά τη δημιουργία του πίνακα triangles, ώστε οι κορυφές να δίνονται με τη σωστή σειρά. Η τεχνική αυτή ονομάζεται **back-face culling** και χρησιμοποιείται για να βελτιώσει σημαντικά την απόδοση κατά τη σχεδίαση αντικειμένων, μειώνοντας τον αριθμό των τριγώνων που πρέπει να σχεδιαστούν. Για να ήταν κάθε τρίγωνο ορατό και από τις δύο όψεις θα έπρεπε να σχεδιαστούν δύο διαφορετικά τρίγωνα για κάθε όψη, διπλασιάζοντας έτσι τον συνολικό αριθμό τριγώνων του αντικειμένου.



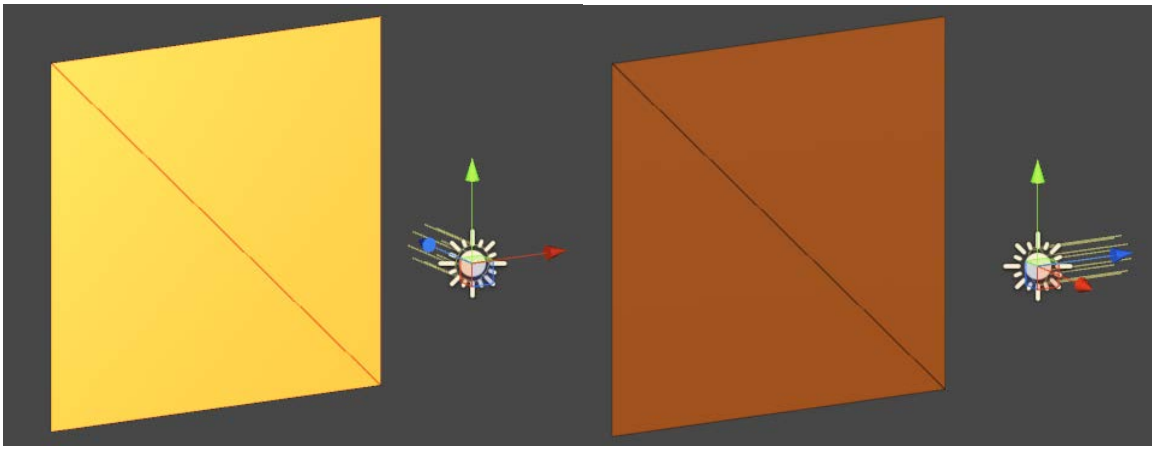
Σχήμα 5.5: Θέαση τριγώνων από παρατηρητή σε δύο ημιχώρους A και B.

Για κάθε κορυφή του πίνακα vertices μπορεί να παρέχονται επιπλέον πληροφορίες που είναι χρήσιμες για τη γεωμετρία. Οι πληροφορίες αυτές αποθηκεύονται σε ξεχωριστούς πίνακες ίσου μεγέθους με τον πίνακα vertices. Οι πίνακες αυτοί υπάρχουν προαιρετικά και μπορεί να είναι οι normals, colors, tangents και uv.

Για τους σκοπούς της παρούσας εργασίας θα χρησιμοποιηθεί επιπλέον μόνο ο πίνακας normals. Ο πίνακας αυτός περιέχει τα διανύσματα νόρμας (normal) για κάθε κορυφή. Ως **νόρμα** ορίζεται ένα διάνυσμα το όποιο έχει κατεύθυνση εξωτερική από την επιφάνεια του μοντέλου και ξεκινά από τη θέση της αντίστοιχης κορυφής στην οποία ανήκει. Το διάνυσμα αυτό καθορίζει την ένταση του φωτός σε κάθε κορυφή και συνεπώς στις επιφάνειες του μοντέλου. Αν η κατεύθυνση του διανύσματος νόρμας μίας κορυφής είναι αντίθετη με αυτήν του φωτός, τότε ο φωτισμός θα είναι στη μέγιστη ένταση σε εκείνη την κορυφή. Αν οι δύο διευθύνσεις είναι κάθετες, τότε η κορυφή δεν φωτίζεται. Στο σχήμα 5.6 φαίνεται πως φωτίζεται μία επιφάνεια και μία κορυφή αναλόγως τα διανύσματα νόρμας. Στο σχήμα 5.7 φαίνεται ο φωτισμός στο περιβάλλον της Unity όταν το φως είναι παράλληλο και κάθετο στα διανύσματα νόρμας των κορυφών. Στο σχήμα αυτό τα διανύσματα νόρμας είναι όλα κάθετα στην επιφάνεια.



Σχήμα 5.6: Φωτισμός επιφάνειας και κορυφών σε σχέση με τα διανύσματα νόρμας



Σχήμα 5.7: Φωτισμός στο περιβάλλον Unity όταν η πηγή φωτός είναι παράλληλη (αριστερά) και κάθετη (δεξιά) στα διανύσματα νόρμας των κορυφών

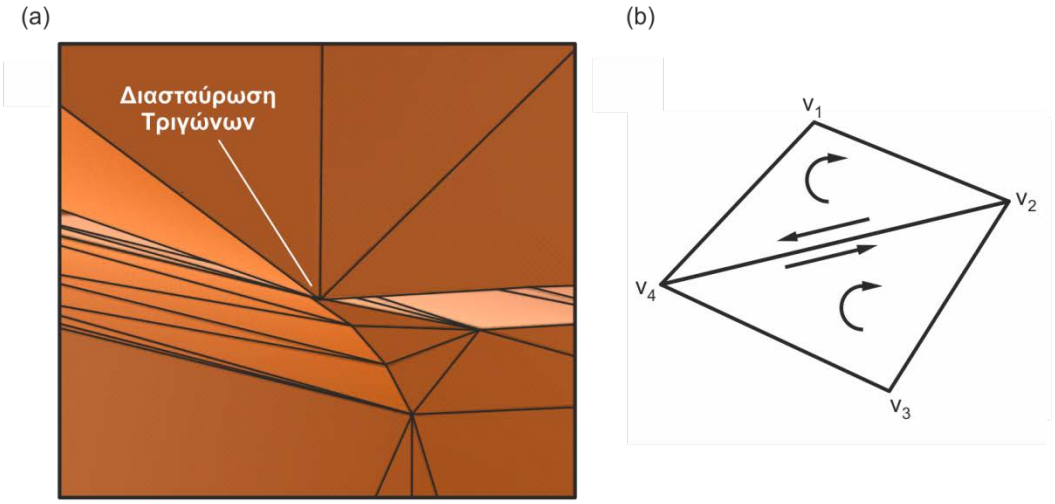
5.2 Παραδοχές τομής

Για την αποτελεσματική εφαρμογή των αλγορίθμων που θα περιγραφούν στη συνέχεια, λαμβάνονται οι παρακάτω παραδοχές για τα μοντέλα στα οποία θα εφαρμοστεί η τομή.

Παραδοχή 1: Κάθε ακμή του δικτύου γεωμετρίας μοιράζεται με δύο ακριβώς τρίγωνα και δεν πρέπει να υπάρχουν σημεία διασταύρωσης μεταξύ των τριγώνων, ακόμα και στην περίπτωση που η γεωμετρία αποτελείται από ξεχωριστά αντικείμενα σε μία συναρμολόγηση.

Παραδοχή 2: Δύο τρίγωνα που είναι ορατά και μοιράζονται μία κοινή ακμή θα πρέπει να έχουν αντίθετες κατευθύνσεις στην κοινή τους ακμή.

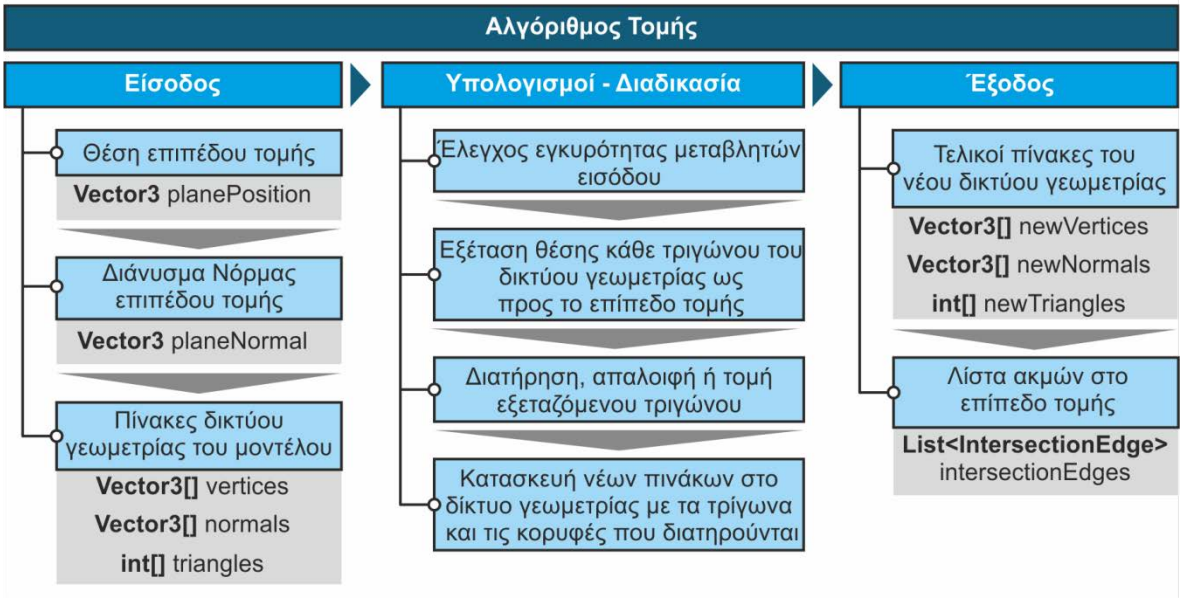
Σχετικά με την 1η παραδοχή, στο σχήμα 5.8 (a) φαίνεται μία γεωμετρία που αποτελείται από δύο ξεχωριστά αντικείμενα ενώ ταυτόχρονα υπάρχουν σημεία διασταύρωσης μεταξύ των τριγώνων. Μία τέτοια γεωμετρία δεν είναι αποδεκτή. Η 2η παραδοχή είναι προφανής εφόσον τα τρίγωνα είναι ορατά στη δεξιόστροφη φορά των κορυφών τους. Επομένως, σε δύο τρίγωνα με κοινή ακμή οι κατευθύνσεις θα είναι αντίθετες, όπως φαίνεται στο σχήμα 5.8 (b).



Σχήμα 5.8: Παράδειγμα μη αποδεκτής γεωμετρίας με διασταύρωση τριγώνων (a). Δύο τρίγωνα που μοιράζονται μία κοινή ακμή (b)

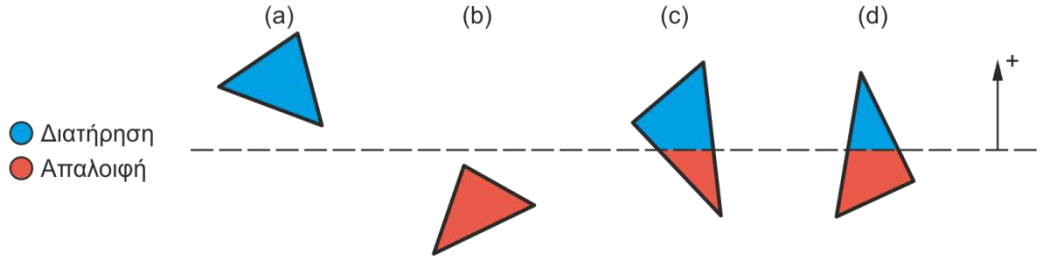
5.3 Ο Αλγόριθμος Τομής (Section Algorithm)

Το πρώτο στάδιο για τη διαδικασία τομής είναι η εφαρμογή του αλγορίθμου τομής στο μοντέλο. Ο αλγόριθμος αυτός, εξετάζοντας όλα τα τρίγωνα του δικτύου γεωμετρίας του μοντέλου, προσδιορίζει τη θέση τους σχετικά με το επίπεδο τομής το οποίο δίνεται ως είσοδος και εξετάζει ποιο μέρος του τριγώνου διατηρείται. Το επίπεδο τομής χωρίζει το χώρο σε δύο ημιεπίπεδα, όπου το θετικό ημιεπίπεδο προσδιορίζεται από ένα διάνυσμα νόρμας (**normal vector**) που είναι κάθετο στο επίπεδο τομής. Συνοπτικά ο αλγόριθμος παρουσιάζεται στο διάγραμμα του σχήματος 5.9.

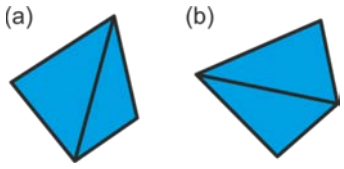


Σχήμα 5.9: Διάγραμμα Αλγορίθμου Τομής

Στο σχήμα 5.10 απαριθμούνται οι τέσσερις γενικές περιπτώσεις που μπορεί να βρίσκεται ένα τρίγωνο σε σχέση με το επίπεδο τομής. Εκτός από αυτές, υπάρχουν και ειδικές περιπτώσεις που θα αναφερθούν στη συνέχεια. Με μπλε σκίαση σημειώνεται το μέρος του τριγώνου που διατηρείται, ενώ με κόκκινη σκίαση το μέρος του τριγώνου που πρόκειται να απαλειφθεί. Όσα τρίγωνα βρίσκονται στην πλευρά του θετικού ημιεπιπέδου διατηρούνται (σχήμα 5.10 a) ενώ απαλείφονται τα τρίγωνα που βρίσκονται στο αρνητικό ημιεπίπεδο (σχήμα 5.11 b). Προφανώς, ένα τρίγωνο είναι δυνατόν να διασταυρώνεται με το επίπεδο τομής, αλλά και σε αυτή την περίπτωση θα πρέπει να γίνει τομή στο ίδιο το τρίγωνο ώστε να διατηρηθεί μόνο το μέρος του τριγώνου που βρίσκεται στο θετικό ημιεπίπεδο (σχήμα 5.11 c, d).



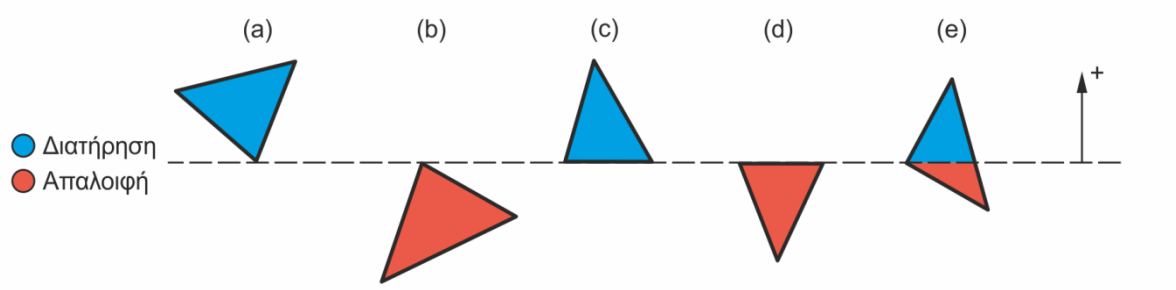
Σχήμα 5.10: Παραδείγματα των τεσσάρων περιπτώσεων που μπορεί να βρίσκεται το τρίγωνο σε σχέση με το επίπεδο τομής. Με βέλος προσδιορίζεται η κατεύθυνση της νόρμας του επιπέδου



Σχήμα 5.11: Διαχωρισμός Τετράπλευρου

Στις περιπτώσεις (c) και (d) δημιουργούνται δύο νέες κορυφές στο δίκτυο γεωμετρίας, οι οποίες είναι τα σημεία τομής των τριγώνων με το επίπεδο. Στη περίπτωση (d) δημιουργείται ένα νέο τρίγωνο ενώ στην περίπτωση (c) το μέρος του τριγώνου που διατηρείται είναι ένα τετράπλευρο. Το τετράπλευρο αυτό για να μπορεί να εισαχθεί στο δίκτυο γεωμετρίας θα πρέπει να χωριστεί σε δύο τρίγωνα όπως απεικονίζεται στο σχήμα 5.11.

Όσον αφορά τις ειδικές περιπτώσεις, τα τρίγωνα είναι δυνατόν να βρίσκονται σε οριακές θέσεις σε σχέση με το επίπεδο τομής. Οι οριακές αυτές θέσεις αφορούν τις περιπτώσεις όπου μία ή δύο κορυφές του τριγώνου βρίσκονται πάνω στο επίπεδο τομής. Οι περιπτώσεις αυτές απαιτούν ειδική διαχείριση και απεικονίζονται στο σχήμα 5.12.

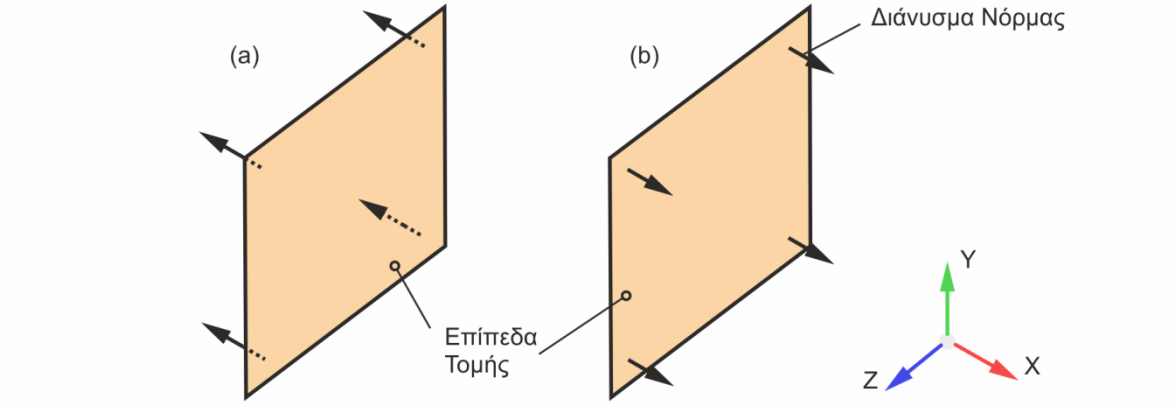


Σχήμα 5.12: Ειδικές περιπτώσεις θέσεων ενός τριγώνου σε σχέση με το επίπεδο τομής

Στις περιπτώσεις (a) και (c), όπου μία ή δύο κορυφές βρίσκονται πάνω στο επίπεδο τομής αντίστοιχα, τότε όλο το τρίγωνο διατηρείται, εφόσον το μεγαλύτερο μέρος του βρίσκεται στο θετικό ημιεπίπεδο. Αντίθετα, στις περιπτώσεις (b) και (d) το τρίγωνο απαλείφεται. Στην περίπτωση (e) όπως και στο σχήμα 5.10 (d) διατηρείται το μπλε σκιασμένο τρίγωνο με τη διαφορά ότι μόνο μία νέα κορυφή χρειάζεται να προστεθεί στο δίκτυο γεωμετρίας, εφόσον η δεύτερη κορυφή είναι ήδη κορυφή του αρχικού τριγώνου. Για όλες τις περιπτώσεις που εφαρμόζεται τομή σε κάποιο τρίγωνο, οι κορυφές στο επίπεδο τομής δημιουργούν μία νέα ακμή, η οποία αποθηκεύεται ξεχωριστά για χρήση στο επόμενο στάδιο της διαδικασίας τομής.

5.3.1 Υλοποίηση στη Unity

Η υλοποίηση του αλγορίθμου τομής γίνεται στην κλάση MeshClip. Η βασική συνάρτηση στην οποία υλοποιείται ο αλγόριθμος είναι η ClipMesh, η οποία δέχεται το διάνυσμα νόρμας και τη θέση του επιπέδου τομής στον τρισδιάστατο χώρο. Σημειώνεται ότι για τους σκοπούς της εφαρμογής, το διάνυσμα νόρμας του επιπέδου τομής έχει μόνο 6 διαφορετικές τιμές (2 για κάθε άξονα στον τρισδιάστατο χώρο). Στο σχήμα 5.13 παρουσιάζονται οι δύο εκδοχές του διανύσματος νόρμας για ένα επίπεδο τομής.



Σχήμα 5.13: Οι δύο εκδοχές του διανύσματος νόρμας του επιπέδου τομής για τον άξονα X. Τα βέλη των επιπέδων δείχνουν την κατεύθυνση του διανύσματος

Ο αλγόριθμος διατηρεί ένα πίνακα 6 θέσεων που για κάθε εκδοχή του διανύσματος νόρμας αποθηκεύει τη θέση του επιπέδου που έγινε η τελευταία τομή. Έτσι αποφεύγεται να ξαναγίνει τομή στην ίδια θέση με το ίδιο διάνυσμα νόρμας. Ο κώδικας του πίνακα 5.2 ε-

λέγχει εάν η θέση του επιπέδου είναι διαφορετική από την προηγούμενη, όπου και σε αυτήν την περίπτωση αντικαθίσταται η παλιά με τη νέα θέση και ο αλγόριθμος συνεχίζει την εκτέλεσή του.

Έλεγχος θέσης τομής

```
if (planeNormal == Vector3.left && (oldPlanePositions [0] == Vector3.positiveInfinity    || oldPl
anePositions [0] != planePosition)) {
    oldPlanePositions [0] = planePosition;
} else {
    Debug.Log ("Same plane position... No Clipping is necessary!");
    return -1;
}
```

Πίνακας 5.2: Κώδικας ελέγχου αν η θέση τομής είναι αποδεκτή

Αρχικά η συνάρτηση αρχικοποιεί κάποιες μεταβλητές οι οποίες είναι απαραίτητες στην πορεία του αλγορίθμου. Το μοντέλο στο οποίο θα εφαρμοστεί ο αλγόριθμος ανακτάται μέσω της συνάρτησης **GameObject.FindGameObjectWithTag()** που παρέχει η Unity. Έπειτα αρχικοποιούνται τρεις πίνακες στους οποίους θα αποθηκευτούν οι νέες κορυφές, τα νέα τρίγωνα και οι νέες νόρμες του δικτύου γεωμετρίας του μοντέλου. Το παλιό δίκτυο γεωμετρίας επίσης αποθηκεύεται σε περίπτωση που ο χρήστης επιθυμεί αργότερα να αναιρέσει την τομή που μόλις έκανε. Επιπλέον διατηρείται ένα πίνακας στον οποίο θα αποθηκεύονται οι ακμές που προκύπτουν επάνω στο επίπεδο από τις τομές των τριγώνων με αυτό. Έπειτα δημιουργείται ο δισδιάστατος πίνακας **vertexInfo** μεγέθους [vertexLength, 2], ο οποίος για κάθε κορυφή του παλιού δικτύου γεωμετρίας κρατά ως πληροφορία το εάν η κορυφή αυτή έχει προστεθεί στον νέο πίνακα κορυφών καθώς και τη θέση της στον πίνακα αυτόν. Έτσι αποφεύγεται η προσθήκη μίας κορυφής στον νέο πίνακα κορυφών πάνω από μία φορά. Τέλος, δημιουργείται ένα νέο αντικείμενο τύπου **Plane** το οποίο αντιστοιχεί στο επίπεδο τομής. Η δημιουργία του δισδιάστατου πίνακα vertexInfo καθώς και η αρχικοποίηση του επιπέδου τομής παρουσιάζεται στον κώδικα του πίνακα 5.3.

Δημιουργία πίνακα πληροφοριών για τις κορυφές και αρχικοποίηση του επιπέδου τομής.

```
vertexInfo = new int[oldVertices.Length, 2];

for (int i = 0; i<vertexInfo.GetLength (0); i++) {
    vertexInfo [i,0] = -1;    //(-1 -> not added , 1-> added);
    vertexInfo [i,1] = 0;
}
// new Vertice list index
listIndex = 0;
// Initialize the Clipping Plane
clipPlane = new Plane(planeNormal,planePosition);
```

Πίνακας 5.3: Κώδικας δημιουργίας πίνακα πληροφοριών κορυφών και αρχικοποίηση επιπέδου τομής

Στη συνέχεια για κάθε τρίγωνο προσδιορίζεται η θέση του σε σχέση με το επίπεδο τομής. Για να γίνει αυτό, αποθηκεύονται ξεχωριστά οι συντεταγμένες των κορυφών του τριγώνου και υπολογίζεται η απόσταση κάθε κορυφής από το επίπεδο. Για τον υπολογισμό της κάθετης απόστασης χρησιμοποιείται η συνάρτηση **GetDistanceToPoint(Vector3 point)**

που παρέχει η κλάση Plane της Unity και η οποία υπολογίζει την κάθετη απόσταση ενός σημείου από ένα επίπεδο. Οι αποστάσεις των τριών κορυφών από το επίπεδο τομής α-ποθηκεύονται ξεχωριστά.

Εάν οι αποστάσεις των τριών κορυφών είναι όλες θετικές ή ίσες με 0 τότε το τρίγωνο βρίσκεται εξ ολοκλήρου στο θετικό ημιεπίπεδο ή βρίσκεται σε κάποια από τις οριακές θέσεις του σχήματος 5.12 (a) και (c). Σε κάθε περίπτωση, το τρίγωνο προστίθεται στους νέους πίνακες μέσω της συνάρτησης **AddNewTriangle(int [] vertexIndices)**, η οποία δέχεται τους δείκτες κορυφών στον παλιό πίνακα κορυφών. Συγκεκριμένα και μόνο για την περίπτωση 5.12 (c), η ακμή που δημιουργείται πάνω στο επίπεδο προστίθεται στον πίνακα ακμών. Στον πίνακα 5.4 παρουσιάζονται οι συναρτήσεις προσθήκης νέου τριγώνου και ελέγχου κορυφής.

Συνάρτηση Προσθήκης νέου τριγώνου

```
void AddNewTriangle(int[] vertexIndices){

        int index;

        for (int i = 0; i < 3; i++) {
            index = vertexIndices [i];
            CheckAndAddOldVertex (index);

        }

    }
```

Συνάρτηση Ελέγχου κορυφής

```
void CheckAndAddOldVertex(int vertexIndex){
    if (vertexInfo [vertexIndex, 0] != 1) {
        newVertices.Add (oldVertices[vertexIndex]);
        newNormals.Add (oldNormals [vertexIndex]);
        newTriangles.Add (listIndex);

        vertexInfo [vertexIndex, 0] = 1;
        vertexInfo [vertexIndex, 1] = listIndex;

        listIndex++;
     } else {
      newTriangles.Add (vertexInfo [vertexIndex, 1]);
     }
}
```

Πίνακας 5.4: Οι συναρτήσεις προσθήκης νέου τριγώνου και ελέγχου κορυφής

Για κάθε κορυφή, η συνάρτηση **AddNewTriangle** καλεί τη συνάρτηση **CheckAndAddOldVertex.** Η τελευταία ελέγχει εάν η κορυφή που δόθηκε έχει ήδη προστεθεί στον νέο πίνακα κορυφών. Εάν έχει προστεθεί, τότε προσθέτει στον νέο πίνακα τριγώνων μία καταχώρηση με τη θέση της κορυφής αυτής. Διαφορετικά η καταχώρηση στο νέο πίνακα τριγώνων γίνεται αφού πρώτα η κορυφή προστεθεί στον νέο πίνακα κορυφών. Στην περίπτωση που και οι τρεις αποστάσεις των κορυφών είναι αρνητικές ή ίσες με το 0, τότε το τρίγωνο βρίσκεται στο αρνητικό ημιεπίπεδο ή σε κάποια από τις οριακές περιπτώσεις του σχήματος 5.12 (b) και (d). Σε τέτοιες περιπτώσεις το τρίγωνο απαλείφεται και ο αλγόριθμος προχωρά στην εξέταση του επόμενου τριγώνου.

Η πιο σημαντική περίπτωση είναι αυτή που απαιτείται τομή στο εξεταζόμενο τρίγωνο, καθώς αυτό βρίσκεται σε μία από τις περιπτώσεις του σχήματος 5.10 (c) και (d) και του σχήματος 5.12 (e). Εδώ απαιτείται ιδιαίτερη προσοχή ώστε να προσδιοριστούν σωστά οι κορυφές που βρίσκονται στο θετικό ή στο αρνητικό ημιεπίπεδο και οι κορυφές εκείνες που βρίσκονται επάνω στο επίπεδο τομής. Για το λόγο αυτό δημιουργούνται οι λίστες **positive** και **negative**, οι οποίες θα διαχωρίσουν τις κορυφές του θετικού ημιεπιπέδου από αυτές του αρνητικού. Σημειώνεται ότι εάν κάποια κορυφή βρίσκεται επάνω στο επίπεδο τομής, θα προστίθεται στη λίστα με τις κορυφές του αρνητικού ημιεπιπέδου. Στον πίνακα

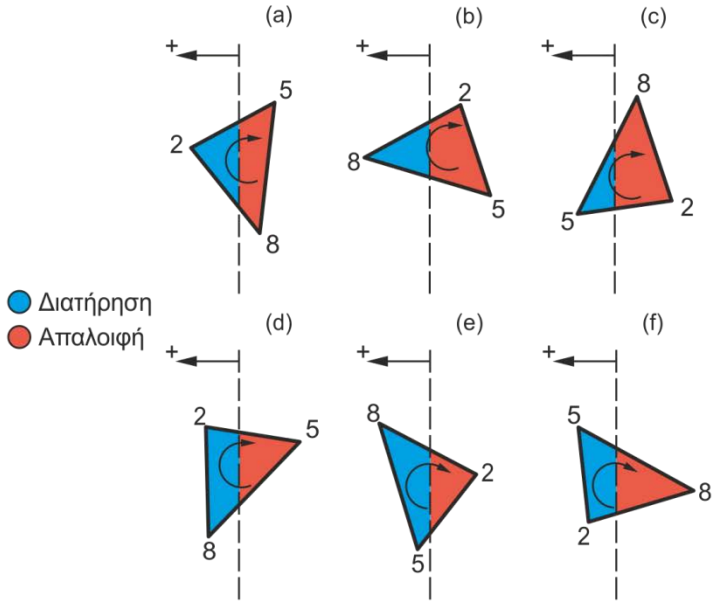
5.5 παρουσιάζεται ο κώδικας που διαχωρίζει τις κορυφές ανάλογα με τη θέση που βρίσκονται ως προς το επίπεδο τομής.

Διαχωρισμός κορυφών

```
for (int j = 0; j < 3; j++) {
    if (pointDistances [j] > 0f) {
        positive.Add(vertexIndices [j]);
    } else if (pointDistances [j] <= 0f) {
        negative.Add(vertexIndices [j]);
    }
}
```

Πίνακας 5.5: Κώδικας διαχωρισμού κορυφών του θετικού και αρνητικού ημιεπιπέδου

Υπενθυμίζεται ότι οι κορυφές του τριγώνου αποθηκεύονται με δεξιόστροφη φορά. Για παράδειγμα, αν οι δείκτες κορυφών του τριγώνου ήταν οι 2, 5, 8 τότε το τρίγωνο θα βρισκόταν σε μία από τις θέσεις του σχήματος 5.14.



Σχήμα 5.14: Περιπτώσεις θέσεων κορυφών για ένα τρίγωνο που βρίσκεται επάνω στο επίπεδο τομής. Οι αριθμοί δηλώνουν τους δείκτες της αντίστοιχης κορυφής στον πίνακα κορυφών. Το παράδειγμα αφορά το τρίγωνο με δείκτες 2,5,8

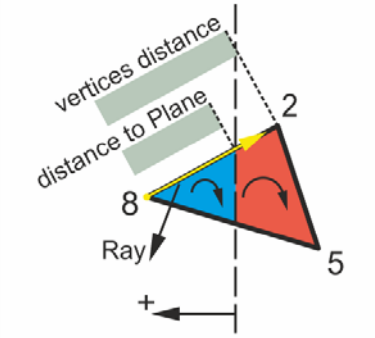
Στις τρεις πρώτες περιπτώσεις μόνο μία από τις κορυφές βρίσκεται στο θετικό ημιεπιπέδο, ενώ στις τρεις τελευταίες μόνο μία από τις κορυφές βρίσκεται στο αρνητικό ημιεπιπέδο. Για την ορθή λειτουργία της συνέχειας του αλγορίθμου, απαιτείται οι λίστες **positive** και **negative** εφόσον έχουν 2 στοιχεία, να έχουν πάντα ως πρώτο στοιχείο το δείκτη της κορυφής που βρίσκεται πιο ψηλά. Αυτό θα ισχύει σε όλες τις περιπτώσεις, εκτός από αυτές του σχήματος 5.14 (c), (e) και (f). Στις περιπτώσεις αυτές, επειδή οι κορυφές εξετάζονται με τη σειρά 2, 5, 8 δεν θα μπουν στις λίστες με τη επιθυμητή σειρά και έτσι απαιτείται αντιστροφή των στοιχείων της αντίστοιχης λίστας.

Αντιστροφή στοιχείων λίστας

```
if (positive.Count == 1 && pointDistances [1] > 0) {
   negative.Reverse ();
}
else if(negative.Count==1 && pointDistances [1] > 0){
   positive.Reverse ();
}
```

Πίνακας 5.6: Κώδικας αντιστροφής στοιχείων λίστας στις περιπτώσεις που οι κορυφές έχουν προστεθεί με λανθασμένη σειρά

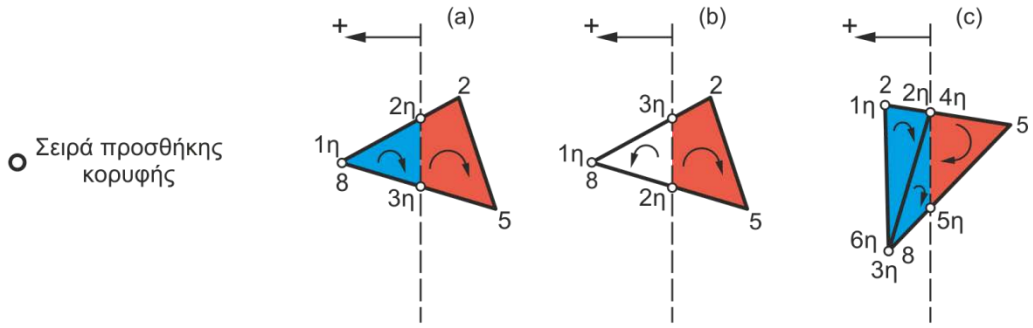
Στη συνέχεια θα πρέπει να υπολογιστούν τα σημεία τομής του τριγώνου με το επίπεδο. Ο υπολογισμός αυτός γίνεται με τη χρήση της κλάσης **Ray** και της συνάρτησης **Vector3.Lerp(Vector3 a, Vector3 b, float t)** που παρέχει η Unity. Ένα αντικείμενο τύπου Ray, αποτελεί μία νοητή ακτίνα η οποία δέχεται ως παραμέτρους τη θέση από την οποία ξεκινά καθώς και την κατεύθυνσή της. Στη συνέχεια με τη χρήση της συνάρτησης **Plane.Raycast(Ray ray, out float distance)** επιστρέφεται στη μεταβλητή **distance** η απόσταση από την αρχή της ακτίνας μέχρι την τομή της στο επίπεδο. Η απόσταση αυτή κανονικοποιείται έτσι ώστε να βρεθεί το ποσοστό της απόστασης των δύο κορυφών από το επίπεδο τομής. Η διαδικασία φαίνεται στο σχήμα 5.15.



$$\text{distance Normalized} = \frac{\text{distance to Plane}}{\text{vertices Distance}}$$

Σχήμα 5.15: Δημιουργία ακτίνας (Ray) και υπολογισμός κανονικοποιημένης απόστασης μεταξύ δύο κορυφών που βρίσκονται σε διαφορετικά ημιεπίπεδα

Εφόσον βρεθεί η κανονικοποιημένη απόσταση με τη συνάρτηση **Vector3.Lerp(Vector3 a, Vector3 b, float t)** υπολογίζεται η ακριβής θέση του σημείου τομής. Η συνάρτηση αυτή δέχεται ως ορίσματα δύο σημεία και επιστρέφει το σημείο που καθορίζεται από τη μεταβλητή t. Η μεταβλητή t δέχεται τιμές από 0 έως 1. Το σημείο που επιστρέφεται βρίσκεται στο ποσοστό t της απόστασης μεταξύ των σημείων a και b. Με την ίδια συνάρτηση υπολογίζεται και η νόρμα για το σημείο τομής όπου ως ορίσματα δίνονται τα διανύσματα νόρμας των κορυφών. Η αντιστροφή στις λίστες που εφαρμόστηκε παραπάνω, εξασφαλίζει ότι πάντα το πρώτο σημείο τομής που θα βρεθεί θα είναι αυτό που βρίσκεται πιο ψηλά στο επίπεδο τομής. Με τον παραπάνω τρόπο υπολογίζονται και τα δύο σημεία τομής με το επίπεδο. Η ακμή που δημιουργείται από τα δύο αυτά σημεία προστίθεται στον πίνακα ακμών. Ως επόμενο βήμα θα πρέπει να προστεθούν στον πίνακα τριγώνων οι δείκτες κορυφών των τριγώνων που δημιουργούνται. Σημαντικό ρόλο σε αυτό το σημείο έχει η σειρά που προστίθενται οι δείκτες. Λανθασμένη σειρά έχει ως αποτέλεσμα το τρίγωνο να σχεδιαστεί με αντίθετη φορά και συνεπώς να μην είναι ορατό από τη σωστή θέαση. Στο σχήμα 5.16 (a) και (c) απεικονίζεται ο σωστός τρόπος εισαγωγής των δεικτών κορυφών στον πίνακα τριγώνων. Η περίπτωση (b) αφορά το αποτέλεσμα μίας λανθασμένης εισαγωγής.

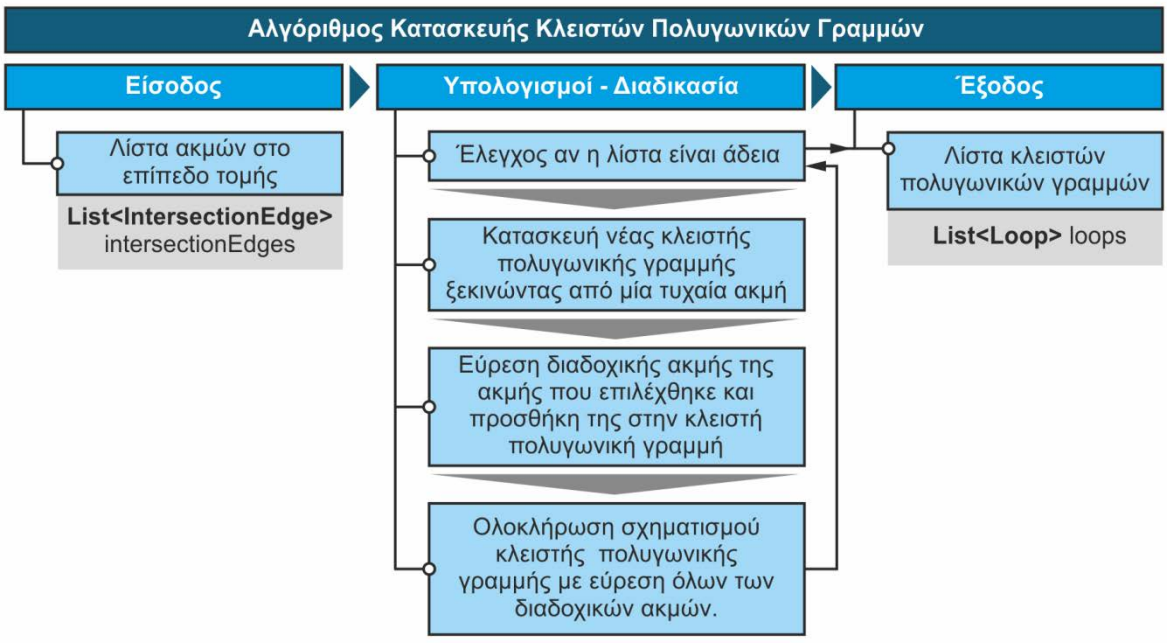


Σχήμα 5.16: Ορθή και λανθασμένη σειρά εισαγωγής των δεικτών κορυφών στον πίνακα τριγώνων

Όταν όλα τα τρίγωνα εξεταστούν και επεξεργαστούν, ο αλγόριθμος έχει ολοκληρωθεί. Το δίκτυο γεωμετρίας του μοντέλου ανανεώνεται με τους νέους πίνακες για τις κορυφές, τις νόρμες τους και τα τρίγωνα. Το τελικό αποτέλεσμα του αλγορίθμου είναι η τομή του μοντέλου επιφανειών όπως είχε φανεί στο σχήμα 5.3.

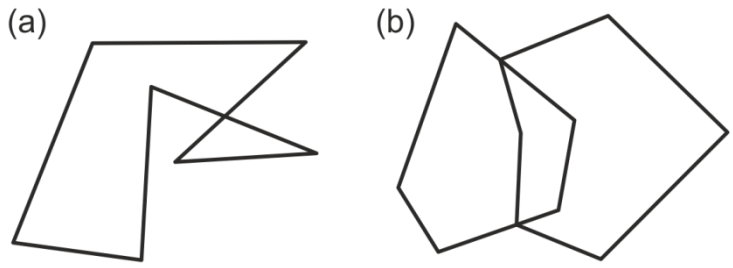
5.4 Ο Αλγόριθμος κατασκευής κλειστών Πολυγωνικών γραμμών

Όπως αναφέρθηκε στην ενότητα 5.3, ο αλγόριθμος τομής διατηρεί σε ένα πίνακα όλες τις ακμές που δημιουργούνται πάνω στο επίπεδο τομής, είτε από τομές τριγώνων, είτε από οριακές περιπτώσεις όπου μία πλευρά ενός τριγώνου βρίσκεται ακριβώς επάνω στο επίπεδο. Εφόσον ο αλγόριθμος τομής ολοκληρωθεί χωρίς σφάλματα, τότε ο πίνακας θα περιέχει κάποιες ακμές, οι οποίες αν ενωθούν σωστά θα δημιουργήσουν κάποιες κλειστές πολυγωνικές γραμμές επάνω στο επίπεδο τομής. Επειδή τα τρίγωνα του δικτύου γεωμετρίας στον αλγόριθμο τομής εξετάζονται με τυχαία σειρά, τότε οι ακμές του πίνακα ακμών μετά το πέρας του αλγορίθμου βρίσκονται επίσης σε τυχαίες θέσεις. Σκοπός του παρόντος αλγορίθμου είναι ο σχηματισμός και η αποθήκευση όλων των πολυγωνικών γραμμών που δημιουργούνται από την εξέταση όλων των ακμών του πίνακα ακμών. Ο σχηματισμός των πολυγωνικών γραμμών αποτελεί το πρώτο βήμα για το κλείσιμο των ανοικτών χώρων που δημιουργούνται, μετά την τομή του μοντέλου επιφανειών με το επίπεδο τομής. Ο αλγόριθμος παρουσιάζεται συνοπτικά στο διάγραμμα του σχήματος 5.17.



Σχήμα 5.17: Διάγραμμα Αλγορίθμου Κατασκευής κλειστών πολυγωνικών γραμμών

Βασική προϋπόθεση για την ορθή λειτουργία του αλγορίθμου είναι να μην σχηματίζονται πολυγωνικές γραμμές, οι οποίες διασταυρώνονται μεταξύ τους, ή πολυγωνικές γραμμές που έχουν αλληλοδιασταυρώμενες ακμές. Παραδείγματα τέτοιων πολυγωνικών γραμμών παρουσιάζονται στο σχήμα 5.18.

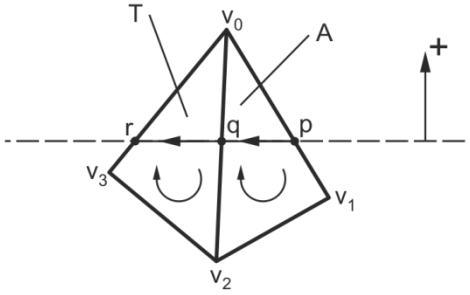


Σχήμα 5.18: Παραδείγματα μη αποδεκτών κλειστών πολυγωνικών γραμμών. Πολυγωνική γραμμή με αλληλοδιασταυρώμενες ακμές (a). Αλληλοδιασταυρώμενες πολυγωνικές γραμμές (b)

Στη συνέχεια θα αποδειχθεί, ότι οι δύο παραπάνω περιπτώσεις δεν είναι δυνατόν να εμφανιστούν στα μοντέλα στα οποία εφαρμόζεται ο αλγόριθμος.

Αρχικά θα αποδειχθεί, ότι για κάθε ακμή $s = pq$ που σχηματίζεται επάνω στο επίπεδο τομής, υπάρχει μία διαδοχική ακμή $s' = qr$ στον πίνακα ακμών S , όπου $s, s' \in S$ και p, q, r οι κορυφές των ακμών s και s'.

Απόδειξη: Έστω $A = (v_0, v_1, v_2)$ ένα τρίγωνο του δικτύου γεωμετρίας M με $A \in M$, όπου τουλάχιστον μία από τις κορυφές του τριγώνου $v \in (v_0, v_1, v_2)$ βρίσκεται στο θετικό ημιεπίπεδο και τουλάχιστον μία από τις κορυφές, έστω $w \in (v_0, v_1, v_2)$ βρίσκεται στο αρνητικό ημιεπίπεδο. Τότε η τομή του τριγώνου A με το επίπεδο τομής P , θα δημιουργήσει μία ακμή $s = pq$ με $s \in S$ όπου p και q είναι αντίστοιχα τα δύο σημεία τομής των πλευρών e_0 και e_1 του τριγώνου A με το επίπεδο τομής P , όπως φαίνεται στο σχήμα 5.19 .



Σχήμα 5.19: Τομή δύο τριγώνων που μοιράζονται μία κοινή ακμή.

Λόγω της παραδοχής 1 της ενότητας 5.2, θα υπάρχει ακριβώς ένα τρίγωνο T όπου $T \in M, T \neq A$, το οποίο μοιράζεται μία κοινή ακμή με το τρίγωνο A. Συνεπώς η τομή του τριγώνου T με το επίπεδο τομής P , θα δημιουργήσει

επίσης μία ακμή s' επάνω στο επίπεδο τομής, της οποίας το ένα άκρο θα είναι το σημείο q. Λόγω της παραδοχής 2 της ενότητας 5.2, η κοινή πλευρά στο τρίγωνο T έχει αντίθετη φορά από ότι στο τρίγωνο A. Επομένως η ακμή s' θα έχει κατεύθυνση από το σημείο q προς ένα νέο σημείο r, που βρίσκεται επάνω στο επίπεδο τομής. Επομένως για κάθε ακμή s υπάρχει και μία διαδοχική ακμή s'.

Στη συνέχεια, θα αποδειχθεί ότι εκτός από δύο διαδοχικές ακμές οι οποίες διασταυρώνονται μόνο στο κοινό άκρο τους, δεν υπάρχει άλλο ζευγάρι ακμών $e_1, e_2 \in S$ με $e_1 \neq e_2$ το οποίο διασταυρώνεται.

Απόδειξη: Έστω $A, B \in M, A \neq B$ δύο τρίγωνα του δικτύου γεωμετρίας M, τα οποία βρίσκονται επάνω στο επίπεδο τομής δημιουργώντας τις δύο ακμές $s_1, s_2 \in S, s_1 \neq s_2$ επάνω σε αυτό. Έστω ότι η πρόταση $s_1 \cap s_2 \neq 0 \Rightarrow A, B \neq 0$ είναι αληθής. Στην περίπτωση αυτή θα πρέπει να ισχύει ένα από τα παρακάτω:

- Τα τρίγωνα A, B μοιράζονται μία κοινή ακμή η οποία έχει ένα σημείο τομής με το επίπεδο τομής, όπως περιγράφεται στην προηγούμενη απόδειξη και επομένως οι ακμές s_1, s_2 είναι διαδοχικές ακμές, οι οποίες διασταυρώνονται μόνο στο κοινό άκρο τους.
- Το δίκτυο γεωμετρίας M δεν ικανοποιεί την παραδοχή 1 εφόσον υπάρχουν σημεία διασταύρωσης μεταξύ των τριγώνων.

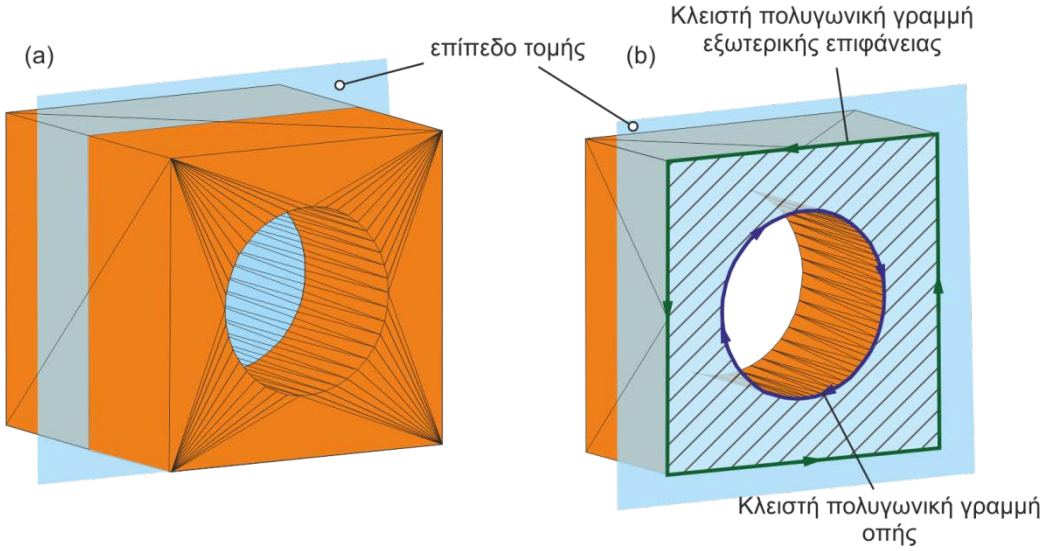
Επομένως έχει αποδειχθεί ότι αφού δεν υπάρχει κανένα ζευγάρι ακμών το οποίο περιέχει σημεία διασταύρωσης, εκτός από αυτό το οποίο αποτελείται από δύο διαδοχικές ακμές, τότε δεν σχηματίζονται πολυγωνικές γραμμές στο επίπεδο τομής οι οποίες διασταυρώνονται μεταξύ τους, ή πολυγωνικές γραμμές που έχουν αλληλοδιασταυρώμενες ακμές.

Στη συνέχεια θα περιγραφεί η λειτουργία του αλγορίθμου. Αρχικά ο αλγόριθμος ξεκινά την κατασκευή μία νέας πολυγωνικής γραμμής, επιλέγοντας ως αρχική ακμή μία τυχαία από τον πίνακα ακμών. Κάθε ακμή έχει ένα άκρο αρχής και ένα άκρο τέλους. Έτσι εφαρμόζεται αναζήτηση στον πίνακα ακμών για μία διαδοχική ακμή και συγκεκριμένα για μία ακμή που έχει άκρο αρχής ίδιο με το άκρο τέλους της επιλεγμένης ακμής. Εφόσον η διαδοχική ακμή βρεθεί, τότε προστίθεται στην πολυγωνική γραμμή. Αν η ακμή αυτή κλείνει την πολυγωνική γραμμή τότε, η κατασκευή έχει ολοκληρωθεί και ο αλγόριθμος ξεκινά από την αρχή, δημιουργώντας μία νέα πολυγωνική γραμμή. Αν η ακμή δεν κλείνει την πολυγωνική γραμμή, τότε γίνεται αναζήτηση της επομένης διαδοχικής ακμής. Η διαδικασία επαναλαμβάνεται μέχρι να μην μείνει καμία ακμή η οποία δεν είναι ήδη μέρος κάποιας κλειστής πολυγωνικής γραμμής.

Εφόσον ο αλγόριθμος ολοκληρωθεί, τότε έχουν σχηματιστεί όλες οι πολυγωνικές γραμμές. Σημειώνεται ότι η κατεύθυνση κάθε πολυγωνικής γραμμής εξαρτάται άμεσα από τη κατεύθυνση των τριγώνων. Όπως έχει ήδη αναφερθεί τα τρίγωνα σχεδιάζονται δεξιόστροφα. Επίσης οι κλειστές πολυγωνικές γραμμές αποτελούν, είτε το περίγραμμα της εξωτερικής επιφάνειας του μοντέλου, είτε το περίγραμμα μίας οπής η οποία υπάρχει στο

μοντέλο. Συνεπώς οι πολυγωνικές γραμμές που αφορούν περίγραμμα της εξωτερικής επιφάνειας θα έχουν αριστερόστροφη κατεύθυνση, ενώ αυτές που αφορούν το περίγραμμα κάποιας οπής θα έχουν δεξιόστροφη κατεύθυνση. Η κατεύθυνση κάθε πολυγωνικής γραμμής παίζει πολύ σημαντικό ρόλο σε επόμενο στάδιο της διαδικασίας τομής.

Η κατεύθυνση των πολυγωνικών γραμμών που σχηματίζονται από την τομή ενός μοντέλου το οποίο περιέχει μία οπή παρουσιάζεται στο σχήμα 5.20. Συγκεκριμένα στο σχήμα 5.20 (a) παρουσιάζεται το αρχικό μοντέλο ενός κύβου, ο οποίος περιέχει στο κέντρο του μία κυλινδρική οπή. Στο σχήμα 5.20 (b) παρουσιάζεται ο ίδιος κύβος μετά την τομή του. Αρχικά παρατηρείται ότι η συγκεκριμένη όψη της τομής επιτρέπει την θέαση μόνο ορισμένων τριγώνων. Η πράσινη πολυγωνική γραμμή αφορά το περίγραμμα της εξωτερικής επιφάνειας του μοντέλου ενώ η μπλε πολυγωνική γραμμή αφορά το περίγραμμα της κυλινδρικής οπής. Η σκιαγραφημένη επιφάνεια είναι αυτή που πρέπει να καλυφθεί, προκειμένου να μην φαίνεται ο ανοικτός χώρος που δημιουργείται μετά την τομή. Όπως φαίνεται στο σχήμα, η δεξιόστροφη πολυγωνική γραμμή ορίζει την επιφάνεια που δεν πρέπει να καλυφθεί σε αντίθεση με την αριστερόστροφη.



Σχήμα 5.20: Τομή ενός μοντέλου και παρουσίαση των κλειστών πολυγωνικών γραμμών που θα κατασκευαστούν από τον αλγόριθμο

5.4.1 Υλοποίηση στη Unity

Για τους σκοπούς της υλοποίησης του αλγορίθμου δημιουργήθηκαν δύο νέες κλάσεις. Η κλάση **IntersectionEdge** αναπαριστά μία ακμή που δημιουργείται στο επίπεδο τομής και περιέχει μόνο 2 μεταβλητές τύπου Vector3, στις οποίες θα αποθηκεύονται τα δύο άκρα της ακμής. Η κλάση **Loop** αναπαριστά μία κλειστή πολυγωνική γραμμή και επομένως περιέχει μία λίστα τύπου IntersectionEdge με όλες τις ακμές της γραμμής αυτής, καθώς και μία λίστα κορυφών. Σημειώνεται ότι οι ακμές αποθηκεύονται διαδοχικά στη λίστα. Ο αλγόριθμος υλοποιείται και πάλι στην κλάση MeshClip και συγκεκριμένα στη συνάρτηση **ConstructLoops()**.

Όπως έχει αναφερθεί νωρίτερα, ο αλγόριθμος τομής προσθέτει στη λίστα ακμών τις ακμές που προκύπτουν στο επίπεδο τομής από τις τομές τριγώνων. Ο παρών αλγόριθμος ξεκινά, δημιουργώντας ένα νέο αντικείμενο πολυγωνικής γραμμής (Loop) και προσθέτοντας σε αυτή το πρώτο στοιχείο από τη λίστα ακμών. Το στοιχείο αυτό είναι η υπό εξέταση ακμή. Στη συνέχεια γίνεται αναζήτηση για τη διαδοχική ακμή της και εφόσον αυτή βρεθεί,

γίνεται ταυτόχρονα η νέα υπό εξέταση ακμή. Η επανάληψη ολοκληρώνεται όταν η κορυφή τέλους της υπό εξέτασης ακμής ισούται με την αρχική κορυφή της πολυγωνικής γραμμής. Στον πίνακα 5.7 παρουσιάζεται η διαδικασία αρχικοποίησης μίας νέας πολυγωνικής γραμμής. Στον πίνακα 5.8 παρουσιάζεται η διαδικασία αναζήτησης της διαδοχικής ακμής για την υπό εξέταση ακμή.

Δημιουργία νέας πολυγωνικής γραμμής

```
Loop newLoop = new Loop ();
Vector3 loopStartVertex;
IntersectionEdge currentEdge = intersectionEdges [0];
newLoop.AddEdge (currentEdge);
intersectionEdges.RemoveAt (0);
loopStartVertex = newLoop.GetFirstEdge ().GetStartVertex ();
```

Πίνακας 5.7: Κώδικας δημιουργίας νέας πολυγωνικής γραμμής

Αναζήτηση διαδοχικής ακμής

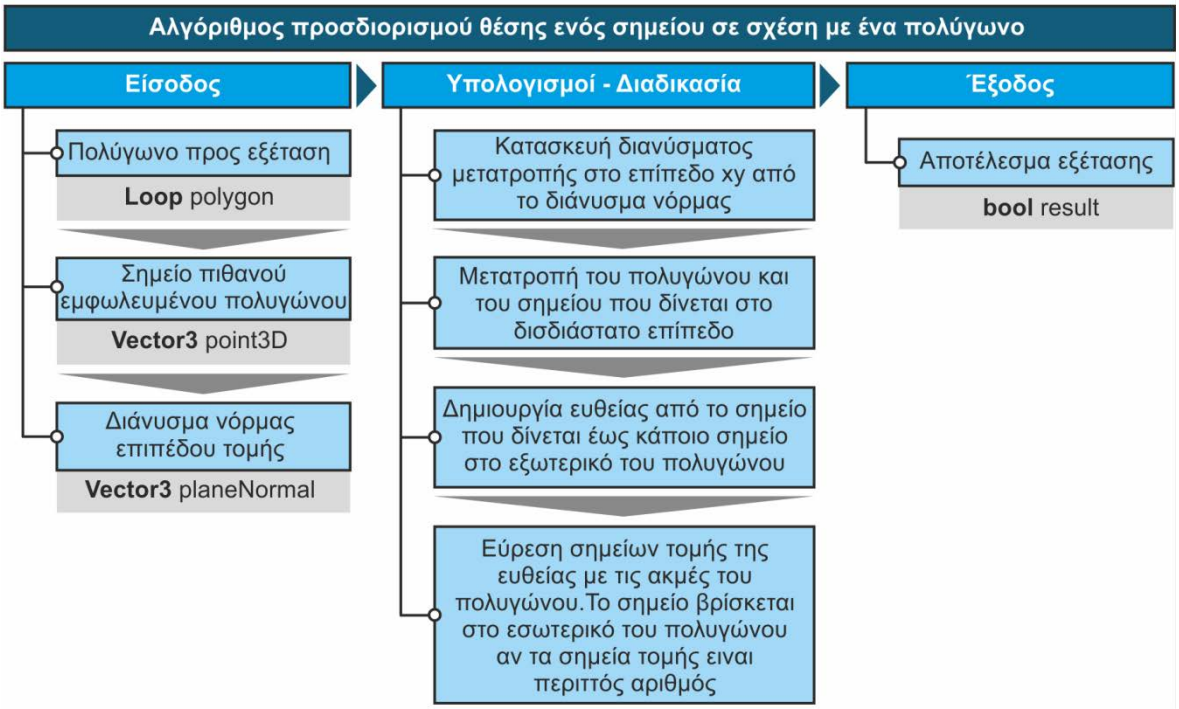
```
IntersectionEdge successorOfCurrent=null;
int successorPosition = -1;

for (int i=0; i<intersectionEdges.Count;i++){

    if (currentEdge.GetEndVertex() == intersectionEdges[i].GetStartVertex()){
        successorOfCurrent=intersectionEdges[i];
        successorPosition = i;
        break;
    }
}
```

Πίνακας 5.8: Κώδικας αναζήτησης διαδοχικής ακμής

Σημειώνεται ότι η σύγκριση για ισότητα με τον τελεστή == σε μεταβλητές float, συγκρίνει μόνο τα 5 πρώτα δεκαδικά ψηφία αυτών. Έτσι αποφεύγονται λανθασμένες συγκρίσεις που μπορεί να οφείλονται σε ανακρίβειες δεκαδικών ψηφίων.

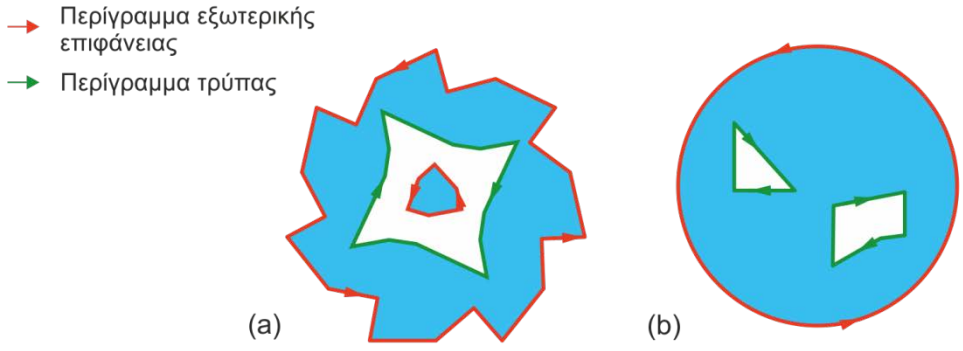
5.5 Ο Αλγόριθμος εύρεσης εμφωλευμένων πολυγώνων

Στην ενότητα αυτή οι κλειστές πολυγωνικές γραμμές της ενότητας 5.4, θα αναφέρονται απλά ως πολύγωνα. Όπως είναι προφανές τα πολύγωνα που δημιουργήθηκαν από τον προηγούμενο αλγόριθμο, είναι δυνατόν να είναι εμφωλευμένα μεταξύ τους. Ο στόχος του παρόντος αλγορίθμου είναι να προσδιορίσει εάν κάποιο πολύγωνο περιέχει εμφωλευμένα πολύγωνα και ποια είναι αυτά. Για όλα τα πολύγωνα έχει αποδειχθεί, ότι δεν είναι δυνατόν να διασταυρώνονται μεταξύ τους. Συνεπώς για να προσδιοριστεί εάν κάποιο πολύγωνο είναι εμφωλευμένο σε κάποιο άλλο, αρκεί να εξεταστεί εάν ένα τυχαίο σημείο του βρίσκεται στο εσωτερικό του άλλου. Ο αλγόριθμος ελέγχου ενός σημείου συνοψίζεται στο σχήμα 5.21.



Σχήμα 5.21: Διάγραμμα Αλγορίθμου προσδιορισμού θέσης ενός σημείου

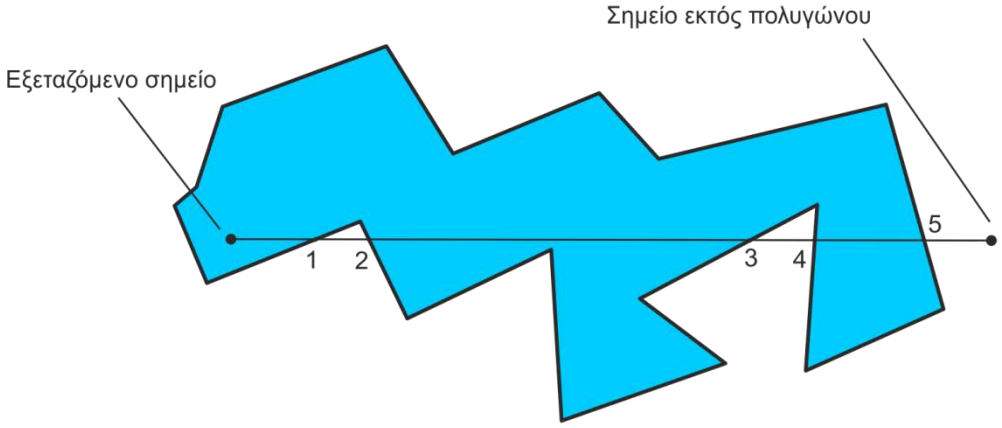
Τα πολύγωνα που βρίσκονται στο εσωτερικό άλλων πολυγώνων αποτελούν, όπως αναφέρθηκε και στην ενότητα 5.4, περιγράμματα από οπές που υπάρχουν στο μοντέλο. Σημειώνεται ότι για τα μοντέλα που χρησιμοποιούνται στην εφαρμογή, δεν υπάρχουν περιπτώσεις που σχηματίζονται διπλά ή τριπλά εμφωλευμένα πολύγωνα, δηλαδή να υπάρχει περίγραμμα εξωτερικής επιφάνειας στο εσωτερικό κάποιας οπής. Τέτοιες περιπτώσεις θα καθιστούσαν τον αλγόριθμο ακατάλληλο, καθώς θα ήταν δύσκολο να προσδιοριστεί ποια εμφωλευμένα πολύγωνα αποτελούσαν περίγραμμα εξωτερικής επιφάνειας και ποια περίγραμμα κάποιας οπής. Στο σχήμα 5.22 (a) παρουσιάζεται ένα παράδειγμα διπλά εμφωλευμένων πολυγώνων, ενώ στο σχήμα 5.22 (b) παρουσιάζεται ένα παράδειγμα δύο απλών εμφωλευμένων πολυγώνων.



Σχήμα 5.22: Παραδείγματα διπλά εμφωλευμένων πολυγώνων (a) και απλών εμφωλευμένων πολυγώνων (b)

Εφόσον οι περιπτώσεις όπως αυτές του σχήματος 5.22 (a) δεν παρουσιάζονται, στη συνέχεια θα εξηγηθεί ο αλγόριθμος που εξετάζει εάν ένα σημείο βρίσκεται στο εσωτερικό ενός πολυγώνου. Ο αλγόριθμος αυτός είναι γνωστός και ως **αλγόριθμος Ray Casting**. Αρχικά επειδή το πολύγωνο καθώς και το σημείο που εξετάζει ο αλγόριθμος βρίσκονται στον τρισδιάστατο χώρο, απαιτείται η μετατροπή τους στο δισδιάστατο επίπεδο. Στη συνέχεια κατασκευάζεται μία ευθεία από το σημείο προς εξέταση, προς ένα τυχαίο σημείο

στο εξωτερικό του πολυγώνου. Τα σημεία τομής της ευθείας με τις ακμές του πολυγώνου, προσδιορίζουν εάν το σημείο προς εξέταση βρίσκεται στο εσωτερικό ή στο εξωτερικό του πολυγώνου. Εάν το εξεταζόμενο σημείο βρίσκεται στο εξωτερικό του πολυγώνου, τότε θα υπάρχει άρτιος αριθμός σημείων τομής. Διαφορετικά, ο αριθμός των σημείων τομής θα είναι περιττός. Στο σχήμα 5.23 παρουσιάζεται μία εφαρμογή του αλγορίθμου.



Σχήμα 5.23: Παράδειγμα εφαρμογής του αλγορίθμου Ray Casting.

Στο παράδειγμα του σχήματος 5.23 οι αριθμοί δηλώνουν τον αριθμό των σημείων τομής της ευθείας μέχρι εκείνο το σημείο της. Το σημείο που εξετάζεται βρίσκεται στο εσωτερικό του πολυγώνου. Αυτό επαληθεύεται καθώς ο αριθμός των σημείων τομής της ευθείας μέχρι το σημείο εκτός του πολυγώνου είναι περιττός. Σημειώνεται ότι η μεθοδολογία αυτή δεν είναι ορθή εάν το εξεταζόμενο σημείο βρίσκεται επάνω σε κάποια ακμή του πολυγώνου. Ο αλγόριθμος αυτός εφαρμόζεται για κάθε συνδυασμό πολυγώνων μέχρις ότου να βρεθούν όλα τα εμφωλευμένα πολύγωνα κάθε πολυγώνου εφόσον αυτά υπάρχουν.

5.5.1 Υλοποίηση στη Unity

Η υλοποίηση του αλγορίθμου Ray Casting γίνεται στην συνάρτηση **IsPointInPolygon** η οποία βρίσκεται στην κλάση MeshClip. Τα ορίσματα της συνάρτησης αναφέρονται στο σχεδιάγραμμα του σχήματος 5.21. Το πρώτο βήμα είναι ο μετασχηματισμός του πολυγώνου και του σημείου που δίνεται στο επίπεδο xy. Αυτό γίνεται με τη χρήση της κλάσης **Quaternion** που παρέχει η Unity. Η κλάση Quaternion χρησιμοποιείται για την αναπαράσταση περιστροφών και παρέχει ιδιαίτερα χρήσιμες συναρτήσεις για το χειρισμό τους. Με τη χρήση της συνάρτησης **Quaternion.FromToRotation(Vector3 fromDirection, Vector3 toDirection)** δημιουργείται ένα διάνυσμα περιστροφής από την κατεύθυνση fromDirection προς την κατεύθυνση toDirection. Στις παραμέτρους fromDirection και toDirection δίνεται το διάνυσμα κατεύθυνσης της νόρμας του επιπέδου και το διάνυσμα κατεύθυνσης **Vector2.forward**.

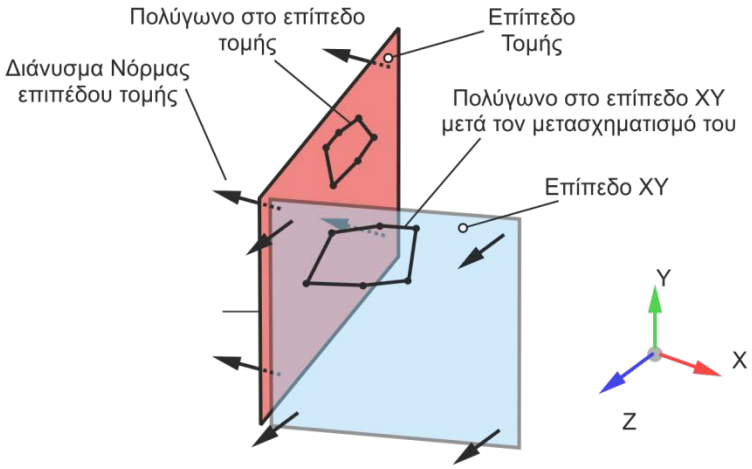
Το διάνυσμα Vector3.forward είναι κάθετο στο επίπεδο xy. Για να μετασχηματιστεί ένα σημείο από το επίπεδο τομής στο επίπεδο xy, αρκεί να πολλαπλασιαστεί με το διάνυσμα περιστροφής που δημιουργήθηκε. Στο κώδικα του πίνακα 5.9 παρουσιάζεται η δημιουργία του διανύσματος περιστροφής και ο μετασχηματισμός του εξεταζόμενου σημείου από το επίπεδο τομής στο επίπεδο xy. Η παράμετρος z του σημείου θα έχει πλέον την τιμή 0, οπότε το σημείο μετατρέπεται σε μεταβλητή τύπου Vector2.

Δημιουργία διανύσματος περιστροφής και μετασχηματισμός σημείου στο επίπεδο xy

```
Quaternion rotation;
if (planeNormal == Vector3.forward) {
   rotation = Quaternion.identity;
} else {
   rotation = Quaternion.FromToRotation (planeNormal, Vector3.forward);
}

Vector2 point = (Vector2)(rotation * point3D);
```

Πίνακας 5.9: Κώδικας μετασχηματισμού σημείου στο επίπεδο xy

Στο σχήμα 5.24 παρουσιάζεται σχηματικά το αποτέλεσμα μετασχηματισμού του πολυγώνου στο επίπεδο xy.



Σχήμα 5.24: Αποτέλεσμα μετασχηματισμού πολυγώνου στο επίπεδο xy

Εφόσον το πολύγωνο καθώς και σημείο το οποίο θα εξεταστεί βρίσκονται πλέον στο επίπεδο xy, ο αλγόριθμος προχωρά στο επόμενο βήμα. Η διαδικασία που παρουσιάζεται στο σχήμα 5.23 ακολουθεί κάποιες διαφοροποιήσεις προκειμένου να υλοποιηθεί προγραμματιστικά, χωρίς όμως να επιφέρει διαφορετικό αποτέλεσμα. Ουσιαστικά ο αλγόριθμος για κάθε ακμή του πολυγώνου, εξετάζει εάν το εξεταζόμενο σημείο βρίσκεται ανάμεσα στα δύο άκρα της και δεξιά από αυτήν. Δηλαδή, ο αλγόριθμος αναζητεί όλες τις ακμές που βρίσκονται αριστερά του εξεταζόμενου σημείου. Έτσι προσομοιώνεται μία οριζόντια ευθεία, με δύο άκρα το σημείο προς εξέταση και κάποιο σημείο εκτός και αριστερά του πολυγώνου.

Εφόσον επαληθευτεί ότι το σημείο βρίσκεται ανάμεσα στα δύο άκρα της εξεταζόμενης ακμής, δημιουργείται η εξίσωση ευθείας που διέρχεται από αυτά. Έστω $y = ax + b$ η εξίσωση ευθείας και (x_1, y_1) το εξεταζόμενο σημείο. Τότε αν $y_1 < ax_1 + b$ και $a > 0$ ή $y_1 > ax_1 + b$ και $a < 0$ το σημείο βρίσκεται στα δεξιά της ευθείας. Για κάθε ακμή που ικανοποιεί τα παραπάνω κριτήρια, αλλάζει η τιμή μίας λογικής μεταβλητής η οποία έχει αρχικοποιηθεί με την τιμή false. Έτσι, μετά την ολοκλήρωση του αλγορίθμου, η τιμή της μεταβλητής αυτής καθορίζει εάν το εξεταζόμενο σημείο βρίσκεται εντός του πολυγώνου.

5.6 Η Διπλά Συνδεμένη Λίστα Ακμών (DCEL)

Για την υλοποίηση των αλγορίθμων που θα αναφερθούν στις επόμενες ενότητες, καθίσταται αναγκαία η υλοποίηση μίας δομής δεδομένων για την αποτελεσματική διαχείριση των ακμών, κορυφών και επιφανειών των πολυγώνων που έχουν δημιουργηθεί. Η δομή αυτή ονομάζεται διπλά συνδεμένη λίστα ακμών και είναι γνωστή ως **Doubly-Connected Edge List (DCEL)** ή **Half-Edge Data Structure**.

Η δομή αυτή χρησιμοποιείται πολύ συχνά σε αλγορίθμους υπολογιστικής γεωμετρίας, καθώς προσφέρει ιδιαίτερα χρήσιμες λειτουργίες, όπως την προσπέλαση όλων των ακμών μίας επιφάνειας, την προσπέλαση όλων των ακμών που έχουν κοινό άκρο μία συγκεκριμένη κορυφή ή την άμεση πρόσβαση στην επόμενη και προηγούμενη ακμή μίας ακμής. Οι λειτουργίες αυτές είναι αρκετά πολύπλοκες όταν όλες οι πληροφορίες είναι αποθηκευμένες σε απλούς πίνακες ή λίστες.

Έστω ένα πολύγωνο το οποίο είναι δυνατόν να περιέχει εμφωλευμένα πολύγωνα, τα οποία ορίζουν κάποιες οπές. Η δομή κρατά μία καταχώρηση για κάθε κορυφή, επιφάνεια και ακμή του συνόλου πολυγώνων. Η κύρια ιδέα της δομής είναι ότι εφόσον κάθε ακμή αποτελεί πάντα όριο δύο διαφορετικών επιφανειών, τότε αντικαθίσταται από δύο κατευθυνόμενες ημι-ακμές, μία για κάθε επιφάνεια, γνωστές και ως **Half-Edges**. Οι ημι-ακμές αυτές έχουν αντίθετες κατευθύνσεις και η μία ορίζεται ως δίδυμη (**Twin**) της άλλης. Στη συνέχεια αναφέρονται όλες οι μεταβλητές που αποθηκεύονται στη δομή για κάθε κορυφή, επιφάνεια και ακμή του συνόλου πολυγώνων.

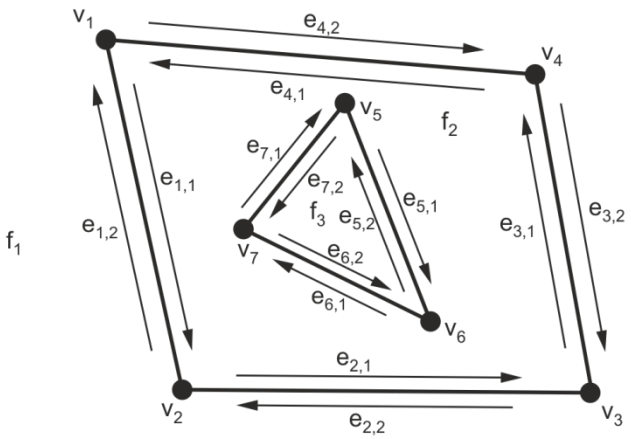
- Για κάθε κορυφή v αποθηκεύονται οι συντεταγμένες της στο πεδίο $Position(v)$ καθώς και ο δείκτης $IncidentEdge(v)$, ο οποίος δείχνει σε μία τυχαία προσπίπτουσα ημι-ακμή που έχει την κορυφή v ως άκρο αρχής της.

- Για κάθε επιφάνεια f αποθηκεύεται ο δείκτης $OuterComponent(f)$, ο οποίος δείχνει σε μία τυχαία ημι-ακμή στο περίγραμμα της επιφάνειας. Σημειώνεται ότι στη δομή αποθηκεύεται επιπλέον μία επιφάνεια, η οποία είναι εξωτερική όλων των πολυγώνων και δεν έχει περίγραμμα. Η επιφάνεια αυτή ορίζεται ως **unbounded Face.** Επίσης για κάθε επιφάνεια f, αποθηκεύεται η λίστα δεικτών $InnerComponents(f)$, η οποία για κάθε οπή στο εσωτερικό του πολυγώνου περιέχει ένα δείκτη προς μία ημι-ακμή στο περίγραμμα της οπής.

- Για κάθε ημι-ακμή e αποθηκεύεται ο δείκτης $Origin(e)$ προς την κορυφή του άκρου αρχής της, ο δείκτης $Twin(e)$ προς τη δίδυμη ημι-ακμή της, καθώς και ο δείκτης $IncidentFace(e)$ προς την προσπίπτουσα επιφάνεια που βρίσκεται στα αριστερά της ημι-ακμής. Σημειώνεται ότι δεν αποθηκεύεται δείκτης προς την κορυφή στο άκρο τέλους, καθώς αυτή ισούται με $Origin(Twin(e))$. Επιπλέον στην ημι-ακμή αποθηκεύονται οι δείκτες $Next(e)$ και $Prev(e)$, οι οποίοι δείχνουν στην επομένη και στην προηγούμενη ημι-ακμή με ίδια προσπίπτουσα επιφάνεια.

Στο σχήμα 5.25 παρουσιάζεται ένα παράδειγμα ενός πολυγώνου με μία οπή στο εσωτερικό του, πάνω στο οποίο σημειώνονται όλες οι κορυφές, ακμές και επιφάνειες που αποθηκεύονται στη Διπλά Συνδεμένη Λίστα Ακμών.

v: Κορυφή
e: Ακμή
f: Επιφάνεια



Σχήμα 5.25: Παράδειγμα δύο πολυγώνων αποθηκευμένα σε Διπλά Συνδεμένη Λίστα ακμών

Στους παρακάτω πίνακες παρουσιάζονται αναλυτικά όλες οι μεταβλητές για τις κορυφές, ακμές και επιφάνειες του παραπάνω παραδείγματος.

Vertex	IncidentEdge
v_1	$e_{1,1}$
v_2	$e_{2,1}$
v_3	$e_{3,1}$
v_4	$e_{4,1}$
v_5	$e_{5,1}$
v_6	$e_{6,1}$
v_7	$e_{7,1}$

Πίνακας 5.10: Πίνακας κορυφών

Face	OuterComponent	InnerComponents
f_1	null	$e_{1,2}$
f_2	$e_{1,1}$	$e_{5,1}$
f_3	$e_{5,2}$	null

Πίνακας 5.11: Πίνακας επιφανειών

Edge	Origin	Twin	IncidentFace	Next	Prev
$e_{1,1}$	v_1	$e_{1,2}$	f_2	$e_{2,1}$	$e_{4,1}$
$e_{1,2}$	v_2	$e_{1,1}$	f_1	$e_{4,2}$	$e_{2,2}$
$e_{2,1}$	v_2	$e_{2,2}$	f_2	$e_{3,1}$	$e_{1,1}$
$e_{2,2}$	v_3	$e_{2,1}$	f_1	$e_{1,2}$	$e_{3,2}$
$e_{3,1}$	v_3	$e_{3,2}$	f_2	$e_{4,1}$	$e_{2,1}$
$e_{3,2}$	v_4	$e_{3,1}$	f_1	$e_{2,2}$	$e_{4,2}$
$e_{4,1}$	v_4	$e_{4,2}$	f_2	$e_{1,1}$	$e_{3,1}$
$e_{4,2}$	v_1	$e_{4,1}$	f_1	$e_{3,2}$	$e_{1,2}$
$e_{5,1}$	v_5	$e_{5,2}$	f_2	$e_{6,1}$	$e_{7,1}$
$e_{5,2}$	v_6	$e_{5,1}$	f_3	$e_{7,2}$	$e_{6,2}$
$e_{6,1}$	v_6	$e_{6,2}$	f_2	$e_{7,1}$	$e_{2,1}$
$e_{6,2}$	v_7	$e_{6,1}$	f_3	$e_{5,2}$	$e_{7,2}$
$e_{7,1}$	v_7	$e_{7,2}$	f_2	$e_{5,1}$	$e_{6,1}$
$e_{7,2}$	v_5	$e_{7,1}$	f_3	$e_{6,2}$	$e_{5,2}$

Πίνακας 5.12: Πίνακας ακμών

Σύμφωνα με τα παραπάνω είναι αρκετά εύκολο να εκτελεστούν κάποιες βασικές λειτουργίες. Για παράδειγμα για να βρεθεί το εξωτερικό περίγραμμα μίας επιφάνειας f, ακολουθούνται οι δείκτες $Next(e)$ της ημι-ακμής που δίνεται από το δείκτη $OuterComponent(f)$. Σημειώνεται ότι σε κάποιες εφαρμογές μπορεί να μην είναι απαραίτητο να διατηρούνται ξεχωριστοί τύποι για τις κορυφές ή τις επιφάνειες, καθώς δεν έχουν ιδιαίτερη σημασία. Στις περιπτώσεις αυτές διατηρείται απλά μόνο ο τύπος της ημι-ακμής. Επίσης αρκετές φορές μπορεί να χρειαστεί να προστεθούν επιπλέον πληροφορίες σε κάποιο τύπο για διευκόλυνση της υλοποίησης. Για παράδειγμα οι ημι-ακμές μπορούν να έχουν επιπλέον κάποια μεταβλητή που δηλώνει το βάρος τους.

5.7 Ο Αλγόριθμος Υποδιαίρεσης ενός Πολυγώνου σε y-Μονότονα υποπολύγωνα

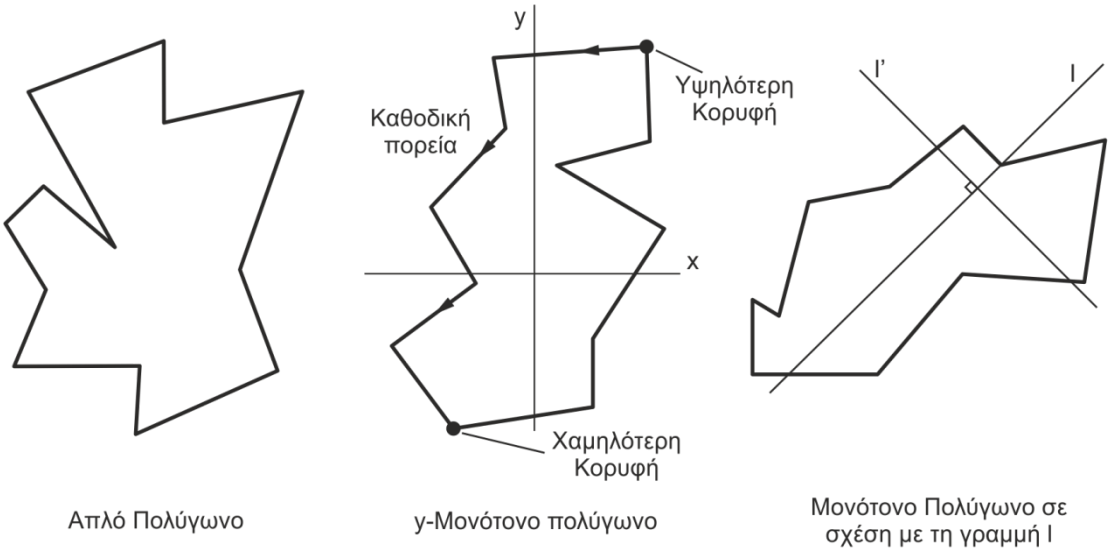
Το τελευταίο στάδιο για την επιτυχή ολοκλήρωση της τομής είναι το κλείσιμο των ανοικτών περιοχών που δημιουργούνται από τον αλγόριθμο της ενότητας 5.5. Με τη βοήθεια των αλγορίθμων που περιγράφηκαν στις ενότητες 5.4 και 5.5, έχουν πλέον σχηματιστεί και αποθηκευτεί όλα τα πολύγωνα επάνω στο επίπεδο τομής, μαζί με τις πιθανές οπές που μπορεί να περιέχουν. Επειδή το δίκτυο γεωμετρίας των μοντέλων αποτελείται από τρίγωνα, γίνεται προφανές ότι ο ανοικτός χώρος που δημιουργείται από την τομή πρέπει να καλυφθεί επίσης με τρίγωνα. Η διαδικασία υποδιαίρεσης ενός πολυγώνου σε τρίγωνα ονομάζεται **τριγωνοποίηση πολυγώνου (Polygon Triangulation)**. Γενικά υπάρχουν πολλοί διαφορετικοί αλγόριθμοι τριγωνοποίησης και επειδή τα είδη πολυγώνων είναι αρκετά, ο καθένας από αυτούς εφαρμόζεται μόνο σε συγκεκριμένα είδη και απαιτεί συγκεκριμένες προϋποθέσεις.

Ο αλγόριθμος τριγωνοποίησης που επιλέχτηκε στην περίπτωση της παρούσας εφαρμογής, απαιτεί τα πολύγωνα προς τριγωνοποίηση να είναι y-μονότονα. Επομένως απαιτείται ένα επιπλέον βήμα πριν εφαρμοστεί ο αλγόριθμος τριγωνοποίησης και αυτό είναι ο διαχωρισμός των πολυγώνων σε y-μονότονα υποπολύγωνα. Ο αλγόριθμος υποδιαίρεσης θα περιγραφεί στην παρούσα ενότητα. Η συγκεκριμένη διαδικασία υποδιαίρεσης και τριγωνοποίησης μπορεί να εφαρμοστεί επιτυχώς και σε περιπτώσεις πολυγώνων με οπές, γι' αυτό και θεωρήθηκε κατάλληλη.

Αρχικά θα δοθεί ο ορισμός ενός y-μονότονου πολυγώνου. Ένα απλό πολύγωνο καλείται **μονότονο** σε σχέση με μία γραμμή l αν για οποιαδήποτε γραμμή l' κάθετη στη γραμμή l τα σημεία τομής του πολυγώνου με τη γραμμή l' είναι το πολύ 2. Έτσι, ένα πολύγωνο που είναι μονότονο σε σχέση με τον y άξονα καλείται **y-μονότονο**. Τα y-μονότονα πολύγωνα έχουν την εξής ιδιότητα:

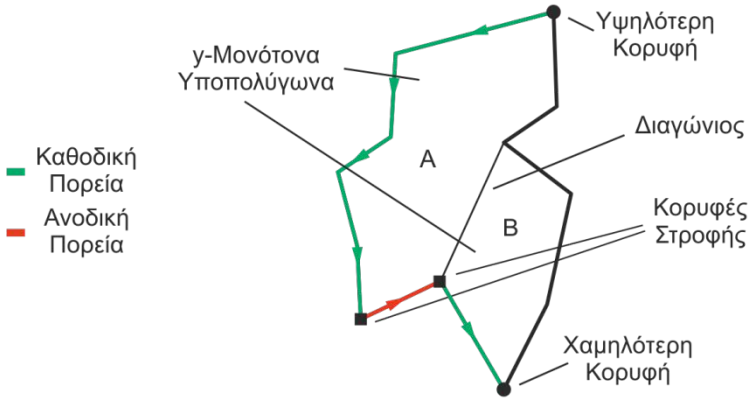
- Με σημείο εκκίνησης την υψηλότερη κορυφή του πολυγώνου και σημείο τερματισμού την αντίστοιχη χαμηλότερη κορυφή του, τότε η πορεία που ακολουθείται κατά μήκος του περιγράμματος του πολυγώνου θα είναι οριζόντια ή καθοδική και ποτέ ανοδική.

Στο σχήμα 5.26 παρουσιάζονται μερικά παραδείγματα μονότονων και μη μονότονων πολυγώνων.



Σχήμα 5.26: Είδη Πολυγώνων

Προκειμένου να χωριστεί ένα πολύγωνο σε y-μονότονα υποπολύγωνα είναι απαραίτητο να απαλειφούν κάποιες κορυφές στροφής. Ως **κορυφή στροφής (turn Vertex)** ορίζεται η κορυφή στην οποία αλλάζει η κατεύθυνση της κίνησης πάνω στο περίγραμμα του πολυγώνου. Δηλαδή μία κορυφή στην οποία μία καθοδική πορεία γίνεται ανοδική, ή μία ανοδική πορεία γίνεται καθοδική. Για κάποια πολύγωνα μερικές κορυφές στροφής είναι προβληματικές και καταργούν την ιδιότητα των y-μονότονων πολυγώνων. Η εξάλειψη των κορυφών αυτών, γίνεται με την προσθήκη διαγωνίων στο εκάστοτε πολύγωνο. Όταν όλες οι κορυφές απαλειφούν, οι διαγώνιες που έχουν προστεθεί δημιουργούν τα όρια των y-μονότονων υποπολυγώνων μέσα στο αρχικό πολύγωνο. Προφανώς οι διαγώνιοι θα πρέπει να προστεθούν με κάποια λογική και όχι εντελώς τυχαία, προκειμένου να υπάρξει ένα ικανοποιητικό αποτέλεσμα. Ο τρόπος προσθήκης διαγωνίων είναι ο βασικός ρόλος του αλγορίθμου και θα προσδιοριστεί στη συνέχεια. Στο σχήμα 5.27 παρουσιάζεται ένα παράδειγμα πολυγώνου με μερικές κορυφές στροφής και μία πιθανή προσθήκη μίας διαγωνίου. Όπως φαίνεται, η διαγώνιος χωρίζει το πολύγωνο σε δύο y-μονότονα υποπολύγωνα.



Σχήμα 5.27: Κορυφές στροφής και προσθήκη διαγωνίου σε ένα τυχαίο πολύγωνο

Στο σχήμα 5.27 φαίνεται ότι κάποιες κορυφές στροφής δεν είναι απαραίτητο να απαλειφούν με προσθήκη διαγωνίων. Σημαντικό ρόλο σε αυτό αποτελεί η θέση των δύο γειτονικών κορυφών μίας κορυφής στροφής.

Έστω οι κορυφές z και w. Τότε η κορυφή z βρίσκεται **κάτω** από την κορυφή w αν και μόνο αν $z_y < w_y$ ή $z_y = w_y$ και $z_x > w_x$. Αντίστοιχα η κορυφή z βρίσκεται **πάνω** από την κορυφή w αν και μόνο αν $z_y > w_y$ ή $z_y = w_y$ και $z_x < w_x$.

Σε ένα πολύγωνο διακρίνονται πέντε είδη κορυφών σε σχέση με τις θέσεις των γειτονικών κορυφών τους, από τα οποία τα τέσσερα είναι κορυφές στροφής. Τα είδη αυτά επεξηγούνται στη συνέχεια:

- **Κορυφή αρχής (Start Vertex)**: Κορυφή στροφής v της οποίας και οι δύο γειτονικές κορυφές βρίσκονται από κάτω της και η εσωτερική γωνία στην κορυφή v είναι μικρότερη από 180^o.

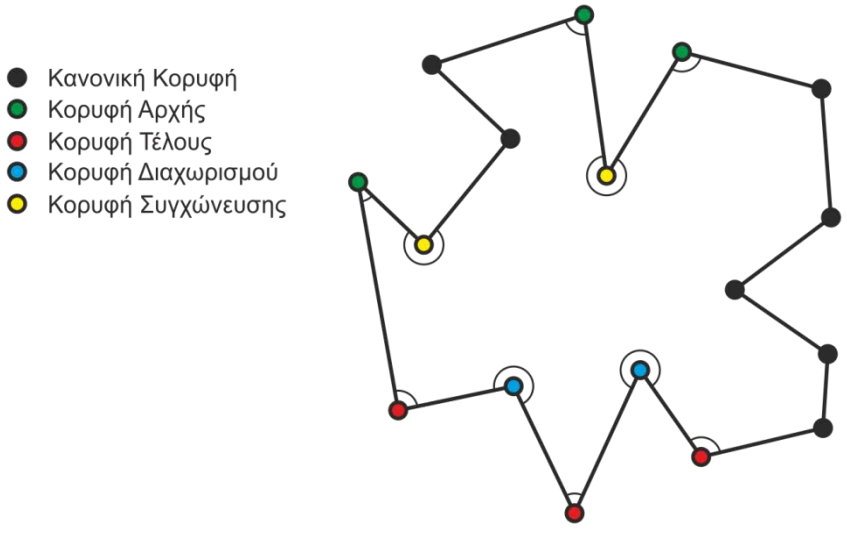
- **Κορυφή διαχωρισμού (Split Vertex)**: Όμοια με την κορυφή αρχής με τη διαφορά ότι η εσωτερική γωνία είναι μεγαλύτερη από 180^o.

- **Κορυφή τέλους (End Vertex)**: Κορυφή στροφής v της οποίας και οι δύο γειτονικές κορυφές βρίσκονται από πάνω της και η εσωτερική γωνία στην κορυφή v είναι μικρότερη από 180^o.

- **Κορυφή συγχώνευσης (Merge Vertex)**: Όμοια με την κορυφή τέλους με τη διαφορά ότι η εσωτερική γωνία είναι μεγαλύτερη από 180^o.

- **Κανονική Κορυφή (Regular Vertex)**: Κορυφή της οποίας η μία γειτονική κορυφή βρίσκεται από κάτω της και η άλλη γειτονική κορυφή βρίσκεται από πάνω της. Αποτελεί το μόνο είδος που δεν είναι κορυφή στροφής.

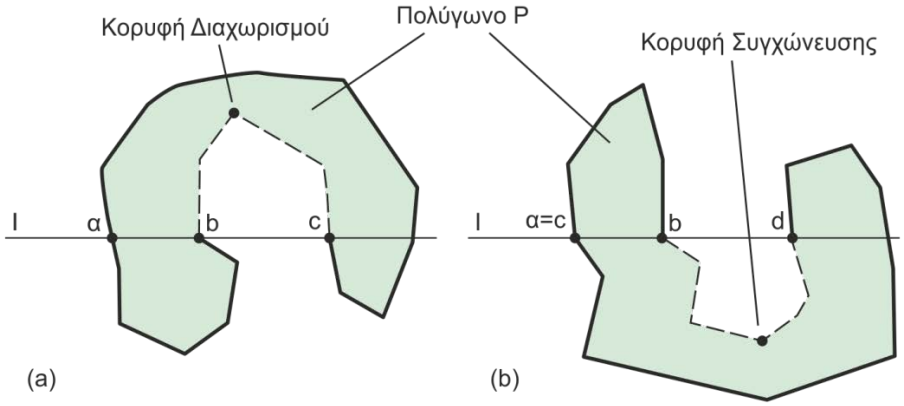
Στο σχήμα 5.28 παρουσιάζεται ένα πολύγωνο πάνω στο οποίο σημειώνονται τα είδη κορυφών του καθώς και οι αντίστοιχες εσωτερικές γωνίες κάθε κορυφής στροφής.



Σχήμα 5.28: Είδη Κορυφών

Σύμφωνα με το σχήμα 5.28 οι κορυφές διαχωρισμού και συγχώνευσης αποτελούν σημεία μη μονοτονικότητας (**non-monotonicity**). Δηλαδή, ένα πολύγωνο είναι y-μονότονο αν και μόνο αν δεν έχει κορυφές διαχωρισμού και κορυφές συγχώνευσης.

Απόδειξη: Έστω το πολύγωνο P το οποίο δεν είναι y-μονότονο. Αρκεί να αποδειχθεί ότι το P περιέχει τουλάχιστον μία κορυφή διαχωρισμού ή μία κορυφή συγχώνευσης. Εφόσον το P δεν είναι y-μονότονο, τότε υπάρχει τουλάχιστον μία οριζόντια γραμμή l η οποία έχει τουλάχιστον 3 σημεία τομής με το πολύγωνο P. Έστω a το 1ο σημείο τομής και b το 2ο σημείο τομής της γραμμής με το πολύγωνο όπως φαίνεται στο σχήμα 5.29.



Σχήμα 5.29: Περιπτώσεις σημείων τομής με ευθεία

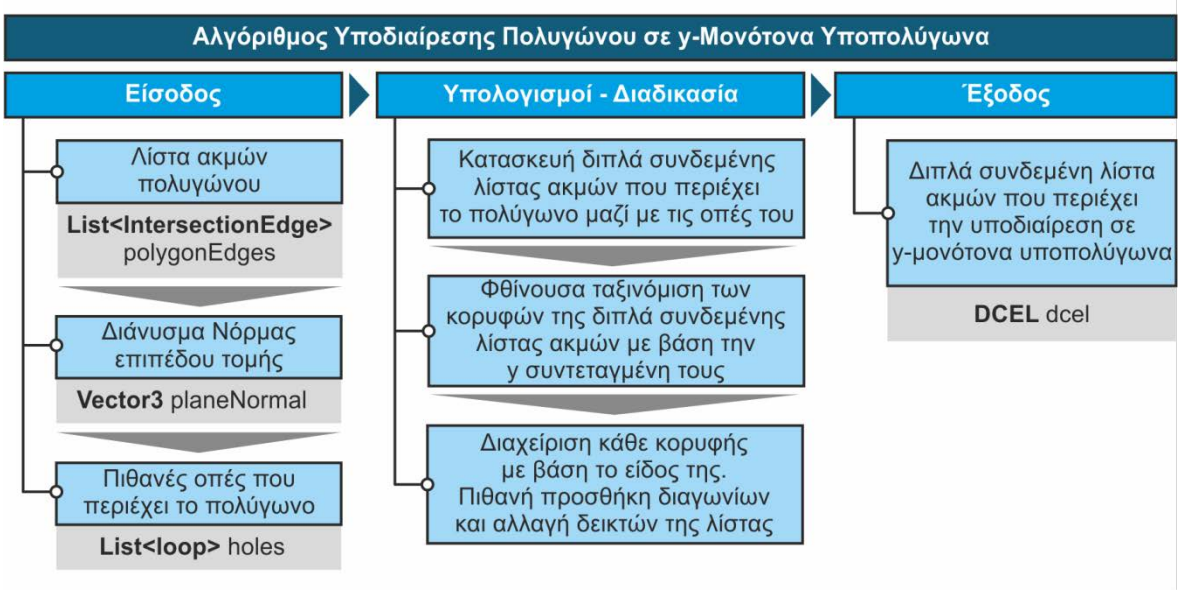
Με αρχή το σημείο τομής b ακολουθείται το περίγραμμα του πολυγώνου P, έτσι ώστε το εσωτερικό του πολυγώνου να βρίσκεται στα αριστερά του περιγράμματος και αναζητείται το επόμενο σημείο τομής της ευθείας με το περίγραμμα αυτό. Δηλαδή το περίγραμμα που ακολουθείται βρίσκεται πάνω από το σημείο b. Έστω c το νέο σημείο τομής. Αν $a \neq c$ τότε η υψηλότερη κορυφή του περιγράμματος θα πρέπει να είναι κορυφή διαχωρισμού όπως απεικονίζεται στο σχήμα 5.29 (a).

Διαφορετικά αν $a = c$, τότε ακολουθείται το περίγραμμα που βρίσκεται κάτω από το σημείο b όπως φαίνεται στο σχήμα 5.29 (b). Επειδή η ευθεία l έχει τουλάχιστον 3 σημεία τομής με το πολύγωνο P, τότε θα πρέπει να βρεθεί ένα ακόμα σημείο τομής d για το οποίο ισχύει $d \neq a$. Αυτό σημαίνει ότι η χαμηλότερη κορυφή του περιγράμματος που ακολουθήθηκε πρέπει να είναι κορυφή συγχώνευσης.

Συνεπώς έχει αποδειχθεί ότι ένα πολύγωνο διαχωρίζεται επιτυχώς σε y-μονότονα υποπολύγωνα, εφόσον έχουν απαλειφτεί από αυτό όλες οι κορυφές διαχωρισμού και συγχώνευσης. Η απαλοιφή γίνεται με την προσθήκη διαγωνίων κατεύθυνσης προς τα πάνω για τις κορυφές διαχωρισμού και κατεύθυνσης προς τα κάτω για τις κορυφές συγχώνευσης. Προφανώς οι διαγώνιες δεν πρέπει να έχουν σημεία τομής μεταξύ τους.

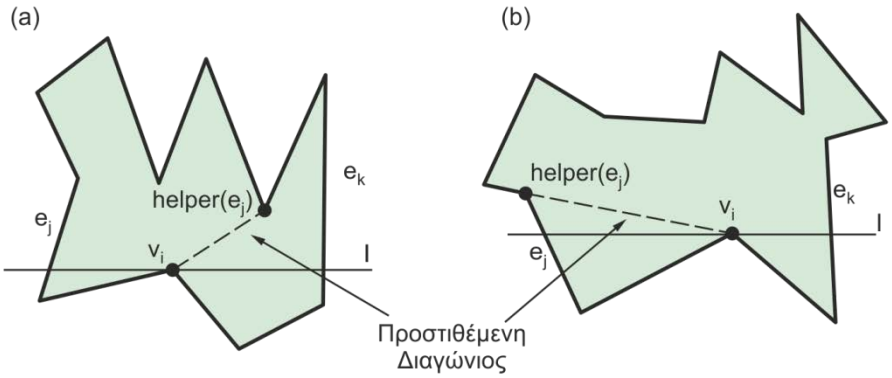
Η βασική ιδέα του αλγορίθμου είναι η σάρωση του πολυγώνου με μία φανταστική οριζόντια γραμμή. Η γραμμή αυτή σαρώνει το πολύγωνο από πάνω προς τα κάτω και σταματά όταν συναντά κάποια κορυφή του πολυγώνου. Ανάλογα το είδος της κορυφής, καλείται και μία κατάλληλη συνάρτηση που τη διαχειρίζεται. Για δύο κορυφές με ίδια συντεταγμένη στον y άξονα, η κορυφή που βρίσκεται πιο αριστερά στον x άξονα θα έχει μεγαλύτερη προτεραιότητα. Για την αποτελεσματική διαχείριση του πολυγώνου και την εύκολη προσθήκη διαγωνίων όποτε αυτό απαιτείται, το πολύγωνο αποθηκεύεται σε μία διπλά συνδε-

μένη λίστα ακμών. Συνοπτικά η λειτουργία του αλγορίθμου παρουσιάζεται στο σχήμα 5.30.



Σχήμα 5.30: Διάγραμμα Αλγορίθμου Υποδιαίρεσης Πολυγώνου σε y-Μονότονα Υποπολύγωνα

Αρχικά θα εξηγηθεί η λογική προσθήκης διαγωνίων για τις κορυφές διαχωρισμού. Όπως προαναφέρθηκε η διαγώνιος που θα προστεθεί θα πρέπει να ενωθεί από την κορυφή διαχωρισμού προς μία κορυφή που βρίσκεται υψηλότερα από αυτήν. Προφανώς μπορεί να υπάρχουν αρκετές τέτοιες κορυφές οπότε η επιλογή δεν πρέπει να είναι τυχαία. Έστω v_i η κορυφή διαχωρισμού, l η οριζόντια γραμμή σάρωσης του πολυγώνου, e_j η ακμή που βρίσκεται ακριβώς αριστερά από την v_i και e_k η ακμή που βρίσκεται ακριβώς δεξιά από την v_i, όπως φαίνεται στο σχήμα 5.31.

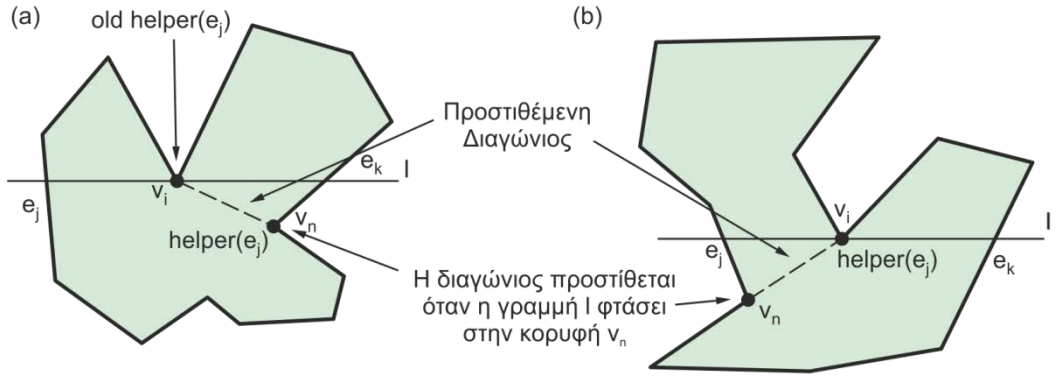


Σχήμα 5.31: Προσθήκη διαγωνίου για μία κορυφή διαχωρισμού

Η χαμηλότερη κορυφή ανάμεσα στην e_j και e_k που βρίσκεται πάνω από την v_i, ορίζεται ως κορυφή $helper(e_j)$, όπως φαίνεται στο σχήμα 5.31 (a). Η διαγώνιος που προστίθεται μπορεί πάντα να ενώνεται με την κορυφή $helper(e_j)$. Σημειώνεται ότι μπορεί να μην υπάρχει χαμηλότερη κορυφή ανάμεσα στις ακμές e_j και e_k. Στην περίπτωση αυτή ως $helper(e_j)$ ορίζεται το πάνω άκρο της κορυφής e_j,όπως φαίνεται στο σχήμα 5.31 (b).

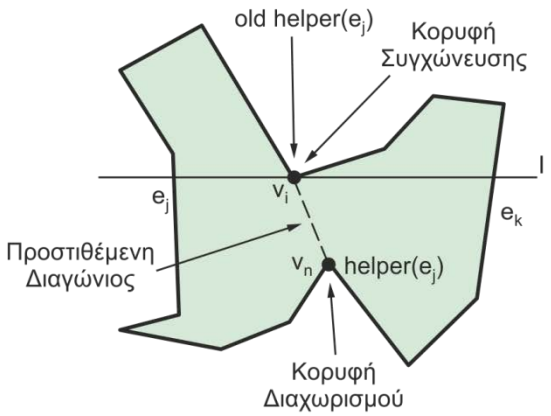
Εφόσον ο τρόπος απαλοιφής των κορυφών διαχωρισμού είναι πλέον γνωστός, οι κορυφές που μένει ακόμα να απαλειφτούν είναι οι κορυφές συγχώνευσης. Για μία κορυφή συγχώνευσης, η διαγώνιος που πρέπει να προστεθεί ενώνεται με μία κορυφή που βρίσκεται χαμηλότερα από αυτήν. Το πρόβλημα σε αυτή την περίπτωση είναι ότι οι κορυφές αυτές δεν είναι γνωστές, καθώς η γραμμή σάρωσης του πολυγώνου βρίσκεται στην κορυφή συγχώνευσης. Συνεπώς η διαγώνιος δεν μπορεί να προστεθεί σε αυτό το σημείο. Το πρόβλημα αυτό λύνεται με τον τρόπο που θα περιγραφεί στη συνέχεια.

Έστω ότι η γραμμή σάρωσης σταματά στην κορυφή συγχώνευσης v_i, l η οριζόντια γραμμή σάρωσης του πολυγώνου, e_j η ακμή που βρίσκεται ακριβώς αριστερά από την v_i και e_k η ακμή που βρίσκεται ακριβώς δεξιά από την v_i, όπως φαίνεται στο σχήμα 5.32.



Σχήμα 5.32: Προσθήκη διαγωνίου για μία κορυφή συγχώνευσης

Για την ακμή e_j η κορυφή v_i ορίζεται ως η νέα κορυφή $helper(e_j)$. Η διαγώνιος θα πρέπει να προστεθεί στην υψηλότερη κορυφή κάτω από την v_i και ανάμεσα στις ακμές e_j και e_k, δηλαδή ακριβώς αντίθετα από τη λογική που εφαρμόστηκε στην κορυφή διαχωρισμού. Η υψηλότερη αυτή κορυφή δεν είναι ακόμα γνωστή καθώς η γραμμή σάρωσης βρίσκεται στην κορυφή v_i. Επομένως μόλις η γραμμή σάρωσης συναντήσει μία κορυφή v_n η οποία αντικαθιστά την κορυφή v_i ως $helper(e_j)$, τότε αυτή είναι η υψηλότερη κορυφή που αναζητείται. Οπότε κάθε φορά που αντικαθιστάται η κορυφή $helper(e_j)$, ελέγχεται εάν η προηγούμενη $helper(e_j)$ είναι κορυφή συγχώνευσης. Εάν αυτό ισχύει τότε προστίθεται μία διαγώνιος μεταξύ της προηγούμενης και της νέας κορυφής $helper(e_j)$, όπως φαίνεται στο σχήμα 5.32 (a). Σημειώνεται ότι στην περίπτωση που η κορυφή $helper(e_j)$ παραμείνει η κορυφή v_i, τότε η διαγώνιος προστίθεται μεταξύ της κορυφής v_i και του κάτω άκρου της ακμής e_j όπως φαίνεται στο σχήμα 5.32 (b). Τέλος εάν η κορυφή v_n που αντικαταστήσει την κορυφή $helper(e_j)$ τυχαίνει και είναι κορυφή διαχωρισμού, τότε απαλείφονται ταυτόχρονα δύο κορυφές με τη χρήση μόνο μίας διαγωνίου, όπως φαίνεται στο παράδειγμα του σχήματος 5.33.



Σχήμα 5.33: Απαλοιφή δύο κορυφών με προσθήκη μίας διαγωνίου

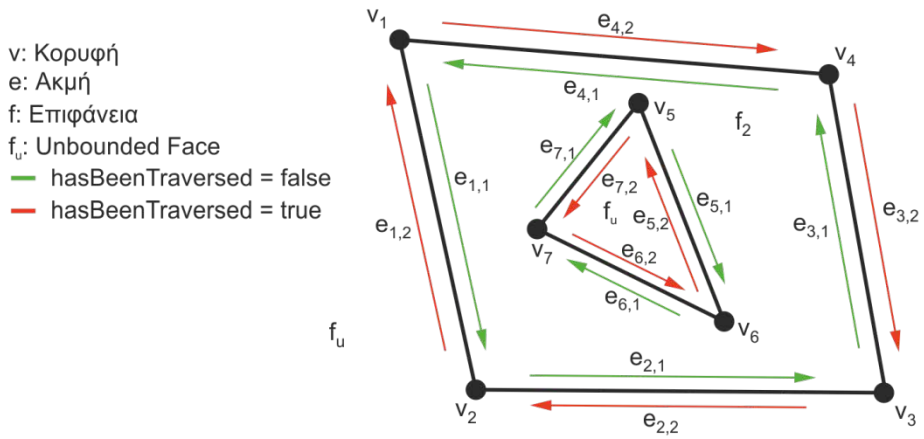
5.7.1 Υλοποίηση στη Unity

Η υλοποίηση του παραπάνω αλγορίθμου γίνεται στην συνάρτηση **TriangulateMonotone**, η οποία βρίσκεται στην κλάση **Triangulator**. Τα ορίσματα της συνάρτησης αναφέρονται στο σχεδιάγραμμα του σχήματος 5.30. Το πολύγωνο δίνεται ως μία λίστα ακμών αριστερόστροφης φοράς ενώ οι πιθανές οπές του έχουν δεξιόστροφη φορά όπως φαίνεται στο σχήμα 5.20. Οι οπές αυτές δίνονται επίσης σε αντίστοιχες λίστες.

Αρχικά θα πρέπει να κατασκευαστεί μία διπλά συνδεμένη λίστα ακμών που περιέχει το πολύγωνο και τις οπές του. Αυτό γίνεται μέσω της συνάρτησης **ConstructDCEL.** Για την αποτελεσματική διαχείριση όλων των πολυγώνων στον συγκεκριμένο αλγόριθμο, θα πρέπει αυτά να μετασχηματιστούν μαζί με τις οπές τους στο επίπεδο xy. Αυτό γίνεται όπως ακριβώς έχει περιγραφεί στην ενότητα 5.5.1. Σημειώνεται ότι οι αρχικές θέσεις των κορυφών στον τρισδιάστατο χώρο διατηρούνται στην διπλά συνδεμένη λίστα ακμών, καθώς θα χρειαστούν μετά το πέρας του αλγορίθμου για την επανατοποθέτηση του πολυγώνου στη αρχική θέση που βρισκόταν.

Η κατασκευή της διπλά συνδεμένης λίστας ακμών γίνεται, προσθέτοντας σε αυτή πρώτα το ίδιο το πολύγωνο και έπειτα τις πιθανές οπές του. Για κάθε πολύγωνο ή οπή, προσθέτονται στη λίστα οι ακμές τους με τη σειρά που δίνονται. Για κάθε ακμή προσθέτονται 2 ημι-ακμές καθώς και οι κορυφές αυτών. Επίσης γίνεται κατάλληλη αρχικοποίηση των δεικτών κάθε ημι-ακμής και κορυφής, όπως αυτοί έχουν περιγραφεί στην ενότητα 5.6. Επιπλέον σε κάθε ημι-ακμή διατηρείται ένας δείκτης για την κορυφή helper, καθώς και η μεταβλητή **hasBeenTraversed** που δηλώνει αν η ημι-ακμή αυτή έχει διασχιστεί. Η μεταβλητή αυτή αρχικοποιείται με την τιμή **false** και χρησιμοποιείται για την εύκολη πρόσβαση στα y-μονότονα πολύγωνα που θα δημιουργηθούν μετά το πέρας του αλγορίθμου. Εξαιρούνται οι ημι-ακμές που δείχνουν στο unbounded Face και δεν θα χρειαστεί ποτέ να διασχιστούν. Στις ακμές αυτές η μεταβλητή αρχικοποιείται με την τιμή **true**.

Στο σχήμα 5.34 παρουσιάζεται ένα παράδειγμα πολυγώνου με μία οπή, για το οποίο έχει κατασκευαστεί μία διπλά συνδεμένη λίστα ακμών. Με πράσινο σημειώνονται οι ημι-ακμές που δεν έχουν διασχιστεί ενώ με κόκκινο σημειώνονται οι ημι-ακμές που θεωρείται ότι έχουν διασχιστεί.



Σχήμα 5.34: Παράδειγμα πολυγώνου με μία οπή, μετά την τοποθέτηση του σε μία διπλά συνδεμένη λίστα ακμών

Επειδή όλα τα y-μονότονα πολύγωνα θα σχηματιστούν στο εσωτερικό του πολυγώνου και εξωτερικά της οπής, οι κόκκινες ημι-ακμές δεν θα αποτελέσουν ποτέ μέρος των πολυγώνων αυτών. Οπότε θεωρείται εξαρχής ότι έχουν διασχιστεί.

Εφόσον η διπλά συνδεμένη λίστα έχει κατασκευαστεί, στη συνέχεια εφαρμόζεται μία φθίνουσα ταξινόμηση των κορυφών της με βάση τη θέση τους στον y άξονα. Στη συνέχεια κάθε κορυφή εξετάζεται ξεχωριστά ξεκινώντας από αυτήν με την υψηλότερη y-συντεταγμένη. Για την εξέταση κάθε κορυφής είναι απαραίτητο να είναι γνωστό το είδος αυτής. Επομένως, αρχικά καλείται μία συνάρτηση η οποία προσδιορίζει το είδος της κορυφής που δίνεται.

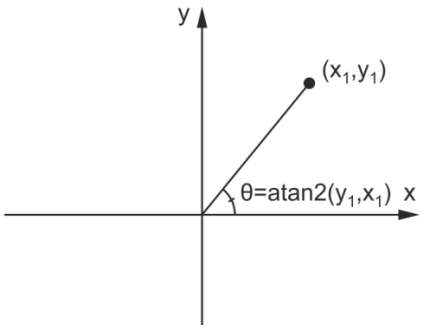
Για τον προσδιορισμό του είδους μίας κορυφής, απαιτούνται οι θέσεις των γειτονικών κορυφών της καθώς και η εσωτερική γωνία του πολυγώνου που σχηματίζεται στην κορυφή αυτήν. Οι θέσεις των γειτονικών κορυφών μπορούν εύκολα να βρεθούν με απλή χρήση των δεικτών της διπλά συνδεμένης λίστας ακμών. Αυτό παρουσιάζεται στον κώδικα του πίνακα 5.13. Η κορυφή που εξετάζεται είναι η μεταβλητή **curVertex**.

Εύρεση γειτονικών κορυφών μίας κορυφής

```
Vertex nextNeighbor = curVertex.GetIncidentEdge ().GetNext ().GetOrigin ();
Vertex prevNeighbor = curVertex.GetIncidentEdge ().GetPrev ().GetOrigin ();
```

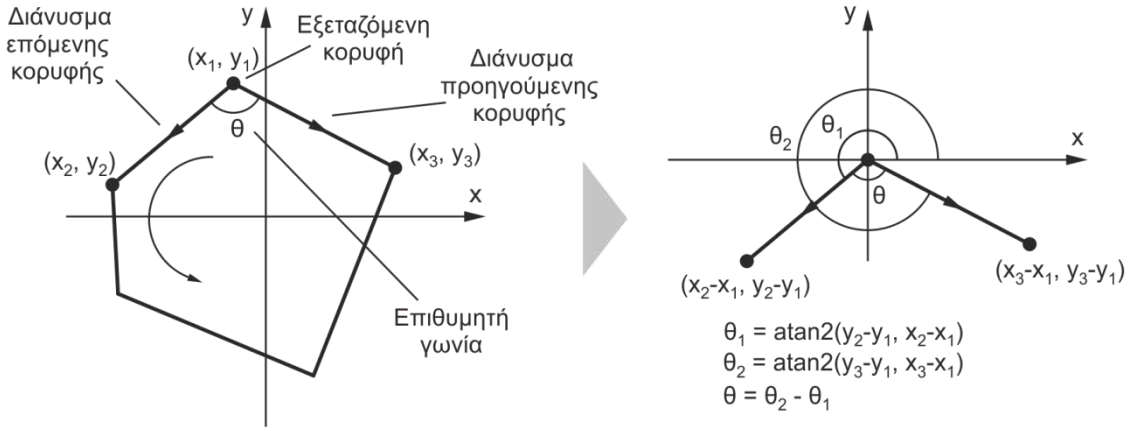
Πίνακας 5.13: Κώδικας εύρεσης γειτονικών κορυφών με κατάλληλη χρήση των δεικτών που παρέχει η διπλά συνδεμένη λίστα ακμών

Αν οι δύο γειτονικές κορυφές βρίσκονται και οι δύο υψηλότερα ή χαμηλότερα από την εξεταζόμενη κορυφή τότε πρέπει να βρεθεί η εσωτερική γωνία του πολυγώνου στην κορυφή αυτή. Αυτό γίνεται με τη χρήση της συνάρτησης Atan2(y,x). Η συνάρτηση αυτή επιστρέφει τη γωνία ανάμεσα στον θετικό x-άξονα και ένα διάνυσμα προς το σημείο (x,y) που δίνεται ως είσοδος. Η γωνία θ που επιστρέφεται βρίσκεται στο εύρος $-\pi < \theta \leq \pi$. Αν η επιστρεφόμενη γωνία είναι αρνητική τότε προστίθενται σε αυτή 360°. Το αποτέλεσμα της συνάρτησης παρουσιάζεται στο σχήμα 5.35.



Σχήμα 5.35: Αποτέλεσμα συνάρτησης Atan2

Η επιθυμητή γωνία μία τυχαίας κορυφής σε ένα πολύγωνο παρουσιάζεται στο σχήμα 5.36, μαζί με τη διαδικασία υπολογισμού της. Για την εύρεση της γωνίας αυτής, αφαιρούνται δύο γωνίες που επιστρέφει η συνάρτηση Atan2 για τα διανύσματα της προηγούμενης και επόμενης κορυφής.



Σχήμα 5.36: Υπολογισμός εσωτερικής γωνίας για μία κορυφή ενός πολυγώνου

Ο παραπάνω υπολογισμός παρουσιάζεται σε μορφή κώδικα στον πίνακα 5.14.

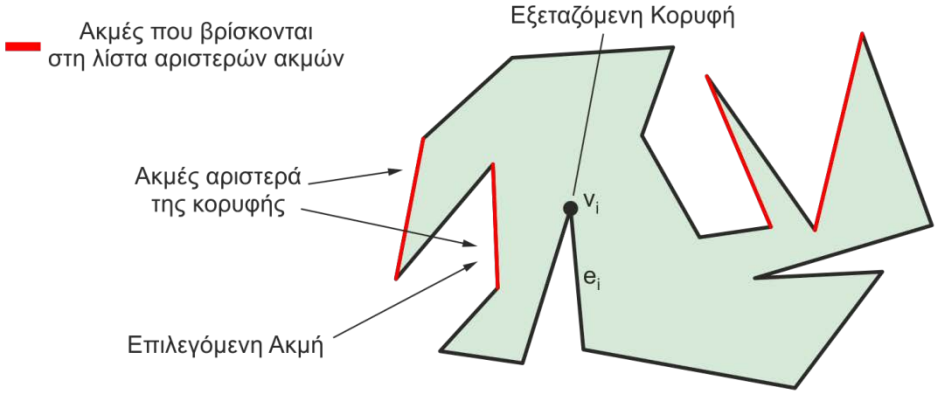
Υπολογισμός Εσωτερικής Γωνίας σε μία Κορυφή

```
        //Vector which has a direction from current Vertex to previous Vertex;
        Vector2 prevDirectedVector = prevNeighbor.GetPosition () -  curVertex.GetPosition ();
        //Vector which has a direction from current Vertex to next Vertex;
        Vector2 nextDirectedVector = nextNeighbor.GetPosition () - curVertex.GetPosition ();

        //Find directed angle from nextDirectedVector to prevDirectedVector (interior angle of
polygon)
        float angle = Mathf.Atan2 (prevDirectedVector.y, prevDirectedVector.x) -
                      Mathf.Atan2 (nextDirectedVector.y, nextDirectedVector.x);
        if (angle < 0) {
            angle += 2 * Mathf.PI;
        }
        angle = angle * Mathf.Rad2Deg;
```

Πίνακας 5.14: Κώδικας υπολογισμού εσωτερικής γωνίας για μία κορυφή ενός πολυγώνου

Εφόσον όλη η απαιτούμενη πληροφορία είναι πλέον διαθέσιμη, βρίσκεται εύκολα το είδος της εξεταζόμενης κορυφής. Τα είδη των κορυφών αναφέρονται συνοπτικά στην αρχή του κεφαλαίου 5.7.

Μία επιπλέον λειτουργία που πρέπει να υλοποιηθεί, είναι η εύρεση μίας ακμής που βρίσκεται ακριβώς αριστερά από μία κορυφή. Για τη λειτουργία αυτή διατηρούνται όλες οι υποψήφιες ακμές σε μία λίστα, που στον παρών αλγόριθμο καλείται ως **λίστα αριστερών ακμών**. Στη λίστα αυτή προσθέτονται μόνο οι ακμές που έχουν στα δεξιά τους το εσωτερικό του πολυγώνου. Επομένως όποτε χρειαστεί να βρεθεί μία τέτοια ακμή γίνεται αναζήτηση μόνο στη συγκεκριμένη λίστα, αποφεύγοντας την αναζήτηση σε όλες τις ακμές του πολυγώνου. Για την εύρεση της κατάλληλης ακμής αρχικά αναζητούνται οι ακμές από την λίστα αριστερών ακμών που βρίσκονται αριστερά της εξεταζόμενης κορυφής, όπως έχει περιγραφεί στην ενότητα 5.5.1. Στη συνέχεια από τις ακμές αυτές, επιλέγεται εκείνη που βρίσκεται πιο δεξιά η οποία είναι και η ζητούμενη ακμή. Η διαδικασία παρουσιάζεται συνοπτικά στο σχήμα 5.37.



Σχήμα 5.37: Εύρεση ακμής που βρίσκεται ακριβώς αριστερά από μία κορυφή

Για κάθε είδος κορυφής καλείται κατάλληλη συνάρτηση, η οποία εφαρμόζει τις απαραίτητες αλλαγές που πρέπει να γίνουν στην διπλά συνδεμένη λίστα ακμών. Αυτές περιλαμβάνουν αλλαγές δεικτών της λίστας, ενημέρωση κορυφών τύπου helper ή προσθήκη διαγωνίων.

Στο σχήμα 5.38 παρουσιάζονται σε μορφή ψευδοκώδικα οι συναρτήσεις που καλούνται για κάθε είδος κορυφής και οι αλλαγές που αυτές εφαρμόζουν.

ΔιαχείρισηΚορυφήςΑρχής (v_i)
1. Εισαγωγή ακμής e_i στην λίστα αριστερών ακμών
   και ορισμός της κορυφής helper(e_i) ως v_i

ΔιαχείρισηΚορυφήςΤέλους (v_i)
1. **Αν** helper(e_{i-1}) είναι κορυφή συγχώνευσης
2. **Τότε** Εισαγωγή διαγωνίου από την v_i προς την helper(e_{i-1})
3. Διαγραφή της ακμής e_{i-1} από την λίστα αριστερών ακμών

ΔιαχείρισηΚορυφήςΔιαχωρισμού (v_i)
1. Εύρεση ακμής e_j στα αριστερά της v_i από την λίστα αριστερών ακμών
2. Εισαγωγή διαγωνίου από την v_i προς την helper(e_j)
3. helper(e_j) = v_i
4. Εισαγωγή ακμής e_i στην λίστα αριστερών ακμών
   και ορισμός της κορυφής helper(e_i) ως v_i

ΔιαχείρισηΚορυφήςΣυγχώνευσης (v_i)

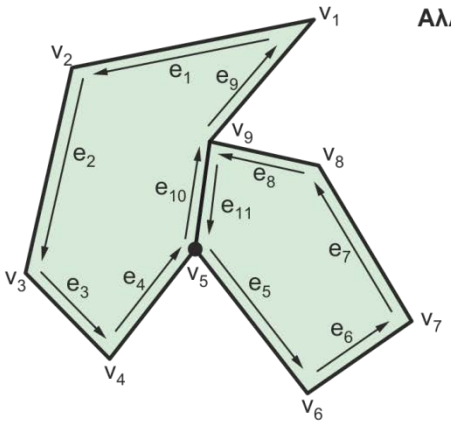
1. **Αν** helper(e_{i-1}) είναι κορυφή συγχώνευσης
2. **Τότε** Εισαγωγή διαγωνίου από την v_i προς την helper(e_{i-1})
3. Διαγραφή της ακμής e_{i-1} από την λίστα αριστερών ακμών
4. Εύρεση ακμής e_j στα αριστερά της v_i από την λίστα αριστερών ακμών
5. **Αν** helper(e_j) είναι κορυφή συγχώνευσης
6. **Τότε** Εισαγωγή διαγωνίου από την v_i προς την helper(e_j)

ΔιαχείρισηΚανονικήςΚορυφής (v_i)

1. **Αν** το εσωτερικό του πολυγώνου βρίσκεται δεξία της v_i
2. **Τότε αν** η κορυφή helper(e_{i-1}) είναι κορυφή συγχώνευσης
3. **Τότε** Εισαγωγή διαγωνίου από την v_i προς την helper(e_{i-1})
4. Διαγραφή της ακμής e_{i-1} από την λίστα αριστερών ακμών
5. Εισαγωγή ακμής e_i στην λίστα αριστερών ακμών
   και ορισμός της κορυφής helper(e_i) ως v_i
6. **Αλλιώς** Εύρεση ακμής e_j στα αριστερά της v_i από την λίστα αριστερών ακμών
7. **Αν** η κορυφή helper(e_j) είναι κορυφή συγχώνευσης
8. **Τότε** Εισαγωγή διαγωνίου από την v_i προς την
9. helper(e_j) = v_i

Σχήμα 5.38: Ψευδοκώδικες διαχείρισης για κάθε είδος κορυφής

Στους παραπάνω ψευδοκώδικες επισημαίνεται ότι $e_i = v_i.IncidentEdge()$ και $e_{i-1} = e_i.Prev()$. Επιπλέον αναφέρεται συχνά η λειτουργία εισαγωγής μίας διαγωνίου. Στο σχήμα 5.39 παρουσιάζονται οι νέες ημι-ακμές και οι αλλαγές που πρέπει να γίνουν στους δείκτες της διπλά συνδεμένης λίστας ακμών για την ορθή εισαγωγή μίας διαγωνίου.

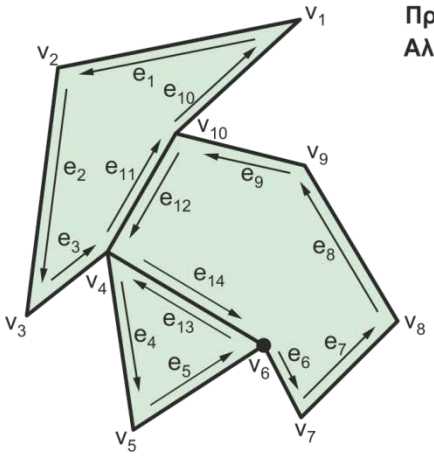


Προσθήκη Διαγωνίου από την κορυφή v_5 προς την v_9
Αλλαγές δεικτών με τη σειρά που αυτές εφαρμόζονται

e_{10}.Origin = v_5
e_{10}.Twin = e_{11}
e_{10}.Prev = v_5.IncidentEdge.Prev
e_{11}.Origin = v_9
e_{11}.Twin = e_{10}
e_{11}.Next = v_5.IncidentEdge
e_{10}.Next = v_9.IncidentEdge
e_{11}.Prev = v_9.IncidentEdge.Prev
v_9.IncidentEdge.Prev.Next = e_{11}
v_9.IncidentEdge.Prev = e_{10}
v_5.IncidentEdge.Prev.Next = e_{10}
v_5.IncidentEdge.Prev = e_{11}
v_5.IncidentEdge = e_{10}

Σχήμα 5.39: Προσθήκη διαγωνίου και απαραίτητες αλλαγές δεικτών

Σημειώνεται ότι η τελευταία αλλαγή στο παραπάνω παράδειγμα, αλλάζει την προσκείμενη ακμή $IncidentEdge(v_i)$ της κορυφής του άκρου αρχής της διαγωνίου. Η αλλαγή αυτή εξασφαλίζει την ορθή εισαγωγή μελλοντικών διαγωνίων στο πολύγωνο. Η προσθήκη μίας διαγωνίου με αλλαγές δεικτών όπως αυτές παρουσιάζονται στο σχήμα 5.56, είναι σωστή για την πλειοψηφία των περιπτώσεων. Υπάρχει όμως και μία ειδική περίπτωση προσθήκης διαγωνίου, όπου οι αλλαγές δεικτών παρουσιάζουν κάποιες διαφοροποιήσεις. Η περίπτωση αυτή αφορά την προσθήκη διαγωνίου από μία οποιαδήποτε κορυφή προς μία κορυφή διαχωρισμού, ενώ το άκρο αρχής της διαγωνίου βρίσκεται δεξιά από το άκρο τέλους. Στο σχήμα 5.40 παρουσιάζεται ένα παράδειγμα μίας τέτοιας περίπτωσης. Με κόκκινο σημειώνονται οι διαφοροποιήσεις δεικτών από μία κλασσική προσθήκη διαγωνίου.



Προσθήκη Διαγωνίου από την κορυφή v_6 προς την v_4
Αλλαγές δεικτών με τη σειρά που αυτές εφαρμόζονται

e_{13}.Origin = v_6
e_{13}.Twin = e_{11}
e_{13}.Prev = v_6.IncidentEdge.Prev
e_{14}.Origin = v_4
e_{14}.Twin = e_{13}
e_{14}.Next = v_6.IncidentEdge
e_{13}.Next = v_4.IncidentEdge.Twin.Next
e_{14}.Prev = v_4.IncidentEdge.Twin
v_4.IncidentEdge.Twin.Next.Prev = e_{13}
v_4.IncidentEdge.Twin.Next = e_{14}
v_6.IncidentEdge.Prev.Next = e_{13}
v_6.IncidentEdge.Prev = e_{14}
v_6.IncidentEdge = e_{13}

Σχήμα 5.40: Ειδική περίπτωση προσθήκης διαγωνίου και απαραίτητες αλλαγές δεικτών

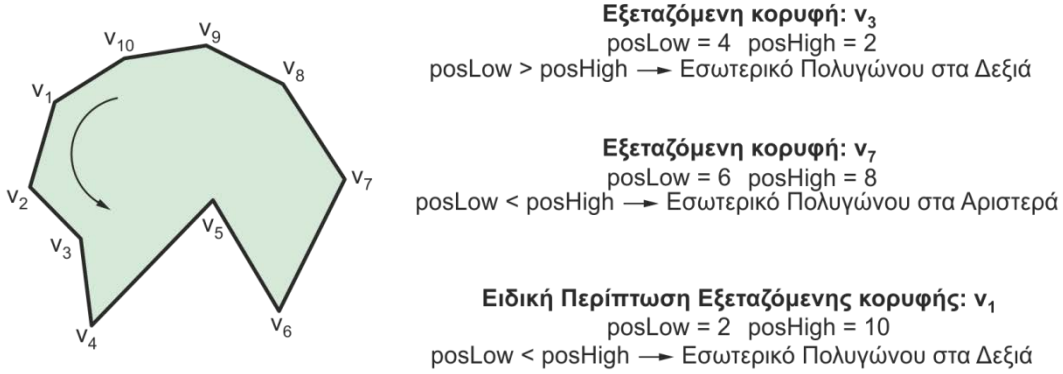
Στο παραπάνω παράδειγμα έχει ήδη προστεθεί μία διαγώνιος στο πολύγωνο και συγκεκριμένα αυτή που ενώνει τις ακμές v_4 και v_5. Εξαιτίας αυτής, για την κορυφή v_4 θα ισχύει ότι $IncidentEdge(v_4) = e_{11}$ και όχι $IncidentEdge(v_4) = e_4$ όπως αναμένεται. Συνεπώς η προσθήκη διαγωνίου όπως αυτή αναφέρεται στο σχήμα 5.39, θα έχει ως αποτέλεσμα λανθασμένη ανάθεση δεικτών.

Μία τελευταία λειτουργία που πρέπει να εξηγηθεί είναι ο τρόπος εύρεσης της θέσης μίας **κανονικής κορυφής** σε σχέση με το εσωτερικό του πολυγώνου. Η λειτουργία αυτή χρησιμοποιείται στη συνάρτηση διαχείρισης μίας κανονικής κορυφής, όπως αναφέρεται στον αντίστοιχο ψευδοκώδικα του σχήματος 5.38. Όπως έχει αναφερθεί νωρίτερα, το πολύγωνο δίνεται με αριστερόστροφη φορά των ακμών του και συνεπώς των κορυφών του. Για την εύρεση της θέσης του εσωτερικού του πολυγώνου σχετικά με μία κορυφή, γίνεται εκμετάλλευση της θέσης αυτής και των γειτονικών κορυφών της στον πίνακα κορυφών. Συγκεκριμένα αρχικά πρέπει να προσδιοριστεί ποια από τις γειτονικές κορυφές της εξεταζόμενης κορυφής βρίσκεται χαμηλότερα στον y-άξονα και ποια υψηλότερα. Η πληροφορία αυτή προσδιορίζεται εύκολα με μία απλή σύγκριση των y-συντεταγμένων των γειτονικών κορυφών. Η εξεταζόμενη κορυφή καθώς και οι γειτονικές κορυφές έχουν τοποθετηθεί σε διαδοχικές θέσεις στον πίνακα κορυφών της διπλά συνδεμένης λίστας ακμών. Επομένως συγκρίνοντας τις θέσεις του πίνακα της χαμηλότερης και της υψηλότερης γειτονικής κορυφής, προσδιορίζεται αν το εσωτερικό του πολυγώνου βρίσκεται αριστερά η δεξιά της εξεταζόμενης κορυφής. Συγκεκριμένα, αν $posLow$ και $posHigh$ είναι αντίστοιχα οι θέσεις της χαμηλότερης και υψηλότερης γειτονικής κορυφής στον πίνακα κορυφών τότε ισχύει ότι:

$$\text{Εσωτερικό Πολυγώνου Δεξιά: } posLow > posHigh$$
$$\text{Εσωτερικό Πολυγώνου Αριστερά: } posLow < posHigh$$

Οι παραπάνω σχέσεις ισχύουν για όλες τις κανονικές κορυφές. Μοναδική εξαίρεση αποτελούν οι κανονικές κορυφές που βρίσκονται στην πρώτη ή στην τελευταία θέση του πίνακα κορυφών. Στην περίπτωση αυτή η φορά για τις παραπάνω ανισότητες αντιστρέφεται.

Στο σχήμα 5.41 παρουσιάζεται ένα παράδειγμα προσδιορισμού της θέσης του εσωτερικού του πολυγώνου. Στο παράδειγμα αυτό θεωρείται ότι οι δείκτες των κορυφών, αντιστοιχούν και στη θέση τους στον αντίστοιχο πίνακα κορυφών στη διπλά συνδεμένη λίστα ακμών.

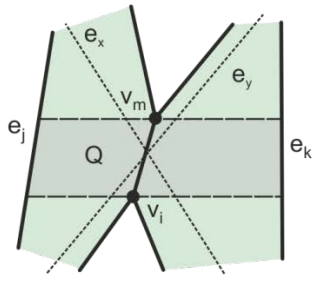


Σχήμα 5.41: Παράδειγμα προσδιορισμού θέσης του εσωτερικού του πολυγώνου σε σχέση με μία κανονική κορυφή

Στο σημείο αυτό η επεξήγηση του αλγορίθμου έχει ολοκληρωθεί. Στη συνέχεια θα αποδειχθεί ότι ο αλγόριθμος υποδιαιρεί σωστά το αρχικό πολύγωνο σε y-μονότονα υποπολύγωνα.

Απόδειξη: Ο αλγόριθμος εισάγει διαγώνιους για κάθε κορυφή διαχωρισμού ή συγχώνευσης. Επομένως, τα υποπολύγωνα που θα δημιουργηθούν δεν θα περιέχουν τέτοιες κορυφές, οπότε και θα είναι y-μονότονα όπως έχει αποδειχθεί στην ενότητα 5.7. Μένει να αποδειχθεί ότι οι διαγώνιες που δημιουργούνται δεν έχουν σημεία τομής μεταξύ τους, ή με τις ακμές του αρχικού πολυγώνου. Για το σκοπό αυτό θα αποδειχθεί ότι κατά την εισαγωγή μίας διαγωνίου, η διαγώνιος αυτή δεν έχει σημεία τομής με κάποια ακμή του πολυγώνου αλλά ούτε με κάποια από τις διαγώνιους που έχουν ήδη προστεθεί. Η απόδειξη θα γίνει για την προσθήκη διαγωνίου για μία κορυφή διαχωρισμού, δηλαδή για τη συνάρτηση ΔιαχείρισηΚορυφήςΔιαχωρισμού. Για προσθήκες μίας διαγωνίου από τις υπόλοιπες συναρτήσεις η απόδειξη είναι παρόμοια.

Έστω $v_i v_m$ η διαγώνιος που προστίθεται από τη συνάρτηση διαχείρισης κορυφής διαχωρισμού όταν η γραμμή σάρωσης βρίσκεται στην κορυφή διαχωρισμού v_i, όπως φαίνεται στο σχήμα 5.42.



Σχήμα 5.42: Απόδειξη αλγορίθμου υποδιαίρεσης πολυγώνου

Έστω e_j η ακμή ακριβώς αριστερά από την v_i και e_k η ακμή ακριβώς δεξιά από την v_i. Τότε όταν η γραμμή σάρωσης φτάνει στην κορυφή v_i θα ισχύει ότι $helper(e_j) = \mathrm{v}_m$. Έστω Q η τετράπλευρη επιφάνεια που ορίζεται από τις

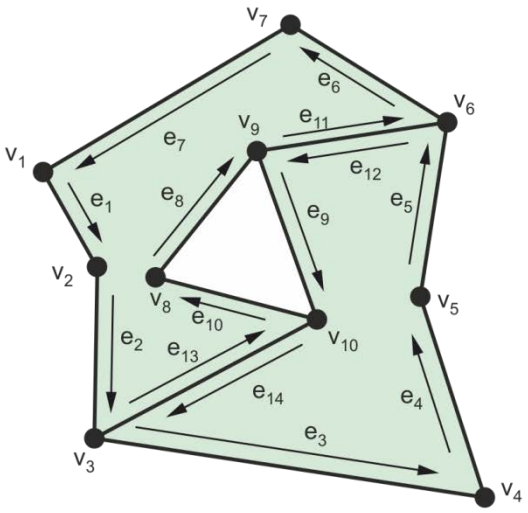
ακμές e_j, e_k και δύο οριζόντιες γραμμές που διέρχονται από τις κορυφές v_i και v_m. Στο εσωτερικό της επιφάνειας Q δεν υπάρχουν κορυφές, γιατί διαφορετικά η πρόταση $helper(e_j) = \mathrm{v}_m$ δεν θα ήταν αληθής. Έστω τώρα ότι υπάρχει μία ακμή του πολυγώνου που διασταυρώνεται με την διαγώνιο που προστέθηκε. Εφόσον η ακμή αυτή δεν μπορεί να έχει κάποιο άκρο της μέσα στην επιφάνεια Q και επειδή οι ακμές του πολυγώνου δεν έχουν σημεία τομής μεταξύ τους, τότε η ακμή θα πρέπει να έχει κάποιο σημείο τομής με την οριζόντια γραμμή που ενώνει την v_m με την e_j, ή με την οριζόντια γραμμή που ενώνει την v_i με την e_j. Πιθανές τέτοιες ακμές είναι οι e_x και e_y. Όμως οι ακμές αυτές είναι αδύνατον να υπάρχουν καθώς για τις κορυφές v_i και v_m η ακμή e_j βρίσκεται ακριβώς αριστερά τους. Επομένως, καμία ακμή του πολυγώνου δεν μπορεί να διασταυρώνεται με την προστιθέμενη διαγώνιο.

Έστω μία διαγώνιος που προστέθηκε πριν από τη διαγώνιο $v_i v_m$. Εφόσον δεν υπάρχουν κορυφές μέσα στην επιφάνεια Q και κάθε διαγώνιος που προστέθηκε πριν την $v_i v_m$ πρέπει να έχει και τα δύο άκρα της υψηλότερα από την κορυφή v_i, τότε καμία διαγώνιος δεν μπορεί να διασταυρώνεται με την διαγώνιο $v_i v_m$.

Στο σημείο αυτό έχει αποδειχθεί ότι ο αλγόριθμος δουλεύει σωστά. Μία τελευταία λειτουργία μετά την εκτέλεση του αλγορίθμου είναι η πρόσβαση σε κάθε y-μονότονο υποπολύγωνο. Αυτό γίνεται με τη χρήση της μεταβλητής **hasBeenTraversed** που περιέχει κάθε ημι-ακμή. Για να διασχιστεί ένα τυχαίο y-μονότονο πολύγωνο που δημιουργήθηκε, ακολουθούνται οι δείκτες Next() μίας τυχαίας ημι-ακμής στη διπλά συνδεμένη λίστα ακμών. Όταν κάποιος δείκτης Next() δείχνει στην αρχική τυχαία ημι-ακμή, τότε το υποπολύγωνο έχει διασχιστεί επιτυχώς.

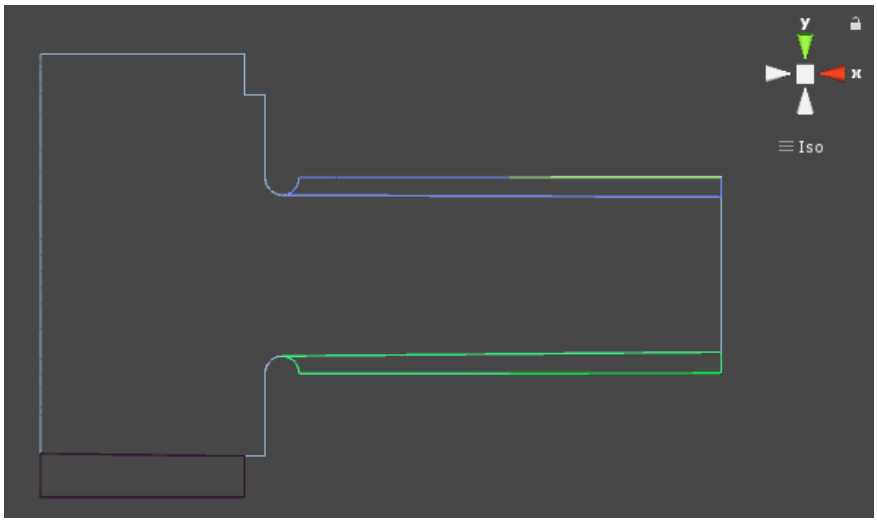
Κατά τη διάρκεια προσπέλασης των ημι-ακμών του υποπολυγώνου, οι μεταβλητές hasBeenTraversed ορίζονται ως true. Επομένως ως αρχή για ένα επόμενο τυχαίο υποπολύγωνο ακολουθείται μία τυχαία ημι-ακμή που δεν εχει διασχιστεί, δηλαδή η μεταβλητή hasBeenTraversed για την ημι-ακμή έχει τιμή false.

Σημειώνεται ότι ο αλγόριθμος δουλεύει σωστά και για πολύγωνα που περιέχουν οπές. Συγκεκριμένα οι οπές δίνονται με δεξιόστροφη φορά των κορυφών τους. Επομένως το εσωτερικό του πολυγώνου θεωρείται ότι βρίσκεται εξωτερικά του πολυγώνου που αναπαριστά μία οπή. Επιπλέον η δεξιόστροφη φορά εξυπηρετεί τη λειτουργία εύρεσης της θέσης του εσωτερικού του πολυγώνου σε σχέση με μία κανονική κορυφή. Οι υπόλοιπες λειτουργίες του αλγορίθμου δουλεύουν με τον ίδιο τρόπο και δεν χρειάζεται να γίνουν διαφοροποιήσεις. Στο σχήμα 5.43 παρουσιάζεται μία εφαρμογή του αλγορίθμου σε ένα πολύγωνο με μία οπή.



Σχήμα 5.43: Παράδειγμα εφαρμογής του αλγορίθμου σε ένα πολύγωνο με μία οπή

Στο σχήμα 5.44 παρουσιάζεται ένα αποτέλεσμα εφαρμογής του αλγορίθμου στο περιβάλλον της Unity. Με διαφορετικό χρώμα σημειώνονται τα y-μονότονα υποπολύγωνα που δημιουργήθηκαν.



Σχήμα 5.44: Αποτέλεσμα εφαρμογής του αλγορίθμου στο περιβάλλον της Unity

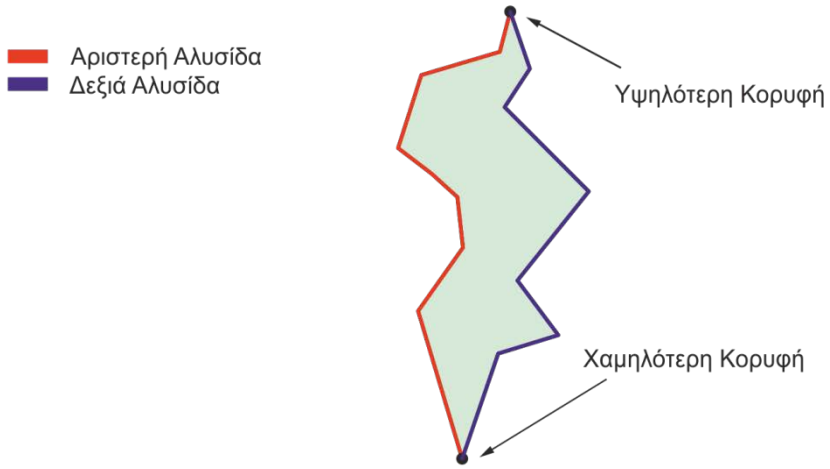
5.8 Ο Αλγόριθμος Τριγωνοποίησης ενός y-Μονότονου Πολυγώνου

Ο αλγόριθμος της προηγούμενης ενότητας έχει υποδιαιρέσει επιτυχώς τα πολύγωνα μαζί με τις οπές που δημιουργούνται επάνω στο επίπεδο τομής, σε y-μονότονα υποπολύγωνα. Το τελευταίο βήμα για την ολοκλήρωση της διαδικασίας τομής είναι η τριγωνοποίηση των υποπολυγώνων αυτών. Στην ενότητα αυτή θα περιγραφεί ένας άπληστος αλγόριθμος τριγωνοποίησης ενός y-μονότονου πολυγώνου.

Η βασική ιδέα του αλγορίθμου όπως και στον αλγόριθμο της ενότητας 5.7, είναι η σάρωση του y-μονότονου πολυγώνου με μία φανταστική οριζόντια γραμμή, η οποία σταματά σε κάθε κορυφή του. Αυτό που επιδιώκεται είναι η υποδιαίρεση του πολυγώνου σε τρίγωνα, με την προσθήκη διαγωνίων μεταξύ των κορυφών. Ο αλγόριθμος διατηρεί μία στοίβα, στην οποία τοποθετούνται κορυφές από τις οποίες έχει περάσει η οριζόντια γραμμή, αλλά μπορεί να χρειαστεί να προστεθούν σε αυτές κάποιες διαγώνιοι στην πορεία του αλγορίθ-

μου. Όταν η οριζόντια γραμμή σταματήσει σε μία κορυφή, ο αλγόριθμος προσθέτει όσο το δυνατόν περισσότερες διαγώνιους στις κορυφές που υπάρχουν στη στοίβα.

Για τη συνέχεια της επεξήγησης του αλγορίθμου, πρέπει αρχικά να οριστούν οι έννοιες της δεξιάς και αριστερής αλυσίδας ενός πολυγώνου. Με τον όρο αλυσίδα ορίζεται το δεξιό ή αριστερό τμήμα του περιγράμματος του πολυγώνου, του οποίου τα άκρα είναι η υψηλότερη και χαμηλότερη κορυφή του πολυγώνου. Στο σχήμα 5.45 παρουσιάζεται ένα παράδειγμα ενός τυχαίου y-μονότονου πολυγώνου, στο οποίο σημειώνονται οι δύο αλυσίδες του.



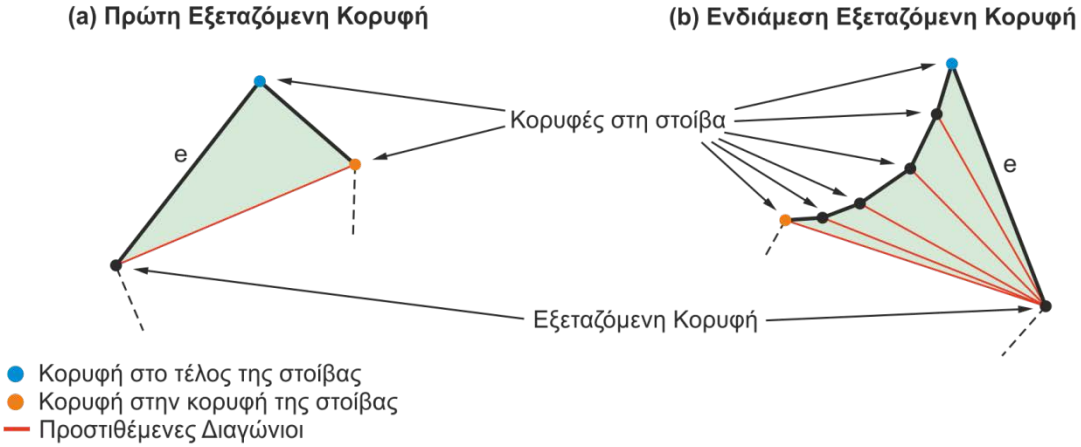
Σχήμα 5.45: Δεξιά και αριστερή αλυσίδα ενός πολυγώνου

Ο αλγόριθμος προσθέτει διαγωνίους προς τις κορυφές της στοίβας, αναλόγως σε ποια αλυσίδα βρίσκεται η κορυφή που εξετάζει. Στο σχήμα 5.46 παρουσιάζεται συνοπτικά η λειτουργία του αλγορίθμου.

Σχήμα 5.46: Διάγραμμα αλγορίθμου τριγωνοποίησης ενός y-μονότονου πολυγώνου

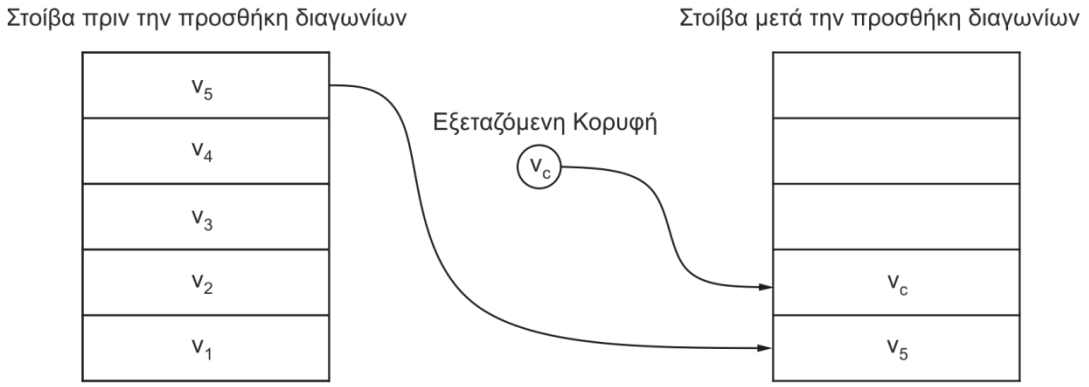
Ο αλγόριθμος προσθέτει αρχικά τις δύο υψηλότερες κορυφές στη στοίβα. Έπειτα τις εξετάζει με φθίνουσα σειρά ως προς την y-συντεταγμένη τους. Υπάρχουν δύο περιπτώσεις στις οποίες μπορεί να ανήκει κάθε εξεταζόμενη κορυφή.

Στην περίπτωση που η εξεταζόμενη κορυφή βρίσκεται σε αντίθετη αλυσίδα από την αλυσίδα της κορυφής που βρίσκεται στην κορυφή της στοίβας, τότε θα πρέπει να αποτελεί το άκρο τέλους μίας ακμής e που βρίσκεται στην αντίθετη αλυσίδα από αυτήν που ανήκουν οι κορυφές της στοίβας. Στο σχήμα 5.47 παρουσιάζονται δύο παραδείγματα αυτής της περίπτωσης. Ένα για την πρώτη εξεταζόμενη κορυφή του αλγορίθμου, καθώς και ένα παράδειγμα για μία τυχαία εξεταζόμενη κορυφή κατά τη διάρκεια εκτέλεσης του αλγορίθμου.



Σχήμα 5.47: Παραδείγματα πρώτης περίπτωσης εξεταζόμενης κορυφής

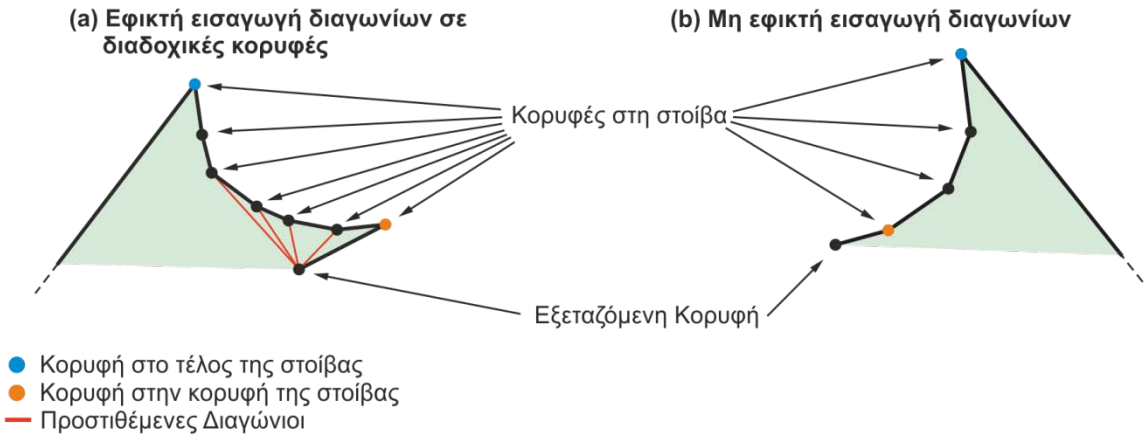
Σε τέτοιες περιπτώσεις γίνεται προσθήκη διαγωνίων από την εξεταζόμενη κορυφή, προς όλες τις κορυφές της στοίβας εκτός από αυτήν που βρίσκεται στο τέλος της. Η κορυφή που βρίσκεται στο τέλος της στοίβας αποτελεί το άνω άκρο της ακμής e, οπότε είναι ήδη συνδεμένη με την εξεταζόμενη κορυφή. Στη συνέχεια γίνεται εξαγωγή όλων των κορυφών από τη στοίβα. Η πρώτη διαγώνιος που προστίθεται, ενώνει την εξεταζόμενη κορυφή με την κορυφή που βρισκόταν στην κορυφή της στοίβας. Το μέρος του πολυγώνου που βρίσκεται πάνω από την διαγώνιο αυτή, έχει πλέον τριγωνοποιηθεί. Οι κορυφές της διαγωνίου αυτής, αποτελούν ταυτόχρονα μέρος του πολυγώνου που δεν έχει ακόμα τριγωνοποιηθεί και βρίσκεται κάτω από την διαγώνιο. Για τον λόγο αυτό γίνεται προσθήκη και των δύο κορυφών στη στοίβα. Συνεπώς η κορυφή που αρχικά βρισκόταν στην κορυφή της στοίβας, θα βρίσκεται στο τέλος της στοίβας μετά την προσθήκη διαγωνίων. Η διαμόρφωση της στοίβας πριν και μετά την προσθήκη διαγωνίων παρουσιάζεται στο σχήμα 5.48.



Σχήμα 5.48: Διαμόρφωση στοίβας πριν και μετά την προσθήκη διαγωνίων για την πρώτη περίπτωση

Στη δεύτερη περίπτωση η εξεταζόμενη κορυφή και η κορυφή στην κορυφή της στοίβας βρίσκονται στην ίδια αλυσίδα. Τότε η εισαγωγή διαγωνίων από την εξεταζόμενη κορυφή

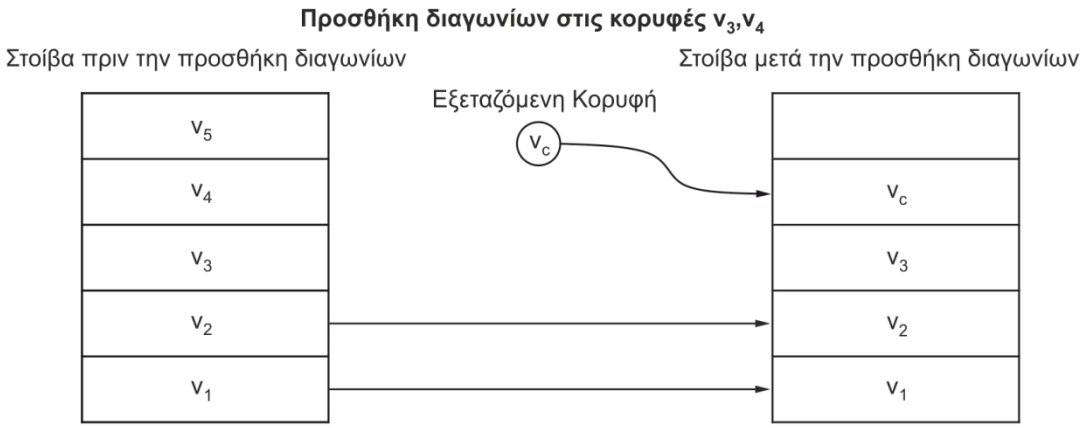
προς κάθε κορυφή της στοίβας μπορεί να μην είναι εφικτή. Οι κορυφές της στοίβας στις οποίες είναι εφικτή η προσθήκη μίας διαγωνίου θα είναι διαδοχικές, όπως φαίνεται στο σχήμα 5.49 (a).



Σχήμα 5.49: Παραδείγματα δεύτερης περίπτωσης εξεταζόμενης κορυφής

Επιπλέον η κορυφή που βρίσκεται στην κορυφή της στοίβας ενώνεται ήδη με την εξεταζόμενη κορυφή, επομένως σίγουρα δεν είναι δυνατή η προσθήκη διαγωνίου προς αυτήν. Επομένως αρχικά εξάγεται η κορυφή που βρίσκεται στην κορυφή της στοίβας και στη συνέχεια ερευνάται η δυνατότητα προσθήκης διαγωνίων προς τις υπόλοιπες κορυφές. Η προσθήκη διαγωνίων σταματά μόλις βρεθεί μία κορυφή στην οποία δεν είναι δυνατή η προσθήκη μίας διαγωνίου επειδή η διαγώνιος θα έχει κάποιο σημείο τομής με κάποια ακμή του πολυγώνου ή δεν θα βρίσκεται στο εσωτερικό του. Για όσες κορυφές έχει προστεθεί μία διαγώνιος εξάγονται από τη στοίβα. Εξαιρείται η κορυφή της τελευταίας διαγωνίου που προστέθηκε, η οποία πρέπει να προστεθεί ξανά στη στοίβα καθώς αποτελεί τμήμα του μη τριγωνοποιημένου πολυγώνου όπως φαίνεται στο σχήμα 5.49 (a).

Σημειώνεται ότι μπορεί να είναι αδύνατη η προσθήκη οποιασδήποτε διαγωνίου, όπως στο σχήμα 5.49 (b). Στην περίπτωση αυτή η πρώτη κορυφή που εξάγεται εισάγεται και πάλι στη στοίβα. Ανεξάρτητα εάν θα προστεθεί κάποια διαγώνιος, η εξεταζόμενη κορυφή προστίθεται πάντα στη στοίβα αφού αποτελεί τμήμα του μη τριγωνοποιημένου πολυγώνου. Οπότε στο τέλος εξέτασης της, θα βρίσκεται πάντα στη κορυφή της στοίβας. Στο σχήμα 5.50 παρουσιάζεται η διαμόρφωση της στοίβας σε περίπτωση που είναι εφικτή η προσθήκη διαγωνίων. Διαφορετικά, η στοίβα μετά την εξέταση της κορυφής θα παραμείνει ίδια με την εξεταζόμενη κορυφή επιπλέον στην κορυφή της.



Σχήμα 5.50: Διαμόρφωση στοίβας πριν και μετά την προσθήκη διαγωνίων για τη δεύτερη περίπτωση

5.8.1 Υλοποίηση στη Unity

Η υλοποίηση του παραπάνω αλγορίθμου γίνεται στην συνάρτηση **PartitionIntoMonotonePieces** η οποία βρίσκεται στην κλάση **Triangulator**. Το πολύγωνο προς τριγωνοποίηση δίνεται ως μία διπλά συνδεμένη λίστα ακμών, η οποία κατασκευάζεται διασχίζοντας τις ημι-ακμές του πολυγώνου αυτού. Η διάσχιση ενός τυχαίου πολυγώνου περιγράφεται στην ενότητα 5.7.1. Η διπλά συνδεμένη λίστα ακμών περιέχει μόνο τις ημι-ακμές που βρίσκονται στο εσωτερικό του πολυγώνου, καθώς μόνο αυτές είναι απαραίτητες για την τριγωνοποίηση. Σημειώνεται, ότι οι ημι-ακμές του πολυγώνου έχουν αριστερόστροφη κατεύθυνση. Ο ψευδοκώδικας του αλγορίθμου παρουσιάζεται στο σχήμα 5.51.

ΤριγωνοποίησηΜονότονουΠολυγώνου (P)

1. Φθίνουσα ταξινόμιση των κορυφών του πολυγώνου ως προς την y-συντεταγμένη.
   Για δύο κορυφές με ίδια y-συντεταγμένη η κορυφή που βρίσκεται πιο αριστερά έχει
   μεγαλύτερη προτεραιότητα. Έστω $v_1,v_2,\ldots v_n$ η ακολουθία των ταξινομημένων κορυφών
2. Αρχικοποίηση στοίβας και εισαγωγή των κορυφών v_1,v_2 σε αυτή
3. **Για** i **από** 3 **μέχρι** n-1
4. **Αν** η κορυφή v_i και η κορυφή στην κορυφή της στοίβας βρίσκονται σε διαφορετικές αλυσίδες
5. **Τότε** Εξαγωγή όλων των κορυφών από τη στοίβα
6. Εισαγωγή διαγωνίου από την v_i προς κάθε κορυφή που βρίσκοταν στη στοίβα εκτός από την τελευταία
7. Εισαγωγή κορυφών v_{i-1} και v_i στη στοίβα
8. **Αλλιώς** (Οι κορυφές βρίσκονται στην ίδια αλυσίδα)
9. **Τότε** Εξαγωγή μίας κορυφής από της στοίβα
10. Επιπλέον εξαγωγή κάθε κορυφής όπου η διαγώνιος από την v_i προς αυτήν βρίσκεται στο εσωτερικό του πολυγώνου. Εισαγωγή αυτής της διαγωνίου
11. Εισαγωγή στη στοίβα της τελευταίας κορυφής που έγινε εξαγωγή
12. Εισαγωγή κορυφής v_i στη στοίβα
13. Εισαγωγή διαγωνίων από την κορυφή v_n προς όλες τις κορυφές τις στοίβας εκτός την πρώτη και την τελευταία

Σχήμα 5.51: Ψευδοκώδικας αλγορίθμου τριγωνοποίησης

Στη συνέχεια θα εξηγηθούν μερικές βασικές λειτουργίες που αναφέρονται στον ψευδοκώδικα. Αρχικά, η λίστα κορυφών ταξινομείται και στη συνέχεια οι δύο πρώτες κορυφές τοποθετούνται στη στοίβα. Καθοριστικής σημασίας είναι ο προσδιορισμός της αλυσίδας του πολυγώνου στην οποία ανήκει κάθε κορυφή. Σε κάθε κορυφή διατηρείται μία επιπλέον μεταβλητή που προσδιορίζει την αλυσίδα στην οποία ανήκει. Για την εύρεση της αλυσίδας κάθε κορυφής ακολουθείται η αριστερή αλυσίδα του πολυγώνου, ξεκινώντας από την υ-

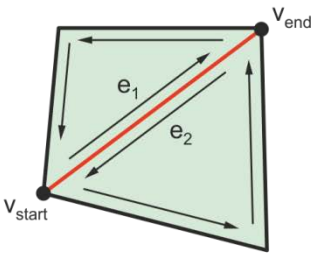
ψηλότερη κορυφή και ακολουθώντας τους δείκτες Next κάθε ημι-ακμής, μέχρις ότου να βρεθεί η χαμηλότερη κορυφή του πολυγώνου. Υπενθυμίζεται ότι οι ημι-ακμές στο εσωτερικό του πολυγώνου έχουν αριστερόστροφη φορά, οπότε με αρχή την υψηλότερη κορυφή οι δείκτες Next των ημι-ακμών παραπέμπουν στην αριστερή αλυσίδα. Η διαδικασία προσδιορισμού της αλυσίδας κάθε κορυφής παρουσιάζεται στον κώδικα του πίνακα 5.15. Κατά τη διάρκεια αυτής της διαδικασίας γίνεται ταυτόχρονα έλεγχος αν το πολύγωνο είναι y-μονότονο, ελέγχοντας τις y συντεταγμένες δύο ακμών της αλυσίδας. Ο ίδιος έλεγχος γίνεται στη συνέχεια και για τη δεξιά αλυσίδα.

Προσδιορισμός αλυσίδας που ανήκει κάθε κορυφή

```
do {
    if (currentVertex.GetPosition().y < currentVertex.GetIncidentEdge ().GetNext ().
        GetOrigin ().GetPosition().y){

        Debug.LogError("Input Polygon is not Monotone!");
        return;
    }
    currentVertex = currentVertex.GetIncidentEdge ().GetNext ().GetOrigin ();
    currentVertex.SetIsOnRightChain (false);

} while(currentVertex.GetPosition () != bottomVertexPosition);
```

Πίνακας 5.15: Κώδικας προσδιορισμού αλυσίδας για κάθε κορυφή

Για την προσθήκη διαγωνίων όπως και στον αλγόριθμο της ενότητας 5.7, είναι απαραίτητη η σωστή αλλαγή δεικτών των ημι-ακμών. Η αλλαγή δεικτών στον παρόντα αλγόριθμο ακολουθεί παρόμοια λογική όπως αυτή του σχήματος 5.39 στην ενότητα 5.7. Συγκεκριμένα, έστω μία νέα διαγώνιος που δημιουργείται και ενώνει τις κορυφές $\mathbf{v_{start}}$ και $\mathbf{v_{end}}$ και $\mathbf{e_1}$ **και** $\mathbf{e_2}$ είναι οι αντίστοιχες ημι-ακμές της. Η ημι-ακμή $\mathbf{e_1}$ έχει ως άκρο αρχής την κορυφή $\mathbf{v_{start}}$. Τότε οι αλλαγές στους δείκτες των ημι-ακμών παρουσιάζονται στο σχήμα 5.52.



Προσθήκη νέας διαγωνίου
Αλλαγές δεικτών με τη σειρά που αυτές εφαρμόζονται

e_1.Origin = v_{start}
e_1.Twin = e_2
e_1.Prev = v_{start}.IncidentEdge.Prev
e_2.Origin = v_{end}
e_2.Twin = e_1
e_2.Next = v_{start}.IncidentEdge
e_1.Next = v_{end}.IncidentEdge
e_2.Prev = v_{end}.IncidentEdge.Prev
v_{end}.IncidentEdge.Prev.Next = e_2
v_{end}.IncidentEdge.Prev = e_1
v_{start}.IncidentEdge.Prev.Next = e_1
v_{start}.IncidentEdge.Prev = e_2

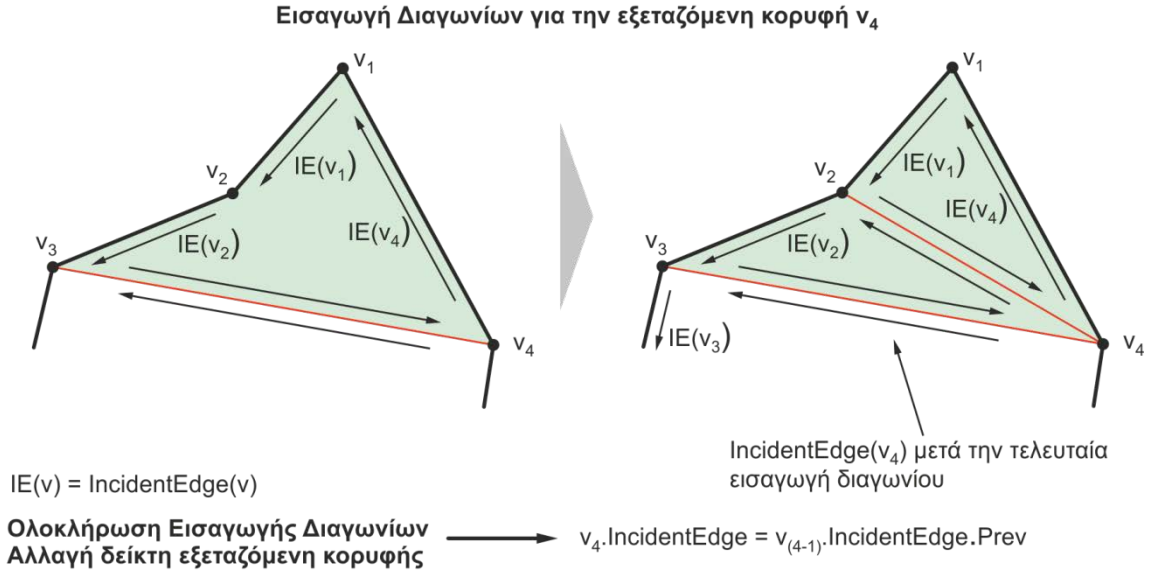
Σχήμα 5.52: Αλλαγή δεικτών ημι-ακμών κατά την προσθήκη μίας διαγωνίου

Αυτό που διαφοροποιείται στις δύο περιπτώσεις εξεταζόμενης κορυφής, είναι η κατάλληλη ανάθεση των δεικτών IncidentEdge για όποιες κορυφές αυτό είναι απαραίτητο, μετά την ολοκλήρωση προσθήκης των διαγωνίων. Οι κορυφές αυτές αφορούν τα δύο άκρα κάθε διαγωνίου που προστίθεται. Ο συγκεκριμένος δείκτης παίζει καθοριστικό ρόλο στην εισα-

γωγή διαγωνίων, καθώς χρησιμοποιείται αρκετά όπως φαίνεται στο σχήμα 5.52. Επομένως, είναι απαραίτητο να δείχνει στη σωστή ημι-ακμή για την ορθή εισαγωγή διαγωνίων κατά την εξέταση των υπόλοιπων κορυφών.

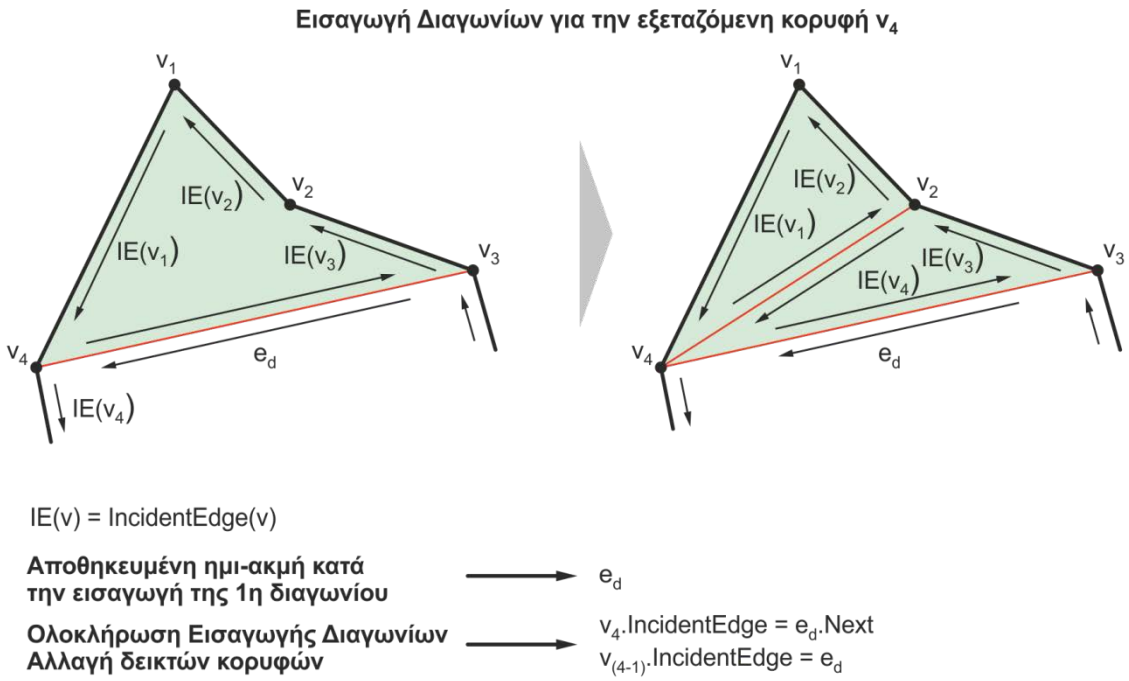
Για την περίπτωση που η εξεταζόμενη κορυφή ανήκει σε διαφορετική αλυσίδα από την κορυφή στην κορυφή της στοίβας, διακρίνονται δύο περιπτώσεις. Είτε να βρίσκεται στην αριστερή είτε στη δεξιά αλυσίδα. Υπενθυμίζεται ότι οι διαγώνιοι εισάγονται ανάποδα. Δηλαδή, πρώτα εισάγεται η διαγώνιος για την κορυφή που βρίσκεται στην κορυφή της στοίβας και στη συνέχεια για τις υπόλοιπες.

Στο σχήμα 5.53 παρουσιάζεται η σειρά εισαγωγής διαγωνίων όταν η εξεταζόμενη κορυφή βρίσκεται στη δεξιά αλυσίδα. Οι δείκτες IncidentEdge δείχνουν στις σωστές ακμές οπότε οι αλλαγές δεικτών κατά την προσθήκη διαγωνίων θα γίνουν σωστά.



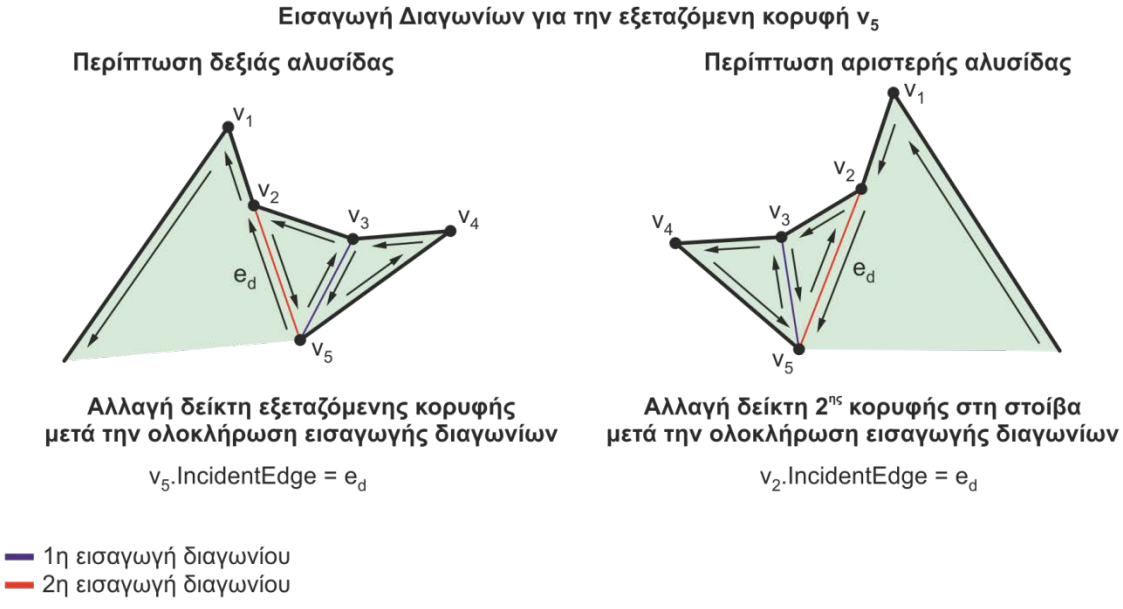
Σχήμα 5.53: Εισαγωγή διαγωνίων και αλλαγές δεικτών για κορυφή στη δεξιά αλυσίδα

Όταν η εξεταζόμενη κορυφή βρίσκεται στην αριστερή αλυσίδα και οι κορυφές της στοίβας στη δεξιά, ο δείκτης IncidentEdge της εξεταζόμενης κορυφής πρέπει να αλλάζει σε κάθε εισαγωγή διαγωνίου και συγκεκριμένα να δείχνει στην πρώτη ημι-ακμή της διαγωνίου που εισάγεται. Μόλις ολοκληρωθούν όλες οι εισαγωγές, ο δείκτης θα πρέπει και πάλι να αλλάξει στην ημι-ακμή που έδειχνε πριν την εισαγωγή διαγωνίων για την ορθή συνέχεια του αλγορίθμου. Επιπλέον πρέπει να γίνει αλλαγή στο δείκτη της κορυφής που βρίσκεται αμέσως υψηλότερα από την εξεταζόμενη κορυφή. Για το λόγο αυτό αποθηκεύεται εξαρχής η δεύτερη ημι-ακμή της πρώτης διαγωνίου που προστίθεται. Η διαδικασία παρουσιάζεται στο σχήμα 5.54. Σε κάθε βήμα παρουσιάζονται οι δείκτες των κορυφών κατά την προσθήκη της εκάστοτε διαγωνίου στο συγκεκριμένο βήμα.



Σχήμα 5.54: Εισαγωγή διαγωνίων και αλλαγές δεικτών για κορυφή στην αριστερή αλυσίδα

Για την περίπτωση που η εξεταζόμενη κορυφή ανήκει στην ίδια αλυσίδα από την κορυφή στην κορυφή της στοίβας, διακρίνονται επίσης δύο περιπτώσεις. Είτε και οι δύο κορυφές βρίσκονται στη δεξιά αλυσίδα είτε στην αριστερή. Σε κάθε περίπτωση η εισαγωγή διαγωνίων δεν παρουσιάζει δυσκολίες και οι αλλαγές δεικτών γίνονται όπως ακριβώς αναφέρονται στο σχήμα 5.52. Μετά την ολοκλήρωση εισαγωγής όλων των διαγωνίων θα πρέπει να γίνει αλλαγή του δείκτη IncidentEdge της εξεταζόμενης κορυφής ή της επόμενης κορυφής στη στοίβα, όπως παρουσιάζεται στο σχήμα 5.55.



Σχήμα 5.55: Εισαγωγή διαγωνίων και αλλαγές δεικτών στις περιπτώσεις ίδιας αλυσίδας

Μία τελευταία σημαντική λειτουργία κατά τη διάρκεια του αλγορίθμου είναι ο έλεγχος εάν μία διαγώνιος βρίσκεται στο εσωτερικό του πολυγώνου και επομένως μπορεί να προστε-

θεί. Για τον έλεγχο αυτό χρησιμοποιείται και πάλι η σχετική θέση μίας κορυφής με μία ευθεία. Συγκεκριμένα εξετάζεται η θέση της τελευταίας κορυφής που αφαιρέθηκε από τη στοίβα, σε σχέση με την ευθεία που διέρχεται από τα δύο άκρα της διαγωνίου εισαγωγής. Η κορυφή αυτή πρέπει να βρίσκεται πάντα δεξιά η αριστερά από την ευθεία αυτή, αναλόγως την αλυσίδα που προστίθεται η διαγώνιος. Έστω v_{last} η τελευταία κορυφή που αφαιρέθηκε από τη στοίβα. Τότε ισχύει:

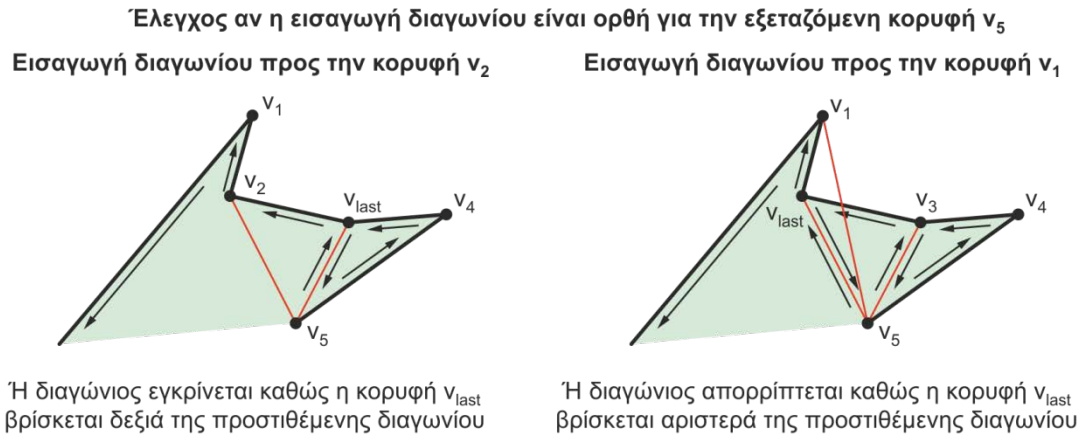
Αριστερή αλυσίδα

- v_{last} στα δεξιά της διαγωνίου εισαγωγής : Η διαγώνιος απορρίπτεται
- v_{last} στα αριστερά της διαγωνίου εισαγωγής : Η διαγώνιος εγκρίνεται

Δεξιά αλυσίδα

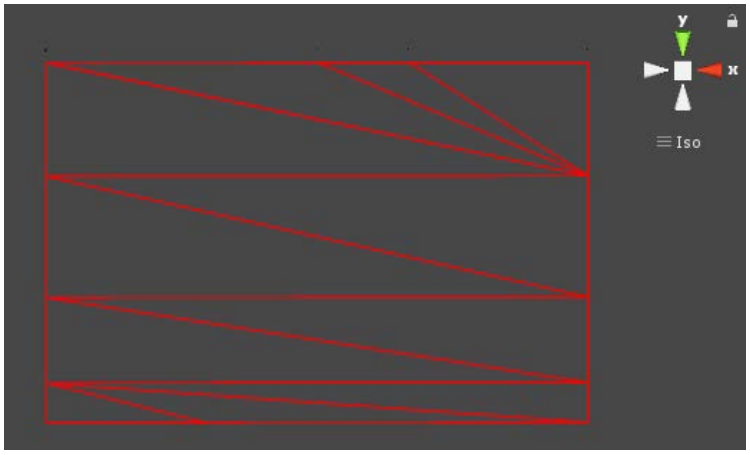
- v_{last} στα δεξιά της διαγωνίου εισαγωγής : Η διαγώνιος εγκρίνεται
- v_{last} στα αριστερά της διαγωνίου εισαγωγής : Η διαγώνιος απορρίπτεται

Στο σχήμα 5.56 παρουσιάζεται ένα παράδειγμα όπου η διαγώνιος εισαγωγής εγκρίνεται ή απορρίπτεται.



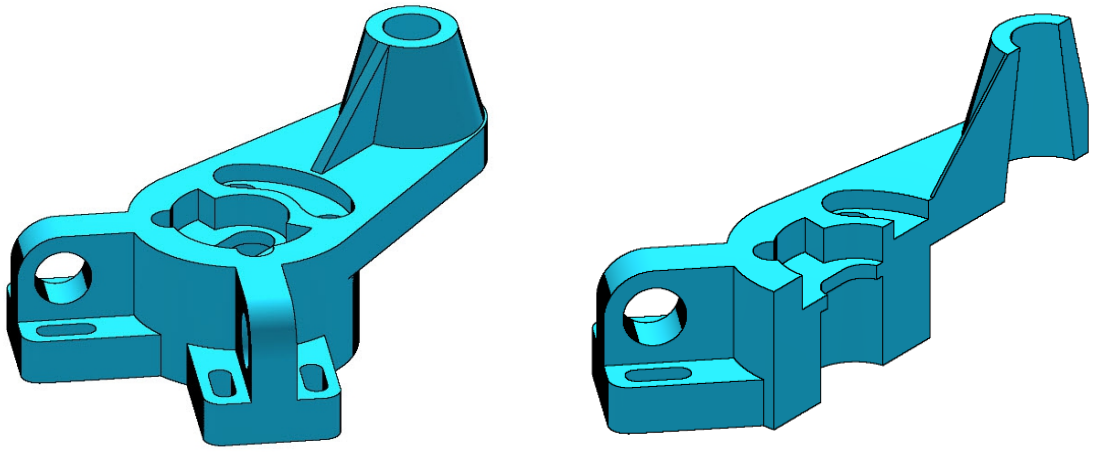
Σχήμα 5.56: Παράδειγμα έγκρισης ή απόρριψης προστιθέμενης διαγωνίου στην δεξιά αλυσίδα

Στο σημείο αυτό η επεξήγηση του αλγορίθμου έχει ολοκληρωθεί. Στο σχήμα 5.57 παρουσιάζεται ένα αποτέλεσμα του αλγορίθμου μέσα στο περιβάλλον της Unity.



Σχήμα 5.57: Αποτέλεσμα εφαρμογής αλγορίθμου στο περιβάλλον της Unity

Η διαδικασία τομής ολοκληρώνεται εφόσον οι πίνακες του δικτύου γεωμετρίας του μοντέλου ανανεωθούν. Τα νέα τρίγωνα θα είναι αυτά που προκύπτουν από τον αλγόριθμο τριγωνοποίησης και οι νέες κορυφές οι αντίστοιχες κορυφές τους. Σημειώνεται ότι στον νέο πίνακα κορυφών τοποθεντούνται οι θέσεις κορυφών στον τρισδιάστατο χώρο που είχαν αποθηκευτεί στη διπλά συνδεμένη λίστα ακμών και όχι ο μετασχηματισμός τους στο επίπεδο xy. Το τελικό αποτέλεσμα συνεργασίας όλων των παραπάνω αλγορίθμων που αναφέρθηκαν είχε φανεί στο σχήμα 5.3 στην αρχή του παρόντος κεφαλαίου και παρουσιάζεται ένα αντίστοιχο στο σχήμα 5.58.



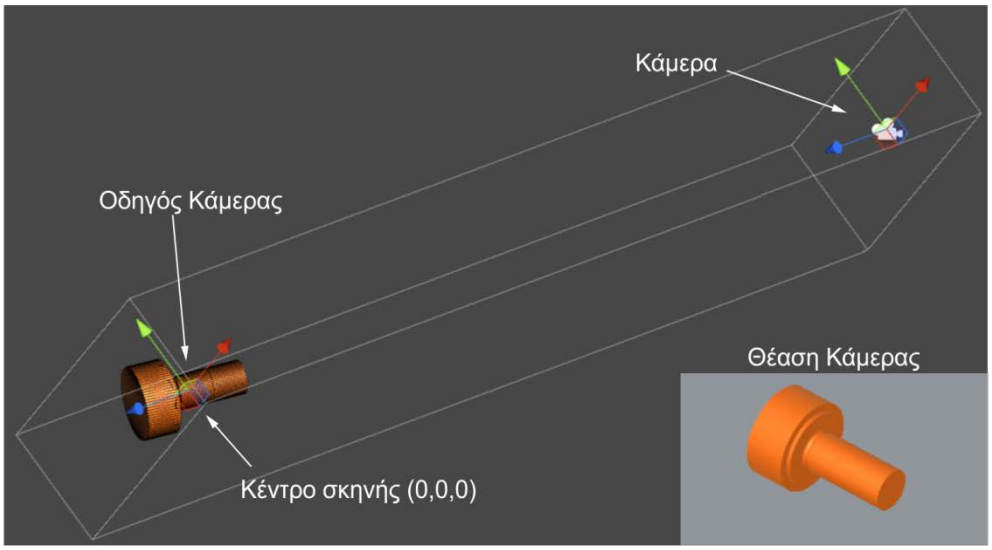
Σχήμα 5.58: Αποτέλεσμα εφαρμογής αλγορίθμου στο περιβάλλον της Unity

6. Ο ΕΛΕΓΧΟΣ ΤΗΣ ΚΑΜΕΡΑΣ

Η διαχείριση του τρισδιάστατου μοντέλου εντός της εφαρμογής γίνεται με τη χρήση αφής. Οι βασικές υποστηριζόμενες λειτουργίες αφορούν την περιστροφή, μεγέθυνση ή μετακίνηση του μοντέλου. Σε όλες αυτές σημαντικό ρόλο κατέχει η κάμερα, της οποίας η κατάλληλη μετακίνηση είναι ουσιαστικά υπεύθυνη για την υλοποίηση τους. Στο κεφάλαιο αυτό θα εξηγηθεί αναλυτικά ο τρόπος μετακίνησης της κάμερας μέσω αφής για την υλοποίηση βασικών λειτουργιών μετακίνησης και θέασης του μοντέλου. Όλες οι λειτουργίες βρίσκονται στο script CameraController το οποίο τοποθετείται στο αντικείμενο της κάμερας στο περιβάλλον της Unity.

6.1 Τοποθέτηση της κάμερας και ο τρόπος προβολής

Για την αποτελεσματική υλοποίηση κάποιων βασικών λειτουργιών, απαραίτητη προϋπόθεση αποτελεί η σωστή τοποθέτηση της κάμερας στη σκηνή. Στην παρούσα εφαρμογή η κάμερα αποτελείται από ένα αντικείμενο-οδηγό, ο οποίος τοποθετείται στο κέντρο του στερεού μοντέλου και από το βασικό αντικείμενο της κάμερας, το οποίο ορίζεται ως παιδί του οδηγού αυτού. Υπενθυμίζεται ότι το στερεό μοντέλο βρίσκεται επίσης στο κέντρο της σκηνής στην προκαθορισμένη θέση του, δηλαδή χωρίς να έχει γίνει κάποια περιστροφή σε αυτό. Στο σχήμα 6.1 παρουσιάζεται η θέση της κάμερας και του οδηγού της στη σκηνή την Unity.

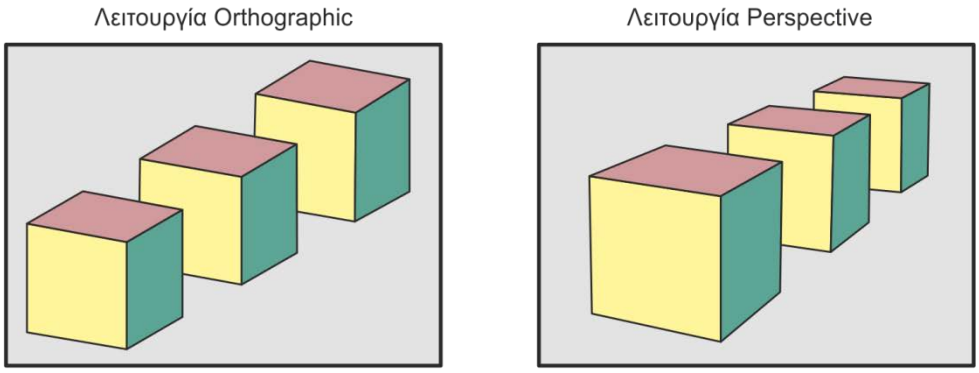


Σχήμα 6.1: Τοποθέτηση της κάμερας στη σκηνή

Η παραπάνω τοποθέτηση διευκολύνει σημαντικά την υλοποίηση κάποιων λειτουργιών όπως θα εξηγηθεί στη συνέχεια.

Ο τρόπος προβολής της κάμερας υποστηρίζει δύο βασικές λειτουργίες. Τη λειτουργία **perspective** και τη λειτουργία **orthographic**. Όταν η κάμερα βρίσκεται σε perspective λειτουργία, τότε σε κάθε όψη της υπάρχει και η αίσθηση του βάθους της εικόνας που προβάλλεται, όπως και στην πραγματικότητα. Η λειτουργία αυτή χρησιμοποιείται συχνά σε εφαρμογές που απαιτείται προσομοίωση της πραγματικότητας (τρισδιάστατα παιχνίδια, προσομοιωτές πτήσης, κ.λπ.). Αντίθετα, η λειτουργία orthographic πρόκειται για την αναπαράσταση τρισδιάστατων αντικειμένων σε δύο μόνο διαστάσεις. Συνεπώς, απουσιάζει η αίσθηση του βάθους και όλα τα αντικείμενα παρουσιάζονται επίπεδα. Η λειτουργία αυτή χρησιμοποιείται συχνά σε δισδιάστατα παιχνίδια ή εφαρμογές μοντελοποίησης, καθώς

προσφέρει ακριβέστερη σχεδίαση. Στο σχήμα 6.2 παρουσιάζονται τα μοντέλα για τρεις κύβους όπως αυτά φαίνονται από μία κάμερα στις δύο αυτές λειτουργίες.



Σχήμα 6.2: Παράδειγμα των λειτουργιών perspective και orthographic

Η αίσθηση του βάθους είναι ορατή στη λειτουργία perspective σε αντίθεση με τη λειτουργία orthographic. Συγκεκριμένα είναι προφανές ότι ο ένας κύβος βρίσκεται πίσω και πιο μακριά από τον προηγούμενο του, όπως και στην πραγματικότητα. Η προκαθορισμένη λειτουργία της παρούσας εφαρμογής είναι η λειτουργία orthographic ενώ ταυτόχρονα προσφέρεται και η λειτουργία perspective εάν αυτή χρειαστεί.

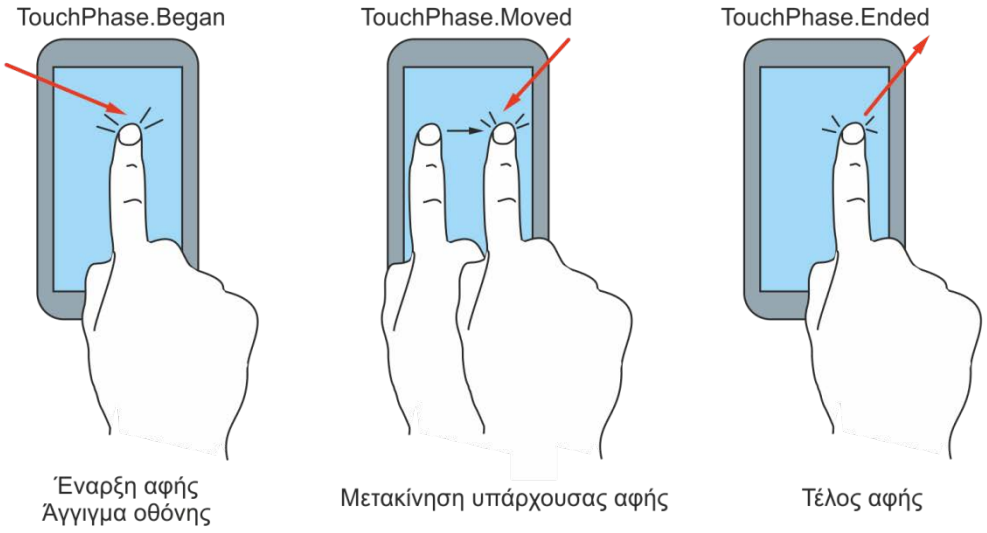
6.2 Διαχείριση αφής στο περιβάλλον της Unity

Όπως ήδη αναφέρθηκε η μετακίνηση της κάμερας γίνεται μέσω αφής. Συνεπώς, είναι απαραίτητη η αναγνώριση αυτής και η κατάλληλη διαχείριση της. Η αναγνώριση και διαχείριση αφής στο περιβάλλον της Unity γίνεται μέσω των κλάσεων **Touch**, **Input** και **TouchPhase.**

Η κλάση Touch αναπαριστά ένα αντικείμενο αφής, δηλαδή την κατάσταση για κάποιο δάκτυλο που αγγίζει την οθόνη. Μέσω αυτής αποκτάται πρόσβαση σε βασικές παραμέτρους, όπως η θέση αφής ή το στάδιο στο οποίο βρίσκεται. Επίσης μπορεί να αναγνωριστεί εάν η αφή αποτελείται από μία ή περισσότερες διαδοχικές επαφές στην οθόνη. Κάθε αντικείμενο αφής έχει ένα μοναδικό αναγνωριστικό **fingerID**, το οποίο εξυπηρετεί στην αναγνώριση του συγκεκριμένου αντικειμένου αφής μεταξύ διαφορετικών frames της εφαρμογής.

Η κλάση Input είναι η βασική κλάση που παρέχει πρόσβαση σε δεδομένα εισόδου από περιφερειακές συσκευές. Μεταξύ αυτών περιέχει μία βασική μεταβλητή, η οποία προσδιορίζει κάθε στιγμή τον αριθμό αγγιγμάτων στην οθόνη της συσκευής. Η μεταβλητή αυτή είναι η **touchCount**. Εάν η μεταβλητή αυτή είναι θετική, τότε κάθε αντικείμενο αφής μπορεί να ανακτηθεί ξεχωριστά μέσω της συνάρτησης **Input.GetTouch(int index)**. Η παράμετρος index προσδιορίσει τον αριθμό του αντικειμένου αφής με τη σειρά που αυτό εμφανίστηκε($1^{ο}$ άγγιγμα, $2^{ο}$ άγγιγμα, κ.λπ).

Τέλος η κλάση TouchPhase προσδιορίζει την κατάσταση που βρίσκεται ένα αντικείμενο αφής. Οι καταστάσεις αυτές δηλώνουν εάν η αφή έχει μόλις ξεκινήσει (**Began**), εάν έχει μετακινηθεί (**Moved**), εάν παραμένει σταθερή (**Stationary**), εάν έχει ακυρωθεί (**Canceled**), ή εάν έχει ολοκληρωθεί (**Ended**). Οι βασικές καταστάσεις ενός αντικειμένου αφής παρουσιάζονται στο σχήμα 6.3.



Σχήμα 6.3: Βασικές καταστάσεις ενός αντικειμένου αφής

6.3 Οι βασικές λειτουργίες της κάμερας

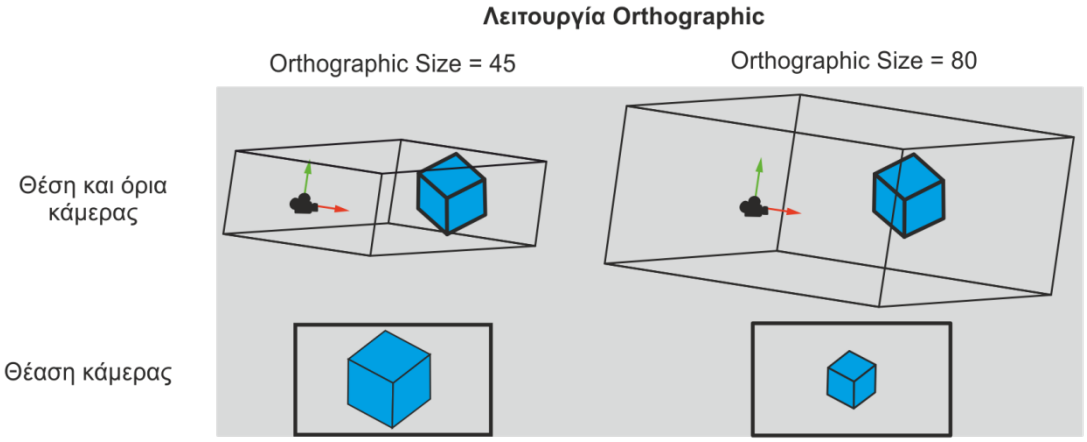
Η πιο σημαντική ίσως λειτουργία κατά τη διάρκεια διαχείρισης ενός μοντέλου, είναι η μεγέθυνση ή σμίκρυνση αυτού (**Zoom in – Zoom out**). Για είσοδο αφής η λειτουργία αυτή εκτελείται συνήθως με τη χρήση δύο δακτύλων, δηλαδή δύο αντικειμένων αφής, και την κατάλληλη τροποποίηση της θέσης της κάμερας. Όταν τα δύο δάκτυλα απομακρύνονται το μοντέλο μεγεθύνεται, διαφορετικά πραγματοποιείται σμίκρυνση σε αυτό. Η υλοποίηση της λειτουργίας αυτής με τη χρήση δύο αντικειμένων αφής, παρουσιάζεται στον κώδικα του πίνακα 6.1.

Μεγέθυνση και σμίκρυνση μοντέλου με τη χρήση κάμερας

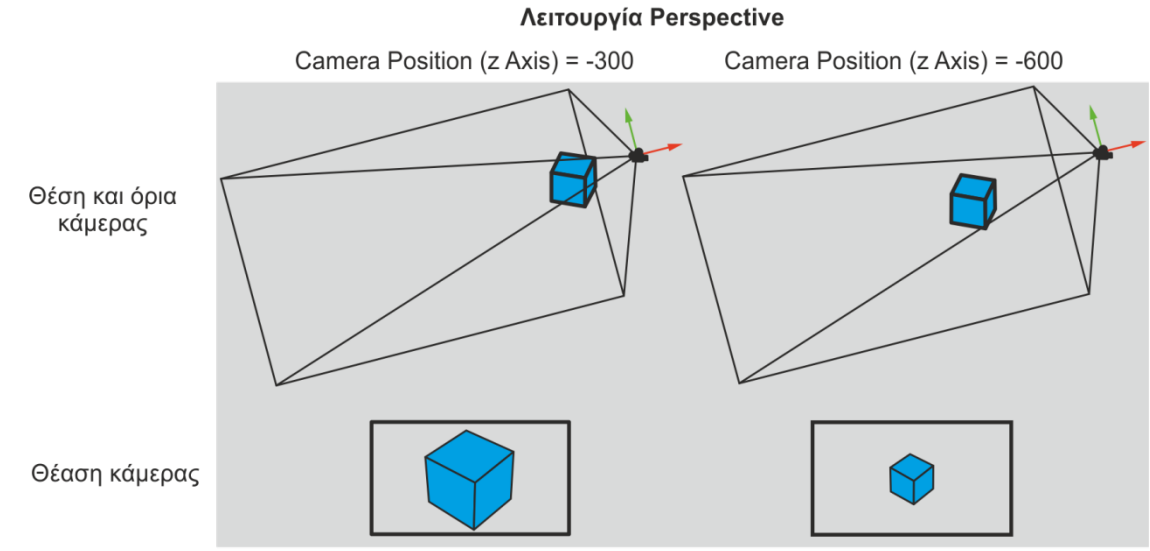
```
if (Input.touchCount == 2) {
   Touch touch0 = Input.GetTouch (0);
   Touch touch1 = Input.GetTouch (1);

   Vector2 touch0PrevPos = touch0.position - touch0.deltaPosition;
   Vector2 touch1PrevPos = touch1.position - touch1.deltaPosition;

   float prevTouchDeltaMag = (touch0PrevPos - touch1PrevPos).magnitude;
   float touchDeltaMag = (touch0.position - touch1.position).magnitude;

   float deltaMagnitudeDiff = prevTouchDeltaMag - touchDeltaMag;

   if (cameraComponent.orthographic) {

      cameraComponent.orthographicSize += deltaMagnitudeDiff * zoomSensitivity;

   } else {
      this.transform.localPosition -= new Vector3 (0, 0, deltaMagnitudeDiff * zoomSensitivity);
   }
}
```

Πίνακας 6.1: Κώδικας μεγέθυνσης ή σμίκρυνσης μοντέλου μέσω αφής

Αρχικά θα πρέπει να βρεθεί η θέση των δακτύλων κατά τη διάρκεια του προηγούμενου frame. Αυτό γίνεται με τη χρήση της μεταβλητής **deltaPosition** που παρέχει τη διαφορά της τωρινής θέσης ενός αντικειμένου αφής και της θέσης του στο προηγούμενο frame. Στη συνέχεια αφαιρούνται οι αποστάσεις των αντικειμένων αφής (αποστάσεις δακτύλων), μεταξύ του τωρινού και του προηγούμενου frame. Η διαφορά αυτή εφόσον είναι θετική, δηλώνει ότι πρέπει να εφαρμοστεί σμίκρυνση, ενώ όταν είναι αρνητική πρέπει να εφαρμοστεί μεγέθυνση. Για τη λειτουργία orthographic εφόσον δεν υπάρχει αίσθηση του βάθους, η μεγέθυνση γίνεται με τον περιορισμό των ορίων της κάμερας (**orthographic size**). Έτσι το ίδιο αντικείμενο φαίνεται μεγαλύτερο σε μία κάμερα με μικρότερα όρια και μικρότερο σε μία κάμερα με μεγαλύτερα όρια. Αντιθέτως για τη λειτουργία perspective αρκεί η κάμερα να πλησιάσει το αντικείμενο για τη λειτουργία μεγέθυνσης και να απομακρυνθεί από αυτό για τη λειτουργία σμίκρυνσης. Το ποσοστό μεγέθυνσης ή σμίκρυνσης ελέγχεται μέσω μίας μεταβλητής (**zoomSensitivity**) που πολλαπλασιάζεται με τη διαφορά αποστάσεων που αναφέρθηκε παραπάνω. Στα σχήματα 6.4 και 6.5 παρουσιάζεται ο τρόπος μεγέθυνσης εάν η κάμερα βρίσκεται σε λειτουργία orthographic ή perspective.



Σχήμα 6.4: Σμίκρυνση σε λειτουργία orthographic



Σχήμα 6.5: Σμίκρυνση σε λειτουργία perspective

Η επόμενη σημαντική λειτουργία είναι αυτή της περιστροφής (**Rotation)** του μοντέλου. Όπως και στη μεγέθυνση η λειτουργία αυτή υλοποιείται με κατάλληλη μετακίνηση της κά-

μερας. Συγκεκριμένα για την περιστροφή της κάμερας δεν περιστρέφεται η ίδια η κάμερα αλλά ο οδηγός που έχει ως γονέα. Έτσι η περιστροφή γίνεται γύρω από το κέντρο του μοντέλου και όχι γύρω από το κέντρο της κάμερας. Η περιστροφή γίνεται με τη χρήση ενός μόνο δακτύλου το οποίο ολισθαίνει στην οθόνη στην κατεύθυνση που είναι επιθυμητή μία περιστροφή και περιστρέφει κατάλληλα τον οδηγό της κάμερας. Επομένως απαιτείται η χρήση ενός μόνο αντικειμένου αφής για την υλοποίηση της. Στον κώδικα του πίνακα 6.2 παρουσιάζεται η υλοποίηση της περιστροφής μέσω ενός αντικειμένου αφής.

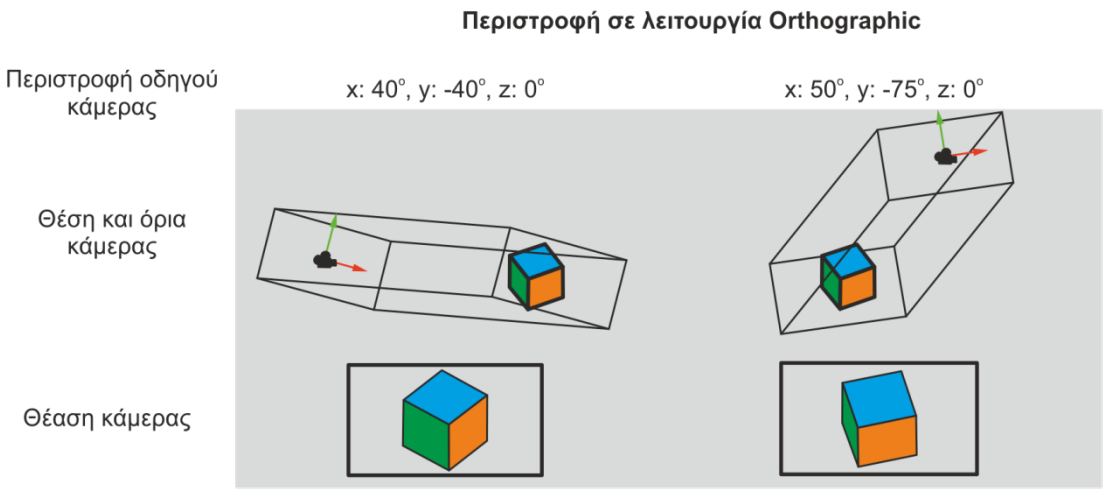
Περιστροφή μοντέλου με τη χρήση κάμερας

```
if (Input.touchCount == 1) {

    Touch touch = Input.GetTouch (0);

    if (touch.phase == TouchPhase.Began) {
        initTouch = touch;
    } else if (touch.phase == TouchPhase.Moved) {

      float deltaX = initTouch.position.x - touch.position.x;
      float deltaY = initTouch.position.y - touch.position.y;

      rotationX += deltaY * Time.deltaTime * rotationSpeed;
      rotationY -= deltaX * Time.deltaTime * rotationSpeed;

      cameraPivot.eulerAngles = new Vector3 (rotationX, rotationY, 0f);

    } else if (touch.phase == TouchPhase.Ended) {
      initTouch = new Touch ();
    }
}
```

Πίνακας 6.2: Κώδικας περιστροφής μοντέλου μέσω αφής

Στο παραπάνω τμήμα κώδικα ελέγχεται αρχικά εάν υπάρχει αφή από το χρήστη. Εάν αυτό ισχύει και το αντικείμενο αφής βρίσκεται στη φάση μετακίνησης, δηλαδή εάν το δάκτυλο έχει ολισθήσει στην οθόνη, υπολογίζεται η απόσταση μετακίνησης στους άξονες x και y στην οθόνη. Η κάθε απόσταση πολλαπλασιασμένη με μία παράμετρο ευαισθησίας προσδιορίζει κατά πόσο θα περιστραφεί κάθε άξονας του οδηγού της κάμερας. Επιπλέον η κάθε απόσταση πολλαπλασιάζεται με την μεταβλητή **Time.deltaTime**, εξασφαλίζοντας ότι η περιστροφή του οδηγού θα γίνεται σε μονάδες ανά δευτερόλεπτο και όχι ανά frame. Στο σχήμα 6.6 παρουσιάζεται ένα παράδειγμα περιστροφής της κάμερας.

Όπως φαίνεται το μοντέλο παραμένει στο κέντρο της οθόνης στην ίδια περιστροφή που βρισκόταν. Περιστρέφοντας τον οδηγό της κάμερας δημιουργείται η ψευδαίσθηση περιστροφής του ίδιου του μοντέλου. Σημειώνεται, ότι για τη λειτουργία perspective ο τρόπος περιστροφής της κάμερας δεν παρουσιάζει διαφορές.



Σχήμα 6.6: Παράδειγμα περιστροφής σε λειτουργία orthographic

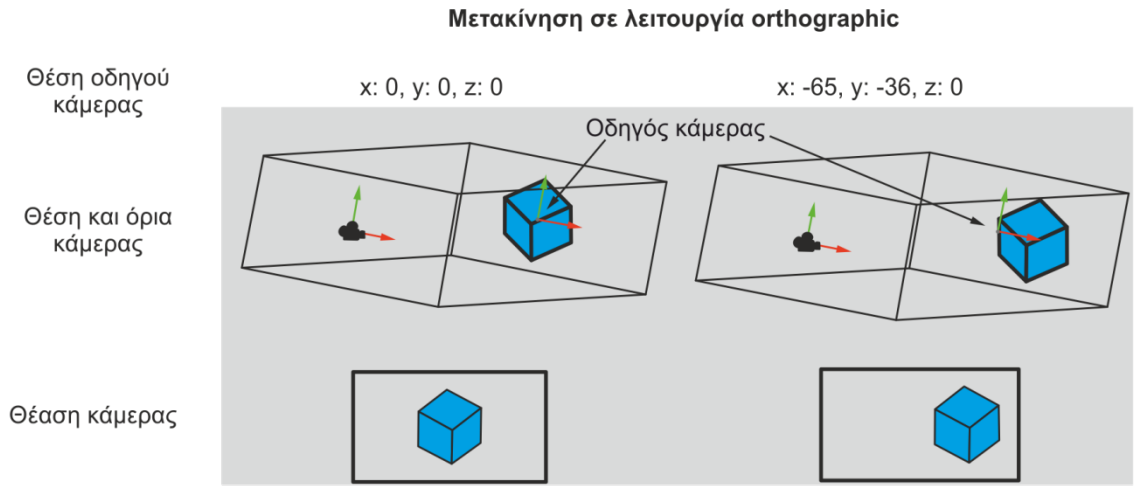
Όσον αφορά τη λειτουργία μετακίνησης του μοντέλου (**pan**), αυτή υλοποιείται και πάλι με τη μετακίνηση του οδηγού της κάμερας. Η μετακίνηση της ίδιας της κάμερας αποφεύγεται για να μην χαθεί η ευθυγράμμιση αυτής με τον οδηγό της. Η μετακίνηση γίνεται με τη χρήση δύο δακτύλων και στη συνέχεια την ολίσθηση αυτών επάνω στην οθόνη, έτσι ώστε η μεταξύ τους απόσταση να παραμένει σταθερή. Αν η απόσταση τους δεν παραμένει σταθερή, εκτελείται η λειτουργία μεγέθυνσης ή σμίκρυνσης. Η υλοποίηση παρουσιάζεται στον κώδικα του πίνακα 6.3.

Μετακίνηση μοντέλου με τη χρήση κάμερας

```
if (Input.touchCount == 2) {
   if (Mathf.Abs (deltaMagnitudeDiff) < 8f && Input.GetTouch (0).phase == TouchPhase.Moved && Inp
      ut.GetTouch (1).phase == TouchPhase.Moved) {

      Vector2 touchDelta = Input.GetTouch (0).deltaPosition;

      cameraPivot.Translate (new Vector3 (-touchDelta.x * moveSpeed, -touchDelta.y *  moveSpeed,
      0));
   }
}
```

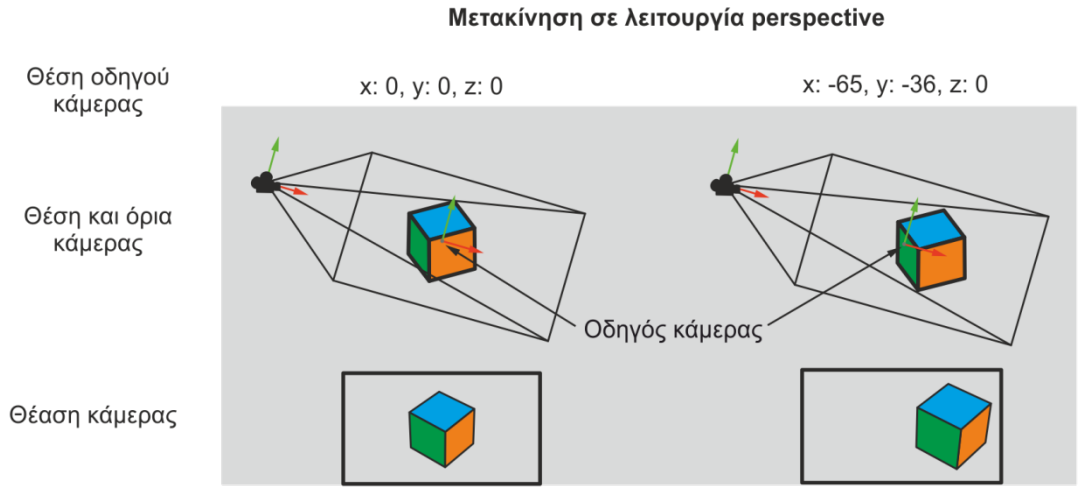
Πίνακας 6.3: Κώδικας μετακίνησης μοντέλου μέσω αφής

Στον κώδικα του πίνακα 6.3 η μεταβλητή **deltaMagnitudeDiff** αντιστοιχεί στη μεταβολή της απόστασης των δύο αντικειμένων αφής, δηλαδή των δύο δακτύλων. Αν η μεταβολή αυτή είναι μικρότερη από ένα ποσό, θεωρείται ότι τα δύο δάκτυλα παραμένουν στην ίδια απόσταση. Το πόσο μεταβολής έχει οριστεί ως 8 και αντιστοιχεί σε αυτό που πειραματικά δίνει τα βέλτιστα αποτελέσματα. Επομένως για μεταβολή μικρότερη του 8 ο οδηγός της κάμερας μετακινείται στους άξονες x και y, σε απόσταση που καθορίζεται από την ολίσθηση των δακτύλων στην οθόνη (μεταβλητή **touchDelta**), πολλαπλασιασμένη με μία μεταβλητή ευαισθησίας (**moveSpeed**). Στο σχήμα 6.7 παρουσιάζεται ένα παράδειγμα μετακίνησης του οδηγού της κάμερας για orthographic λειτουργία.



Σχήμα 6.7: Παράδειγμα μετακίνησης σε λειτουργία orthographic

Στο παραπάνω παράδειγμα παρατηρείται, ότι μετά την μετακίνηση ο οδηγός της κάμερας δεν βρίσκεται πλέον στο κέντρο του μοντέλου. Συνεπώς, η εκτέλεση μίας περιστροφής θα είχε ως αποτέλεσμα την περιστροφή γύρω από το κέντρο του οδηγού, ο οποίος δεν βρίσκεται στο κέντρο του στερεού μοντέλου. Αυτό διορθώνεται με την προσθήκη μίας λειτουργίας που θα αναφερθεί στη συνέχεια. Η μετακίνηση σε perspective λειτουργία γίνεται με τον ίδιο ακριβώς τρόπο. Στην περίπτωση όμως αυτή, επειδή όπως ήδη αναφέρθηκε υπάρχει η αίσθηση του βάθους σε μία θέαση, τότε η ίδια μετακίνηση ενός μοντέλου θα έχει ως αποτέλεσμα και μία αλλαγή στην όψη του. Αυτό παρουσιάζεται στο παράδειγμα μετακίνησης του σχήματος 6.8.



Σχήμα 6.8: Παράδειγμα μετακίνησης σε λειτουργία perspective

Σημειώνεται ότι οι κώδικες για όλες τις παραπάνω λειτουργίες τοποθετούνται στη συνάρτηση **Update()**, καθώς για την αποτελεσματική λειτουργία τους η εκτέλεση κάθε κώδικα πρέπει να γίνεται σε κάθε frame. Έτσι είναι διαθέσιμες κάθε στιγμή στο χρήστη της εφαρμογής.

Για την επαναφορά της κάμερας μαζί με τον οδηγό της στην προκαθορισμένη θέση, υλοποιήθηκε ακόμα η λειτουργία προσαρμογής (**Fit**). Η λειτουργία αυτή ουσιαστικά επαναφέρει το μοντέλο στα όρια της οθόνης εάν αυτό βρίσκεται έξω από αυτά και ορίζει τον οδηγό της κάμερας στο κέντρο του μοντέλου. Για την υλοποίηση της λειτουργίας αυτής, απαιτεί-

ται γνώση των διαστάσεων του μοντέλου. Η μεταβλητή **bounds.size** που βρίσκεται στην κλάση **MeshFilter** του μοντέλου, περιέχει τις διαστάσεις του σε κάθε άξονα. Από τις διαστάσεις αυτές αποθηκεύεται η μεγαλύτερη διάσταση. Στη συνέχεια αναλόγως εάν η κάμερα βρίσκεται σε λειτουργία perspective ή orthographic, γίνεται αναπροσαρμογή του μεγέθους της (orthographic size) ή της απόστασης της από το μοντέλο αντίστοιχα. Το επιθυμητό αποτέλεσμα είναι το μοντέλο να προσαρμόζεται όσο το δυνατόν καλύτερα στα όρια της οθόνης. Αν **maxBound** είναι η μεγαλύτερη διάσταση του μοντέλου, τότε για τη λειτουργία orthographic η αναπροσαρμογή του μεγέθους της κάμερας γίνεται σύμφωνα με την σχέση:

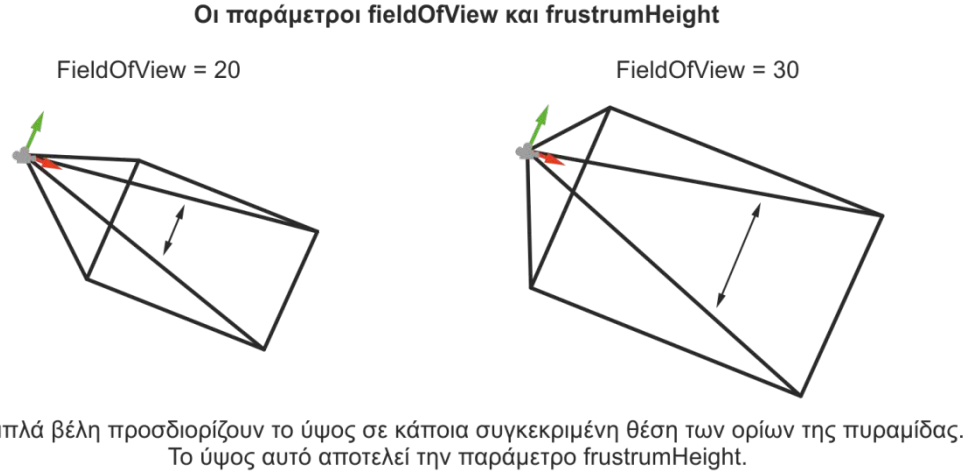
$$OrthographicSize = maxBound\ /1.75 + distanceOffset$$

Η παραπάνω σχέση έχει βρεθεί πειραματικά έτσι ώστε να δίνει τα βέλτιστα αποτελέσματα. Η μεταβλητή **distanceOffset** αποτελεί παράμετρο μικρορύθμισης και ρυθμίζει τις αποστάσεις του μοντέλου από τα πάνω και κάτω όρια της οθόνης.

Για τη λειτουργία perspective τα όρια τις κάμερας είναι διαφορετικά σε κάθε απόσταση από αυτήν και δημιουργούν ένα σχήμα πυραμίδας όπως φαίνεται για παράδειγμα στο σχήμα 6.8. Όπως φαίνεται στο σχήμα 6.5 διαφορετική απόσταση της κάμερας από το μοντέλο προκαλεί μεγέθυνση η σμίκρυνση αυτού. Συνεπώς θα πρέπει να βρεθεί η κατάλληλη απόσταση της κάμερας από το μοντέλο. Σύμφωνα με τη βιβλιογραφία της Unity η παρακάτω σχέση βρίσκει τη ζητούμενη απόσταση:

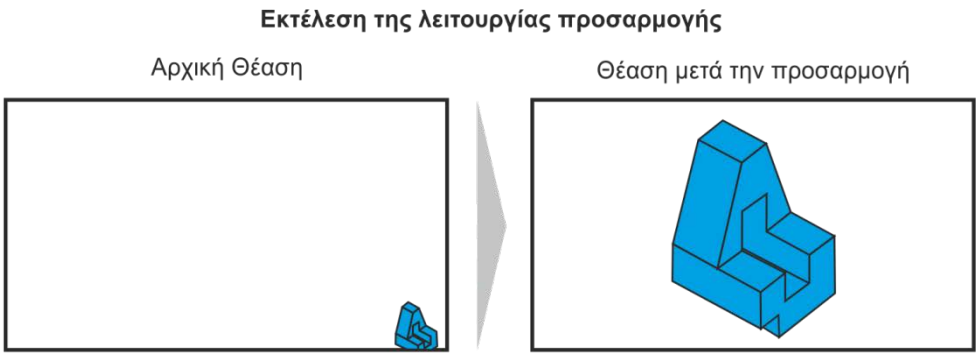
$$Distance = frustrumHeight * 0.5 / \tan(fieldOfView * 0.5 * Math.Deg2Rad)$$

Στην παραπάνω σχέση η **Math.Deg2Rad** είναι η σταθερά μετατροπής από μοίρες σε ακτίνια. Η παράμετρος **frustrumHeight** είναι το ύψος των ορίων της οθόνης, ενώ η παράμετρος **fieldOfView** ορίζει το εύρος των ορίων της πυραμίδας της κάμερας. Οι παράμετροι frustrumHeight και fieldOfView παρουσιάζονται στο σχήμα 6.9.



Σχήμα 6.9: Οι παράμετροι fieldOfView και frustrumHeight

Έτσι για την περίπτωση της λειτουργίας προσαρμογής, η παράμετρος frustrumHeight ταυτίζεται με τη μέγιστη διάσταση του μοντέλου maxBound. Η παράμετρος fieldOfView παραμένει σταθερή και ίση με 60. Στο σχήμα 6.10 παρουσιάζεται ένα παράδειγμα της λειτουργίας προσαρμογής.



Σχήμα 6.10: Παράδειγμα λειτουργίας προσαρμογής

Όπως φαίνεται το μοντέλο βρίσκεται ακριβώς στα όρια της οθόνης μετά την εκτέλεση της προσαρμογής. Σημειώνεται ότι για διαφορετικά μοντέλα οι αποστάσεις από τα όρια της οθόνης ενδέχεται να διαφέρουν, λόγω της γεωμετρίας των μοντέλων και πιθανές αποκλίσεις της παραμέτρου distanceOffset. Στον πίνακα 6.4 παρουσιάζεται ο κώδικας της λειτουργίας προσαρμογής.

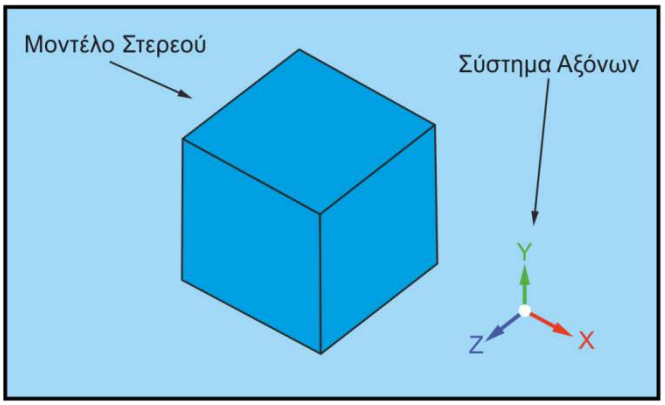
Λειτουργία προσαρμογής

```
if (cameraComponent.orthographic) {
   cameraComponent.orthographicSize = maxBound / 1.75f + distanceOffset;

} else {
   float distance = maxBound * 0.5f / Mathf.Tan (0.5f * cameraComponent.fieldOfView * Mathf.Deg2R
ad);
   this.transform.localPosition = new Vector3 (this.transform.localPosition.x, this.transform.loc
alPosition.y, -distance * distanceOffset);
}
```

Πίνακας 6.4: Κώδικας λειτουργίας προσαρμογής

6.4 Το σύστημα αξόνων

Για την καλύτερη αντίληψη του χώρου κατά την προβολή των τρισδιάστατων μοντέλων στην εφαρμογή, σχεδιάστηκε ένα σύστημα αξόνων το οποίο τοποθετείται κάτω δεξία της οθόνης. Το σύστημα αξόνων αποτελείται από τους τρεις άξονες του μοντέλου (x, y και z) και είναι απαραίτητο να δείχνει στη σωστή κατεύθυνση κατά την περιστροφή της κάμερας που αναφέρθηκε προηγουμένως. Το σύστημα αυτό παρουσιάζεται στο σχήμα 6.11.



Σχήμα 6.11: Θέση του συστήματος αξόνων στην οθόνη της εφαρμογής

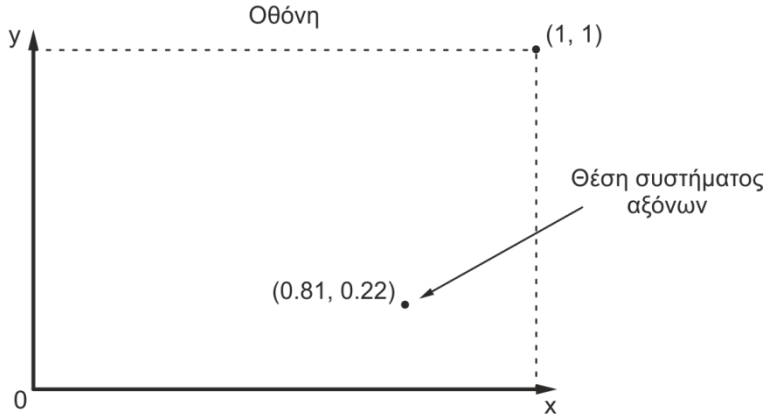
Για την ομαλή λειτουργία του συστήματος αξόνων κατά τη διάρκεια περιστροφών, δημιουργήθηκε μία επιπλέον κάμερα στην εφαρμογή στην οποία είναι ορατό μόνο το σύστημα αξόνων στη σκηνή. Το σύστημα αξόνων παραμένει πάντα σταθερό και δεν εφαρμόζονται περιστροφές σε αυτό, όπως και στο μοντέλο που βρίσκεται στη σκηνή. Η περιστροφή εφαρμόζεται στη δεύτερη κάμερα που δημιουργήθηκε και είναι όμοια με την κύρια κάμερα της σκηνής. Ουσιαστικά. η δεύτερη κάμερα βρίσκεται πάντα στην ίδια θέση και περιστροφή με την κύρια κάμερα. Επειδή δεν είναι δυνατόν να υπάρχουν δύο κάμερες ταυτόχρονα, η δεύτερη κάμερα προβάλλεται μέσω ενός renderer στο περιβάλλον διεπαφής χρήστη της εφαρμογής (**UI Component**). Επομένως για οποιαδήποτε θέαση της κύριας κάμερας, το σύστημα αξόνων θα είναι πάντα ορατό στην οθόνη. Η τοποθέτηση του συστήματος αξόνων κάτω δεξία της οθόνης γίνεται για κάθε θέαση της δεύτερης κάμερας και με τη χρήση της συνάρτησης **ViewportToWorldPoint,** όπως φαίνεται στον κώδικα του πίνακα 6.5.

Τοποθέτηση συστήματος αξόνων κάτω δεξία της οθόνης

```
position = axisCamera.ViewportToWorldPoint (new Vector3 (0.81f, 0.22f, axisCamera.farClipPlane -
 100));
axisObject.position = position;
```

Πίνακας 6.5: Κώδικας τοποθέτησης συστήματος αξόνων

Η συνάρτηση μετασχηματίζει μία θέση (x,y) από το χώρο της οθόνης (**Viewport Space**), σε μία θέση στον τρισδιάστατο χώρο (x, y, z) στη σκηνή της Unity (**World Space)**. Μία θέση στα όρια της οθόνης μπορεί να πάρει τιμές στο [0,1] όπως παρουσιάζεται στο σχήμα 6.12. Η τρίτη παράμετρος αφορά την απόσταση που θα βρίσκεται το αντικείμενο από την κάμερα, στη θέση που αυτό ορίστηκε.



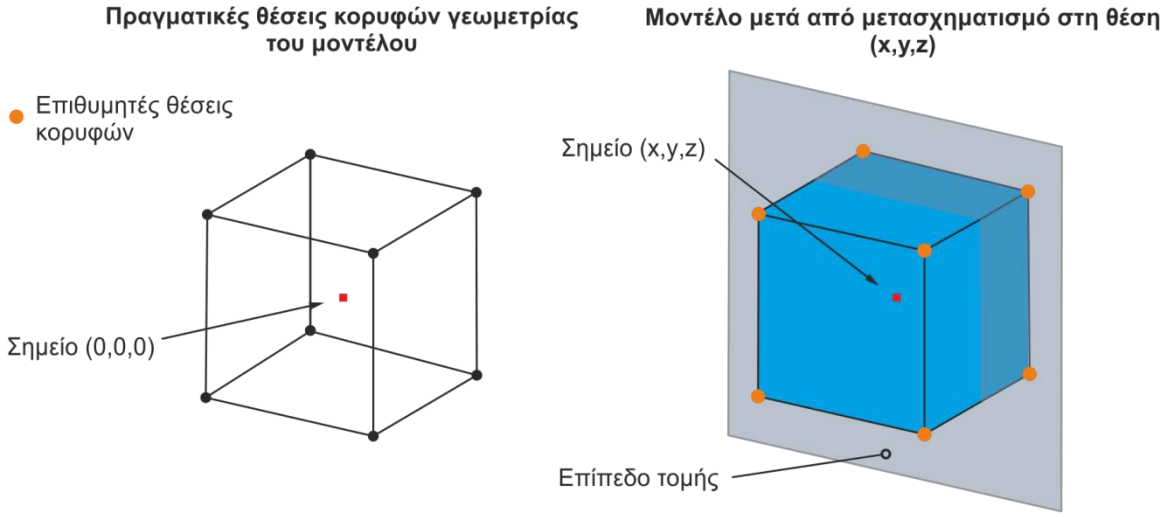
Σχήμα 6.12: Συντεταγμένες θέσεων στο χώρο της οθόνης (Viewport Space)

6.5 Λειτουργίες με μετασχηματισμό του ίδιου του μοντέλου

Οι λειτουργίες που αναφέρθηκαν παραπάνω ήταν, όπως είναι προφανές, πολύ πιο εύκολο να υλοποιηθούν με τον μετασχηματισμό του ίδιου του μοντέλου και όχι της κάμερας. Δηλαδή, η περιστροφή, μετακίνηση και η μεγέθυνση μπορούσαν να εφαρμοστούν στο ίδιο το μοντέλο αποφεύγοντας έτσι οποιαδήποτε κίνηση της κάμερας. Στην παρούσα εφαρμογή κάτι τέτοιο δεν θα ήταν ιδανικό, καθώς πιθανώς θα δημιουργούσε προβλήματα κατά τη λειτουργία τομής του μοντέλου.

Στην Unity η γεωμετρία ενός μοντέλου αποθηκεύεται στον τοπικό χώρο του μοντέλου (**Local Space)**. Η μετακίνηση, περιστροφή ή μεγέθυνση του μοντέλου αφορούν μετασχηματισμούς στον παγκόσμιο χώρο της σκηνής (**World Space**) και όχι στον τοπικό χώρο

του μοντέλου. Η εφαρμογή τέτοιων μετασχηματισμών έχει ως αποτέλεσμα το μοντέλο να προβάλλεται στην επιθυμητή θέση, αλλά τα δεδομένα της γεωμετρίας του να παραμένουν στις αρχικές θέσεις τους στον τοπικό χώρο του μοντέλου. Η ταυτόχρονη μετακίνηση των δεδομένων γεωμετρίας του μοντέλου στη θέση του μοντέλου, θα είχε ως αποτέλεσμα προβλήματα απόδοσης στις εφαρμογές. Το πρόβλημα που δημιουργείται στην παρούσα εφαρμογή, είναι η εξέταση των κορυφών γεωμετρίας κατά την εκτέλεση του αλγορίθμου τομής. Η εξέταση εάν οι κορυφές βρίσκονται στο θετικό ή αρνητικό ημιεπίπεδο τομής δίνει λανθασμένα αποτελέσματα εάν το μοντέλο έχει μετασχηματιστεί όπως αναφέρθηκε, καθώς οι κορυφές γεωμετρίας παραμένουν πάντα σε σταθερή θέση. Ένα παράδειγμα λανθασμένου υπολογισμού στον αλγόριθμο τομής παρουσιάζεται στο σχήμα 6.13.



Σχήμα 6.13: Παράδειγμα λανθασμένου υπολογισμού στον αλγόριθμο τομής

Στο παράδειγμα του σχήματος 6.13 οι κορυφές έπρεπε να βρίσκονται στις επιθυμητές θέσεις κορυφών, προκειμένου ο αλγόριθμος τομής να εκτελεστεί χωρίς λανθασμένα αποτελέσματα. Για να συμβεί αυτό θα έπρεπε να μετασχηματιστούν οι κορυφές από τον τοπικό χώρο του μοντέλου στον παγκόσμιο χώρο της σκηνής. Αυτό γίνεται εύκολα με τη χρήση έτοιμων συναρτήσεων που παρέχει η Unity. Παρόλα αυτά, αυτός ο μετασχηματισμός κορυφών οδηγεί σε αποκλίσεις δεκαδικών ψηφίων της θέσης κάθε κορυφής, οι οποίες δημιουργούν μικρές ανακρίβειες κατά τον αλγόριθμο τομής και πιθανά σφάλματα του. Επομένως ο μετασχηματισμός του ίδιου του μοντέλου αποφεύγεται.

7. ΤΟ ΚΥΡΙΟ ΠΕΡΙΒΑΛΛΟΝ ΤΗΣ ΕΦΑΡΜΟΓΗΣ

Η διαχείριση των τρισδιάστατων μοντέλων γίνεται στο κύριο περιβάλλον της εφαρμογής. Στο περιβάλλον αυτό μέσω μίας γραφικής διεπαφής (User Interface), ο χρήστης έχει τη δυνατότητα να αλληλεπιδράσει με κάποιο μοντέλο και να εφαρμόσει σε αυτό συγκεκριμένες λειτουργίες, οι οποίες συμβάλλουν στην καλύτερη κατανόηση και εποπτεία του στερεού. Στο κεφάλαιο αυτό θα περιγραφούν αναλυτικά οι δυνατότητες που προσφέρονται και θα εξηγηθεί ο τρόπος που αυτές υλοποιούνται.

7.1 Διαχείριση Παραθύρων

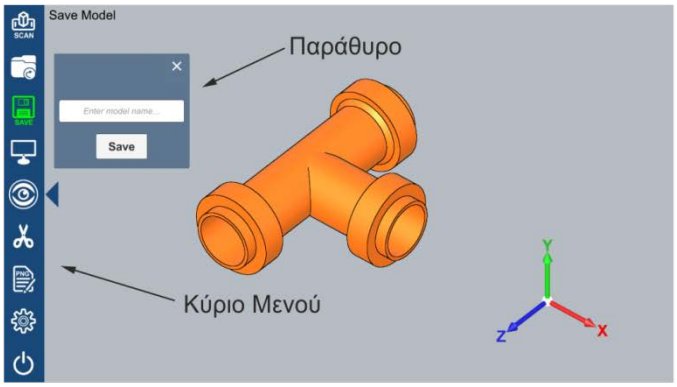
Βασική προϋπόθεση για την ορθή λειτουργία της εφαρμογής, είναι η ομαλή διαχείριση παραθύρων της γραφικής διεπαφής χρήστη. Η εφαρμογή αποτελείται από ένα κύριο μενού στα αριστερά της οθόνης, το οποίο προσφέρει διάφορες επιλογές. Κάποιες από αυτές, οδηγούν στην εμφάνιση ενός δεύτερου μενού επιλογών στα δεξιά της οθόνης ή στο άνοιγμα κάποιου παραθύρου. Για την κατασκευή των παραθύρων και των μενού επιλογών χρησιμοποιήθηκε το αντικείμενο (**UI → Panel**) που προσφέρεται στο περιβάλλον της Unity. Κάθε αντικείμενο Panel περιέχει ένα η περισσότερα κουμπιά (**UI → Button**) ανάλογα με τις απαιτήσεις. Για κάθε παράθυρο ή μενού διατηρείται μία μεταβλητή που δηλώνει εάν αυτό είναι ενεργό. Το άνοιγμα και κλείσιμο τους γίνεται με μία απλή συνάρτηση, όπως αυτή του πίνακα 7.1. Κάθε φορά που αυτή καλείται, το παράθυρο εμφανίζεται ή εξαφανίζεται αναλόγως την τιμή της μεταβλητής που δηλώνει την κατάσταση του.

Ενεργοποίηση και απενεργοποίηση παραθύρου

```
public void togglePropertiesPanel(){
     if (propertiesPanelActive) {
         propertiesPanel.SetActive (false);
         propertiesPanelActive = false;
     } else {
         propertiesPanel.SetActive (true);
         propertiesPanelActive = true;
    }
 }
```

Πίνακας 7.1: Κώδικας ενεργοποίησης και απενεργοποίησης παραθύρου

Μερικές φορές είναι επιθυμητό όταν ανοίγει κάποιο νέο μενού όλα τα υπόλοιπα να απενεργοποιούνται, έτσι ώστε να διευκολύνεται η διαχείριση της εφαρμογής. Για τον λόγο αυτό δημιουργήθηκε μία συνάρτηση που απενεργοποιεί όλα τα παράθυρα και μενού της εφαρμογής ενώ το κύριο μενού στα αριστερά παραμένει πάντα ενεργό. Στο σχήμα 7.1 παρουσιάζεται το κύριο μενού και το παράθυρο save της εφαρμογής.



Σχήμα 7.1: Κύριο μενού της εφαρμογής

7.2 Όψεις (Views)

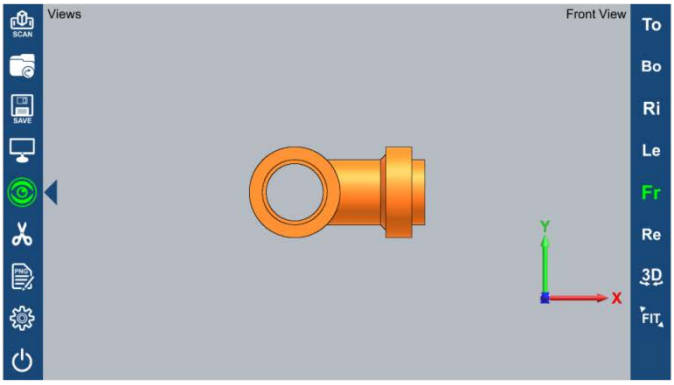
Μία βασική δυνατότητα που προσφέρεται από την εφαρμογή, είναι η προβολή των διαφορετικών όψεων του στερεού από το μενού views. Οι διαφορετικές όψεις υλοποιούνται με απλή περιστροφή του οδηγού της κάμερας όπως αυτή έχει περιγραφεί στο κεφάλαιο 6. Συνολικά προσφέρονται 6 όψεις. Για κάθε διαφορετική όψη που προσφέρεται, η περιστροφή του οδηγού ορίζεται στις κατάλληλες μοίρες ώστε το αποτέλεσμα θέασης να είναι η επιθυμητή όψη. Για παράδειγμα, η υλοποίηση της πίσω όψης γίνεται όπως φαίνεται στον πίνακα 7.2.

Υλοποίηση πίσω όψης

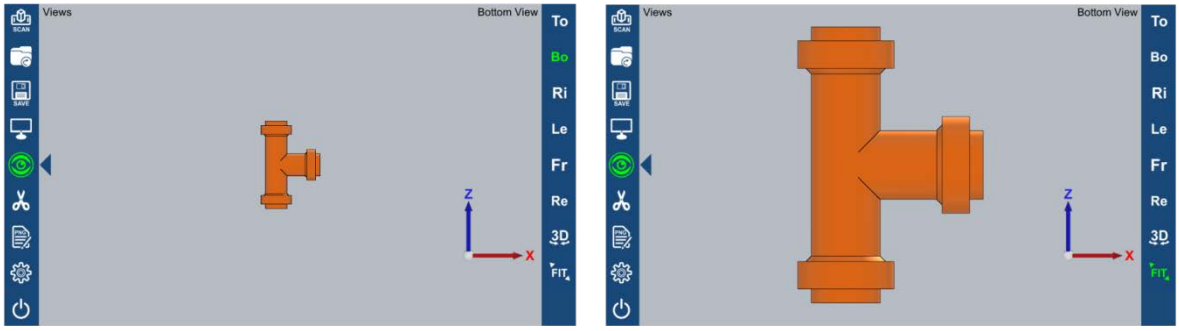
```
rotationX = 0;
rotationY = 180;
cameraPivot.eulerAngles = new Vector3 (rotationX, rotationY, 0);
```

Πίνακας 7.2: Κώδικας υλοποίησης πίσω όψης

Εκτός από τις όψεις, το μενού περιλαμβάνει την προκαθορισμένη επιλογή θέασης του μοντέλου καθώς και την λειτουργία προσαρμογής, η οποία έχει επίσης περιγραφεί στο κεφάλαιο 6. Στο στιγμιότυπο του σχήματος 7.2 παρουσιάζεται το μενού των όψεων μετά από εφαρμογή της πρόοψης ενώ στο σχήμα 7.3 παρουσιάζεται μία εφαρμογή της λειτουργίας προσαρμογής.



Σχήμα 7.2: Εφαρμογή πρόοψης

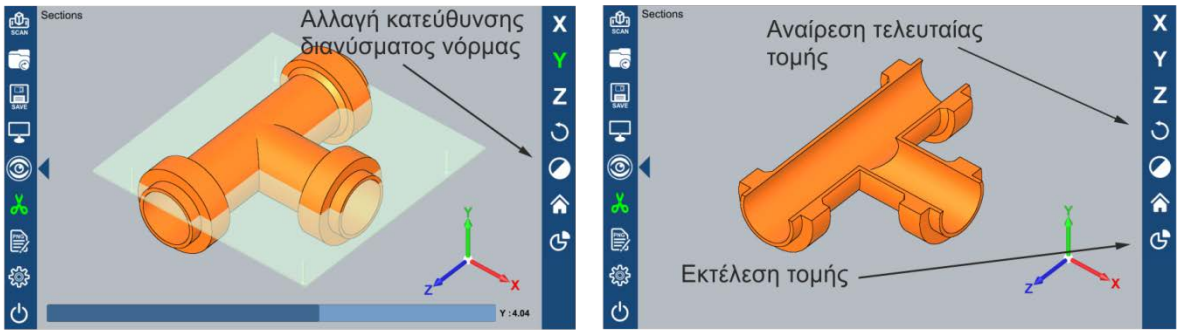


Σχήμα 7.3: Αποτέλεσμα εφαρμογής της λειτουργίας προσαρμογής για την πίσω όψη

7.3 Τομές (Sections)

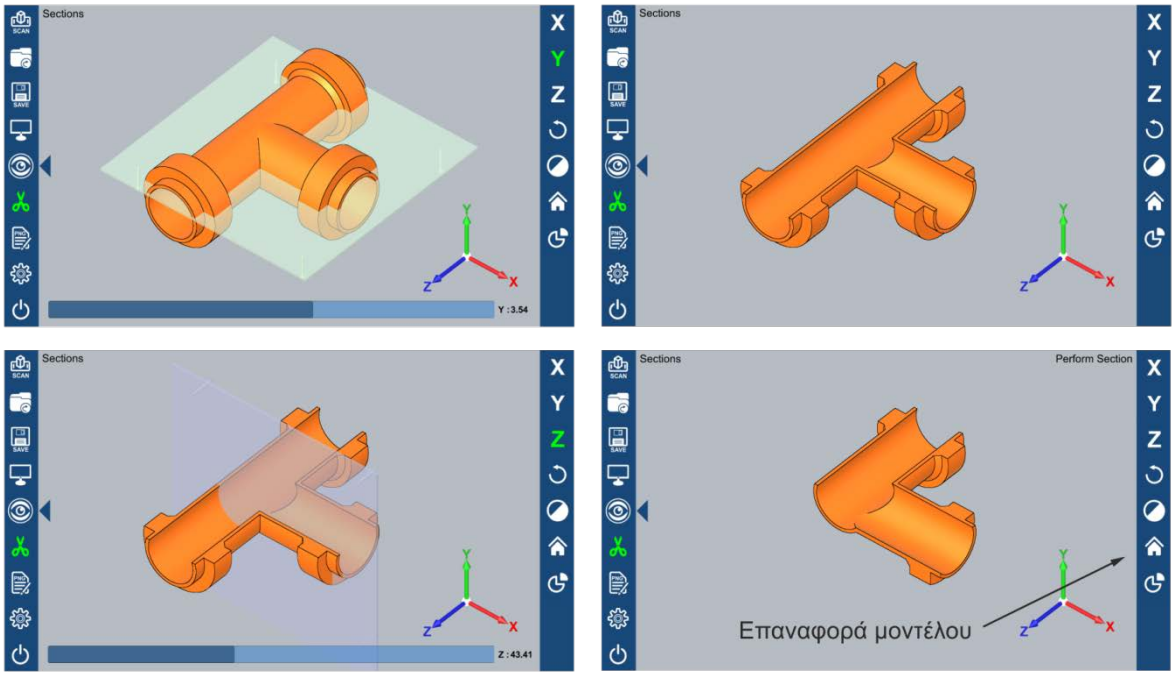
Η βασικότερη δυνατότητα που παρέχει η εφαρμογή είναι η εκτέλεση τομών κατά μήκος των τριών αξόνων x, y και z. Η διαδικασία τομής έχει περιγραφεί αναλυτικά στο κεφάλαιο 5. Ο χρήστης έχει τη δυνατότητα να μετακινήσει το επίπεδο σε όποια θέση επιθυμεί, σε

κάποιον από τους τρεις άξονες του μοντέλου. Για κάθε επίπεδο τομής επιλέγεται η κατεύθυνση του διανύσματος της νόρμας του. Το διάνυσμα αυτό προσδιορίζει το μέρος του μοντέλου που διατηρείται μετά την τομή. Εφόσον ο χρήστης εφαρμόσει μία τομή στο μοντέλο, έχει την δυνατότητα να την αναιρέσει πατώντας το αντίστοιχο κουμπί. Τέλος σε περίπτωση αρκετών διαδοχικών τομών, το κουμπί Restore Original Model (εικονίδιο Home) επαναφέρει το μοντέλο στην κατάσταση που ήταν πριν από οποιαδήποτε τομή. Στο σχήμα 7.4 παρουσιάζεται το μενού τομών και μία τυχαία τομή του μοντέλου στον y άξονα.



Σχήμα 7.4: Παράδειγμα τομής στον y άξονα

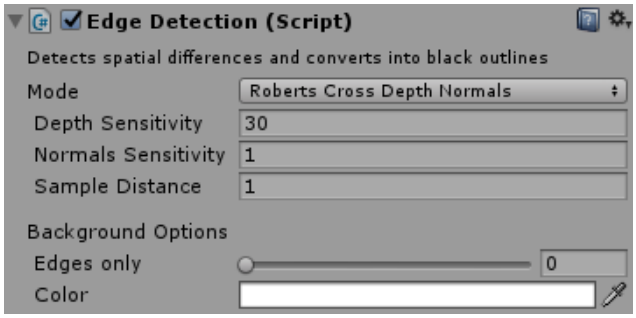
Στο σχήμα 7.5 παρουσιάζεται ένα παράδειγμα δύο διαδοχικών τομών. Η πρώτη στον y άξονα ενώ η δεύτερη στον z άξονα.



Σχήμα 7.5: Παράδειγμα δύο διαδοχικών τομών

7.4 Επιλογές Εμφάνισης (Display)

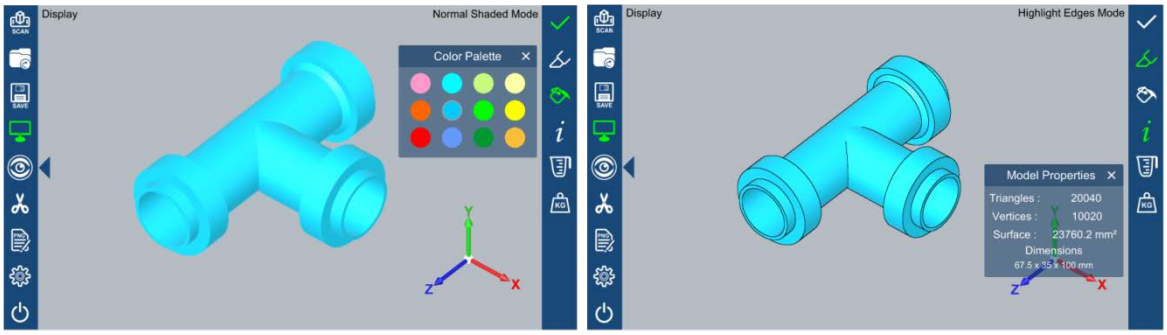
Το μενού εμφάνισης προσφέρει στο χρήστη τη δυνατότητα εφαρμογής απλών αλλαγών που σχετίζονται με την εμφάνιση του μοντέλου, καθώς και την προβολή βασικών πληροφοριών που σχετίζονται με αυτό. Συγκεκριμένα παρέχεται η δυνατότητα εμφάνισης ή όχι των ακμών του μοντέλου, η αλλαγή χρώματος του μοντέλου καθώς και η εμφάνιση πληροφοριών της ψηφιακής δομής του μοντέλου.



Σχήμα 7.6: Ρυθμίσεις αναγνώρισης ακμών

Η επιλογή εμφάνισης ακμών υλοποιείται μέσω του script Edge Detection που περιέχεται στα βασικά εργαλεία τύπου Image Effects που παρέχει η Unity. Το στοιχείο αυτό τοποθετείται στην κάμερα της εφαρμογής και τονίζει τις ακμές των αντικειμένων που βρίσκονται στα όρια θέασης της κάμερας. Επιπλέον παρέχεται η δυνατότητα ρύθμισης διαφόρων παραμέτρων σχετικά με την ευαισθησία βάθους των ακμών (**Depth Sensitivity**), το πάχος τους (**Sample Distance**), ή το είδος αναγνώρισης ακμών (**Mode**). Έπειτα από πειραματισμό οι τιμές παραμέτρων που θεωρήθηκαν ως ιδανικές παρουσιάζονται στο σχήμα 7.6.

Η αλλαγή χρώματος του αντικειμένου γίνεται μέσω μίας παλέτας που περιέχει τα πιο βασικά χρώματα. Η υλοποίηση γίνεται απλά με την αλλαγή χρώματος του στοιχείου του υλικού (**Material**) που υπάρχει στο αντικείμενο. Στο σχήμα 7.7 παρουσιάζεται ένα αντικείμενο με και χωρίς αναγνώριση ακμών καθώς και η παλέτα χρωμάτων και το παράθυρο πληροφοριών.



Σχήμα 7.7: Αναγνώριση ακμών και παλέτα χρωμάτων

Στο παράθυρο πληροφοριών του αντικειμένου εμφανίζονται πληροφορίες σχετικά με τον αριθμό τριγώνων του, τον αριθμό των κορυφών του, την επιφάνειά του σε mm^2 καθώς και οι διαστάσεις του σε mm. Ο αριθμός κορυφών και τριγώνων παρέχονται απευθείας από την κλάση Mesh, όπως αναφέρεται και στο κεφάλαιο 5. Το ίδιο συμβαίνει και με τις διαστάσεις του μέσω της κλάσης Bounds. Σημειώνεται ότι οι διαστάσεις είναι πραγματικές, καθώς η σχεδίαση των μοντέλων έχει γίνει με ακρίβεια μέσω εξωτερικού προγράμματος CAD. Ο υπολογισμός επιφάνειας δεν παρέχεται απευθείας και πρέπει να υλοποιηθεί. Αυτό γίνεται με τον υπολογισμό της επιφάνειας κάθε τριγώνου της γεωμετρίας ξεχωριστά και στη συνέχεια την άθροιση αυτών των επιφανειών. Πιο συγκεκριμένα, έστω ένα αντικείμενο με γεωμετρία που αποτελείται από n τρίγωνα. Τότε για κάποιο τρίγωνο ABC θα ισχύει ότι:

$$triangleHeight = \frac{\left|\overrightarrow{CB} \times \overrightarrow{CA}\right|}{\left|\overrightarrow{CB}\right|}$$

$$triangleSurface = \frac{\left|\overrightarrow{CB}\right| triangleHeight}{2}$$

Επομένως η επιφάνεια του αντικειμένου δίνεται από τη σχέση:

$$objectSurface = \sum_{n} triangleSurface(n)$$

Ο υπολογισμός επιφάνειας ενός τριγώνου παρουσιάζεται στον κώδικα του πίνακα 7.3.

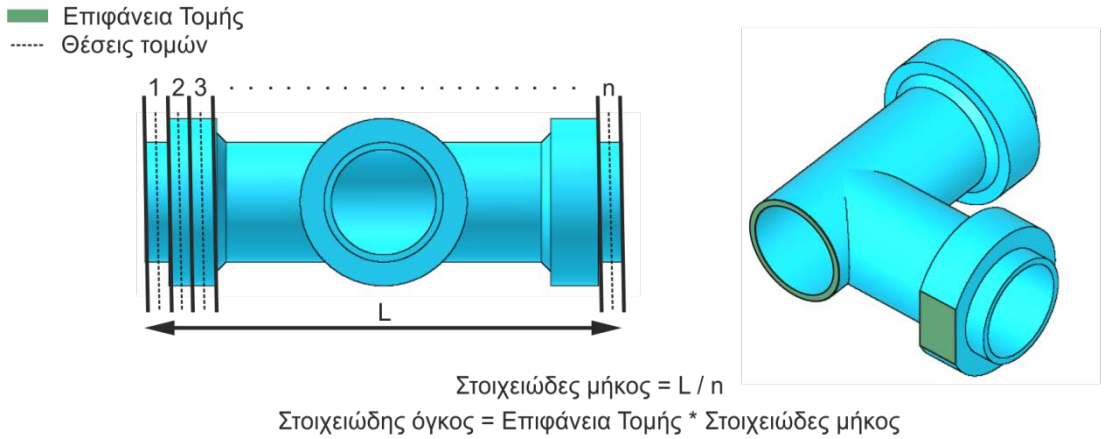
Υπολογισμός επιφάνειας τριγώνου

```
float triangleBase = Vector3.Distance(b,c);

Vector3 ca = a - c;
Vector3 cb = b - c;

float triangleHeight = (Vector3.Cross (cb, ca).magnitude) / cb.magnitude;
float triangleSurface = (triangleBase * triangleHeight) / 2;
```

Πίνακας 7.3: Κώδικας υπολογισμού επιφάνειας τριγώνου

Ο υπολογισμός της επιφάνειας του αντικειμένου με τον παραπάνω τρόπο προσεγγίζει ικανοποιητικά την πραγματική τιμή επιφάνειας του μοντέλου. Η ακρίβεια αυξάνεται όσο μεγαλύτερος είναι ο αριθμός των τριγώνων της γεωμετρίας για ένα πολύπλοκο μοντέλο. Δύο επιπλέον επιλογές που παρέχονται στο συγκεκριμένο μενού, είναι ο υπολογισμός του όγκου του αντικειμένου καθώς και ο υπολογισμός του βάρους του μετά από επιλογή συγκεκριμένου υλικού.

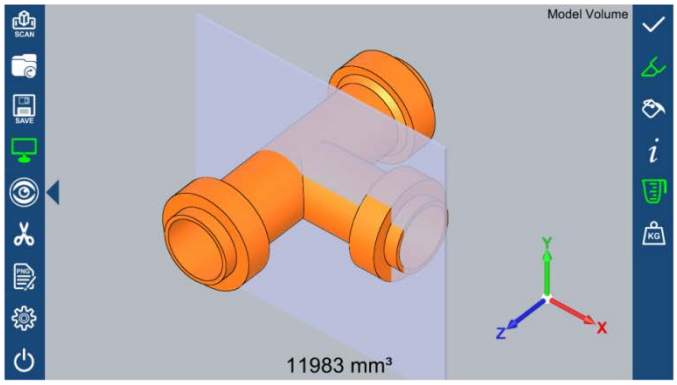
Για τον υπολογισμό του όγκου, το στερεό χωρίζεται σε n κομμάτια κατά μήκος του Z άξονα του. Για κάθε κομμάτι υπολογίζεται ο στοιχειώδης όγκος του. Το τελικό αποτέλεσμα είναι το άθροισμα κάθε στοιχειώδους όγκου. Για τον υπολογισμό του στοιχειώδους όγκου χρησιμοποιούνται διαδοχικές τομές και υπολογισμός της επιφάνειας κάθε τομής. Η διαδικασία παρουσιάζεται στο σχήμα 7.8.



Σχήμα 7.8: Υπολογισμός στοιχειώδους όγκου

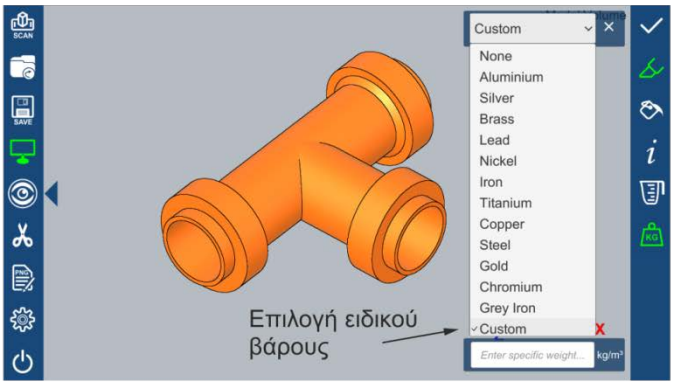
Όπως φαίνεται σχήμα 7.8, οι θέσεις τομών είναι στη μέση κάθε στοιχειώδους κομματιού. Ο συνολικός όγκος του αντικειμένου είναι το άθροισμα κάθε στοιχειώδους όγκου. Όπως είναι προφανές όσο μεγαλύτερος είναι ο αριθμός n των κομματιών, τόσο πιο ακριβές θα είναι το αποτέλεσμα. Από το κεφάλαιο 5, υπενθυμίζεται ότι για κάποια τομή, κατασκευάζεται το τμήμα που απαιτείται για να κλείσει τον ανοικτό χώρο που δημιουργείται μετά την τομή. Η επιφάνεια του τμήματος αυτού υπολογίζεται όπως προηγουμένως, δηλαδή υπο-

λογίζοντας την επιφάνεια κάθε τριγώνου που περιέχει. Ένα τέτοιο τμήμα παρουσιάζεται επίσης στο σχήμα 7.8 με πράσινη σκίαση. Πειραματικά στην παρούσα εφαρμογή ο αριθμός των στοιχειωδών κομματιών ορίζεται σε 100 για λόγους καλύτερης απόδοσης. Ο υπολογισμός του όγκου γίνεται με τη σάρωση του αντικειμένου από ένα επίπεδο τομής στον Z άξονα. Για κάθε στοιχειώδες όγκο που υπολογίζεται, ενημερώνεται η τιμή του συνολικού όγκου στο κέντρο της οθόνης. Το τελικό αποτέλεσμα παρουσιάζεται όταν η σάρωση από το επίπεδο ολοκληρωθεί. Στο σχήμα 7.9 παρουσιάζεται μία ενδιάμεση κατάσταση κατά τον υπολογισμό του όγκου.



Σχήμα 7.9: Ενδιάμεση κατάσταση κατά τη διάρκεια υπολογισμού του όγκου

Τέλος η διαδικασία υπολογισμού του βάρους είναι όμοια με αυτή του υπολογισμού του όγκου με τη μόνη διαφορά ότι κάθε στοιχειώδης όγκος πρέπει να πολλαπλασιαστεί επιπλέον με το ειδικό βάρος του υλικού. Σημειώνεται ότι ο χρήστης έχει τη δυνατότητα να υπολογίσει το βάρος του υλικού χρησιμοποιώντας κάποιο δικό του ειδικό βάρος που δίνει ως είσοδο επιλέγοντας την επιλογή Custom. Στο σχήμα 7.10 παρουσιάζονται τα διαθέσιμα υλικά για τον υπολογισμό του βάρους.



Σχήμα 7.10: Διαθέσιμα υλικά για τον υπολογισμό του βάρους

7.5 Αποθήκευση και Ανάκτηση (Save – Open)

Στο κεφάλαιο 4 αναφέρεται ότι η εισαγωγή μοντέλων γίνεται μετά από επιλογή και μεταφορά τους από το περιβάλλον επαυξημένης πραγματικότητας. Τα μοντέλα αυτά αποθηκεύονται online, οπότε δεν υπάρχει η δυνατότητα προβολής αυτών αν η συσκευή δεν έχει πρόσβαση στο διαδίκτυο. Για τον λόγο αυτό μετά τη λήψη και μεταφορά ενός μοντέλου στο κύριο περιβάλλον της σκηνής, υπάρχει η δυνατότητα αποθήκευσης του τοπικά στη συσκευή. Η αποθήκευση μπορεί να γίνει και μετά την εφαρμογή κάποιας τομής στο μοντέλο. Στην περίπτωση αυτή αποθηκεύεται η γεωμετρία του μοντέλου που προκύπτει μετά από την τομή σε αυτό. Η αποθήκευση γίνεται μέσω της επιλογής Save στο κύριο μενού αριστερά και τον ορισμό ενός ονόματος αποθήκευσης.

Για την αποθήκευση ενός μοντέλου, το μόνο που απαιτείται να αποθηκευτεί είναι τα δεδομένα της γεωμετρίας του. Συγκεκριμένα αυτά είναι οι πίνακες κορυφών (vertices), διανυσμάτων νόρμας (normals), τριγώνων (triangles) και το όνομα του μοντέλου. Η αποθήκευση γίνεται με τη χρήση της μεθόδου σειριοποίησης (Serialization). Ως σειριοποίηση ορίζεται η διαδικασία μετάφρασης δομών δεδομένων (π.χ. πίνακες) σε μία μορφή που μπορεί να αποθηκευτεί (π.χ. ένα αρχείο) ή να μεταφερθεί (π.χ. μέσω δικτύου), έτσι ώστε αργότερα να μπορεί να ανακατασκευαστεί η αρχική δομή από το αποθηκευμένο αρχείο ή την ολοκλήρωση της μεταφοράς. Στην παρούσα εφαρμογή η γεωμετρία αποθηκεύεται σε ένα αρχείο που φέρει το όνομα αποθήκευσης. Η σειριοποίηση και αποθήκευση υλοποιείται μέσω των κλάσεων BinaryFormatter και Filestream και των συναρτήσεων που παρέχουν, όπως παρουσιάζεται στον πίνακα 7.4.

Σειριοποίηση και αποθήκευση γεωμετρίας μοντέλου

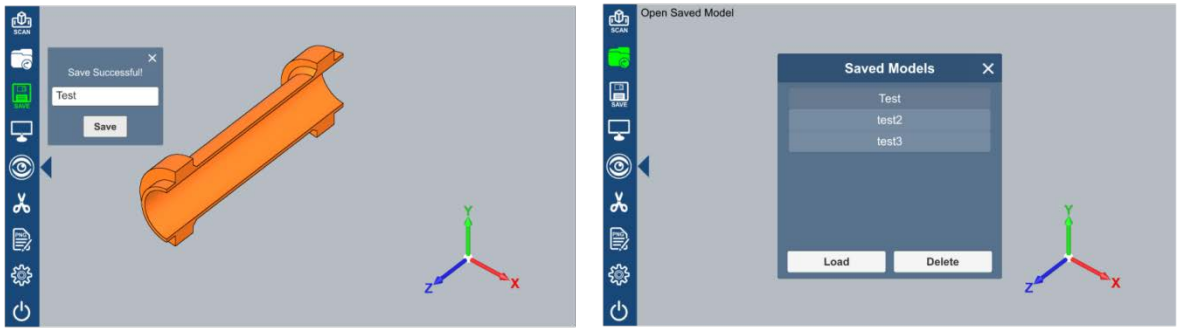
```
BinaryFormatter bf = new BinaryFormatter ();

FileStream file = File.Create (Application.persistentDataPath + "/"+modelName);

ObjectMeshData data = new ObjectMeshData ();
data.vertices = objectMesh.vertices;
data.normals = objectMesh.normals;
data.triangles = objectMesh.triangles;
data.objectName = objectInCenter.name;

bf.Serialize (file, data);
```

Πίνακας 7.4: Κώδικας αποθήκευσης μοντέλου σε αρχείο

Στο παραπάνω τμήμα κώδικα, η κλάση **ObjectMeshData** περιέχει μόνο τις μεταβλητές που απαιτούνται για την αποθήκευση του μοντέλου. Σημαντικό για την επιτυχή σειριοποίηση είναι η κλάση αυτή να έχει οριστεί ως **Serializable** κατά τη δήλωση της.

Μέσω της επιλογής ανάκτησης (Open) στο κύριο μενού της εφαρμογής, ο χρήστης μπορεί να επιλέξει ένα μοντέλο που έχει αποθηκεύσει και στη συνέχεια να το ανοίξει για περαιτέρω επεξεργασία. Η ανάκτηση μοντέλων ακολουθεί την ακριβώς αντίστροφη διαδικασία. Δηλαδή, αφού διαβαστούν τα δεδομένα του αρχείου εφαρμόζεται η διαδικασία της αποσειριοποίησης (**Deserialization**) προκειμένου να ανακτηθούν τα δεδομένα γεωμετρίας του μοντέλου. Στη συνέχεια είτε δημιουργείται ένα νέο μοντέλο με την γεωμετρία που διαβάστηκε, είτε αντικαθίσταται η γεωμετρία του μοντέλου που ήδη βρίσκεται στη σκηνή, με τη νέα γεωμετρία.

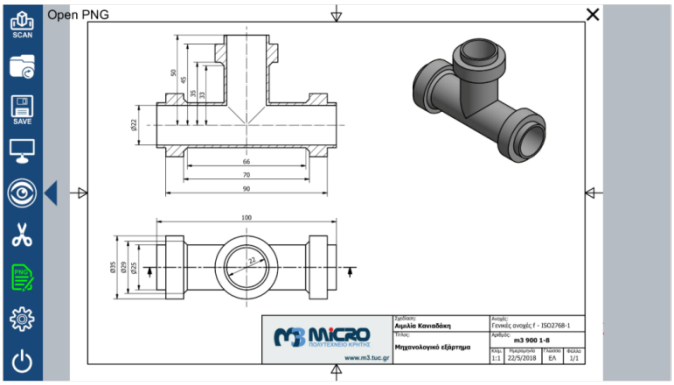
Τέλος υπάρχει και η επιλογή διαγραφής ενός αποθηκευμένου αρχείου, ανοίγοντας το παράθυρο αποθηκευμένων μοντέλων και επιλογής της λειτουργίας διαγραφής. Η διαδικασία αποθήκευσης και ανάκτησης ενός μοντέλου παρουσιάζονται στο σχήμα 7.11.



Σχήμα 7.11: Αποθήκευση και άνοιγμα μοντέλου

7.6 Άνοιγμα μηχανολογικού σχεδίου

Η εφαρμογή προσφέρει επιπλέον της δυνατότητα εμφάνισης του Μηχανολογικού Σχεδίου του αντικειμένου σε μορφή αρχείου png. Αυτό γίνεται από την αντίστοιχη επιλογή στο κύριο μενού της εφαρμογής. Για εκπαιδευτικούς λόγους, τα μηχανολογικά σχέδια δεν είναι διαθέσιμα για όλα τα αντικείμενα. Όπως και τα τρισδιάστατα μοντέλα, έτσι και τα αρχεία των μηχανολογικών σχεδίων αποθηκεύονται online για λόγους εξοικονόμησης χώρου στη φορητή συσκευή. Έτσι για την προβολή του μηχανολογικού σχεδίου, γίνεται λήψη του αντίστοιχου αρχείου εφόσον αυτό υπάρχει και στη συνέχεια προβολή αυτού στο κέντρο της συσκευής. Για την προβολή του αρχείου δεν χρειάζεται ειδική διαδικασία εισαγωγής του στο περιβάλλον της Unity. Ως αρχείο εικόνας, τα δεδομένα του μπορούν απευθείας να φορτωθούν σε ένα αντικείμενο τύπου **Sprite** στο UI της Unity. Στο σχήμα 7.12 παρουσιάζεται ένα στιγμιότυπο προβολής ενός μηχανολογικού σχεδίου εντός της εφαρμογής.



Σχήμα 7.12: Άνοιγμα μηχανολογικού σχεδίου

7.7 Επιλογές (Main Options)

Για λόγους μεγαλύτερης ευελιξίας, ο χρήστης έχει τη δυνατότητα να προσδιορίσει τις παραμέτρους ευαισθησίας για τις λειτουργίες μεγέθυνσης, περιστροφής και μετακίνησης της κάμερας. Η αλλαγή των παραμέτρων αυτών καθίσταται ιδιαίτερα σημαντική κατά τη διάρκεια της εφαρμογής και διευκολύνει τη λειτουργία της. Η αλλαγή μίας τέτοιας παραμέτρου αποθηκεύεται ακόμα και εάν ο χρήστης τερματίσει την εφαρμογή.

Η αποθήκευση μίας προτίμησης χρήστη υλοποιείται με τη χρήση της κλάσης **PlayerPrefs** που παρέχει η Unity. Η κλάση αυτή αντιστοιχεί ένα string με μία μεταβλητή και αποθηκεύει την τιμή της στο φάκελο δεδομένων της εφαρμογής. Χρησιμοποιώντας το όνομα του string η μεταβλητή μπορεί να ανακτηθεί οποιαδήποτε στιγμή. Επομένως κάθε παράμετρος ευαισθησίας που μπορεί να μεταβάλλει ο χρήστης αποθηκεύεται με αυτόν τον τρόπο. Για τη μεταβολή των παραπάνω παραμέτρων, ο χρήστης χρησιμοποιεί το μενού επιλογών το

οποίο επιτρέπει την αλλαγή των τιμών τους σε γραφικό περιβάλλον με τη χρήση κάποιων **Slider**. Επιπλέον προσφέρεται η δυνατότητα αλλαγής χρώματος του background της εφαρμογής, η αλλαγή της λειτουργίας θέασης της κάμερας από perspective σε orthographic ή αντίστροφα, καθώς και η μεταβολή της ευαισθησίας βάθους των ακμών του μοντέλου. Στους πίνακες 7.5 και 7.6 παρουσιάζονται παραδείγματα αποθήκευσης μίας παραμέτρου μετά την αλλαγή της και στη συνέχεια ανάκτηση αυτής.

Αποθήκευση προτίμησης χρήστη

```
public void    UpdateZoomSensitivity(){
    zoomSensitivity = zoomSensitivitySlider.value;
    zoomSliderValue.text = Math.Round (zoomSensitivity, 2).ToString();
    PlayerPrefs.SetFloat ("ZoomSensitivity", zoomSensitivity);
}
```

Πίνακας 7.5: Κώδικας αποθήκευσης προτίμησης χρήστη

Ανάκτηση προτίμησης χρήστη

```
zoomSensitivity = zoomSensitivitySlider.value = PlayerPrefs.GetFloat ("ZoomSensitivity",1);
zoomSliderValue.text = Math.Round (zoomSensitivity, 2).ToString();
```

Πίνακας 7.6: Κώδικας ανάκτησης προτίμησης χρήστη

Στο σχήμα 7.13 παρουσιάζεται το μενού επιλογών που προσφέρει η εφαρμογή.

Σχήμα 7.13: Μενού επιλογών

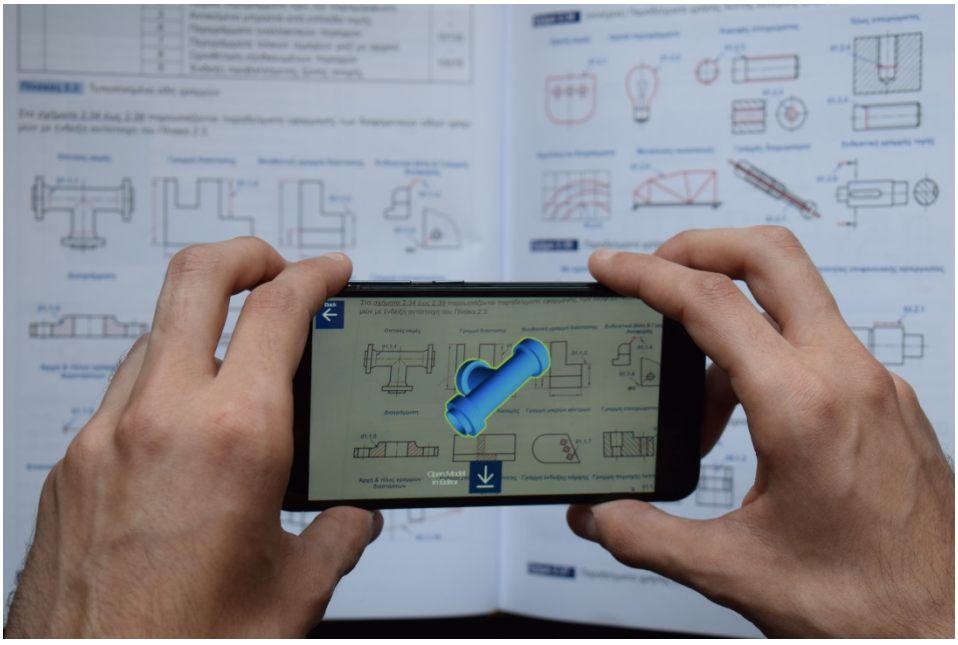
7.8 Μενού εφαρμογής

Οι λειτουργίες που αναφέρθηκαν στις προηγούμενες ενότητες, προσφέρονται μέσω του κύριου περιβάλλοντος της εφαρμογής. Για διευκόλυνση του χρήστη πριν την πρόσβαση του στο κύριο περιβάλλον, δημιουργήθηκε επιπλέον ένα μενού το οποίο εμφανίζεται κατά την έναρξη της εφαρμογής. Το μενού προσφέρει δύο μόνο λειτουργίες. Είτε το άνοιγμα (Open) κάποιου ήδη υπάρχοντος μοντέλου όπως αυτό περιγράφεται στην ενότητα 7.4, είτε την εισαγωγή ενός νέου μοντέλου μέσω σάρωσης αυτού στο περιβάλλον επαυξημένης πραγματικότητας (**Scan**). Τέλος ο χρήστης μπορεί να τερματίσει οποιαδήποτε στιγμή την εφαρμογή μέσω της επιλογής Exit, που βρίσκεται στο μενού καθώς και στο κύριο περιβάλλον της εφαρμογής. Στο σχήμα 7.14 παρουσιάζεται το μενού αυτό.

Σχήμα 7.14: Το μενού – αρχική οθόνη της εφαρμογής

7.9 Εισαγωγή μοντέλου μέσω επαυξημένης πραγματικότητας

Όπως έχει ήδη αναφερθεί, η εισαγωγή κάποιου μοντέλου στο περιβάλλον επεξεργασίας που περιγράφηκε σε αυτό το κεφάλαιο, γίνεται μέσω επαυξημένης πραγματικότητας και συγκεκριμένα μέσω της επιλογής Scan. Μετά την εύρεση μίας εικόνας - στόχου στον πραγματικό κόσμο που αντιστοιχεί σε ένα τρισδιάστατο μοντέλο, ο χρήστης επιλέγει το μοντέλο που προβάλλεται στην κάμερα της συσκευής και στη συνέχεια έχει την επιλογή είτε να το εισάγει στο περιβάλλον επεξεργασίας, είτε να αναιρέσει την επιλογή του και να προχωρήσει στην εύρεση μίας διαφορετικής εικόνας στόχου. Ένα στιγμιότυπο όπου φαίνεται η επιλογή ενός μοντέλου παρουσιάζεται στο σχήμα 7.15.



Σχήμα 7.15: Επιλογή ενός μοντέλου από το χρήστη μέσω επαυξημένης πραγματικότητας

7.10 Αξιολογήσεις Χρηστών

Το τελευταίο στάδιο για την επαλήθευση της καλής λειτουργίας της εφαρμογής ήταν η αξιολόγηση του περιβάλλοντος και των δυνατοτήτων που προσφέρει, από διαφορετικούς χρήστες. Οι παρατηρήσεις τους συνέβαλαν στην τελική μορφοποίηση της εφαρμογής, ώστε να έχει ένα πιο εύχρηστο περιβάλλον επεξεργασίας. Οι πιο σημαντικές αλλαγές μετά από αξιολογήσεις χρηστών αναφέρονται στη συνέχεια.

Αρχικά μία σημαντική διαφοροποίηση ήταν η επιλογή εμφάνισης του μηχανολογικού σχεδίου του μοντέλου. Η επιλογή αυτή εμφάνιζε το μηχανολογικό σχέδιο σε μορφή pdf, το οποίο καθιστούσε απαραίτητη την ύπαρξη μίας εξωτερικής εφαρμογής στη συσκευή για την ανάγνωση αρχείων pdf. Μία τέτοια εφαρμογή δεν ήταν πάντα διαθέσιμη σε όλες τις συσκευές. Συνεπώς η εμφάνιση του μηχανολογικού σχεδίου ενσωματώθηκε σε μορφή png στο περιβάλλον της εφαρμογής, έτσι ώστε να είναι δυνατή η ανάγνωση της από κάθε συσκευή.

Στην αρχή σχεδίασης της εφαρμογής και συγκεκριμένα κατά την έναρξη αυτής, εμφανιζόταν απευθείας το κύριο περιβάλλον που περιγράφηκε στο κεφάλαιο 7. Το κύριο περιβάλλον αρχικά εμφανιζόταν χωρίς την ύπαρξη κάποιου μοντέλου σε αυτό. Το γεγονός αυτό συχνά δημιουργούσε δυσκολίες κατατόπισης του χρήστη, καθώς δεν ήταν προφανές ότι η εισαγωγή μοντέλου γίνεται από την επιλογή scan, ενώ ταυτόχρονα η επιλογή ανάκτησης μοντέλου περιείχε αποθηκευμένα μοντέλα. Για το λόγο αυτό αποφασίστηκε η δημιουργία ενός κεντρικού μενού, όπως αυτό περιγράφεται στην ενότητα 7.8, το οποίο εμφανίζεται κατά την έναρξη της εφαρμογής και προσφέρει στο χρήστη τις δύο δυνατές λειτουργίες εισαγωγής μοντέλου στο κύριο περιβάλλον.

Όσον αφορά τη λειτουργία τομής και συγκεκριμένα τα επίπεδα τομής που προσφέρονται, αυτά είχαν σχεδιαστεί με διαφορετικό τρόπο. Το χρώμα κάθε επιπέδου δεν είχε κάποια διαφάνεια, με αποτέλεσμα κατά τη μετακίνησή του να καθιστά αόρατο ένα τμήμα του μοντέλου. Αυτό είχε ως συνέπεια η τομή που επιθυμούσε ο χρήστης να γινόταν πολλές φορές σε διαφορετικό σημείο, λόγω έλλειψης ορατότητας ολόκληρου του μοντέλου. Για το λόγο αυτό τα επίπεδα τομής έγιναν διαφανή, προσφέροντας έτσι μεγαλύτερη ακρίβεια στο χρήστη. Επιπλέον για τον προσδιορισμό της κατεύθυνσης του επιπέδου τομής, είχε χρησιμοποιηθεί μόνο ένα βέλος στο κέντρο του επιπέδου. Έτσι, κατά την μετακίνηση του επιπέδου τομής, το βέλος αυτό συχνά επικαλυπτόταν από το μοντέλο με αποτέλεσμα ο χρήστης να μη γνωρίζει την κατεύθυνση τομής. Έτσι η παρουσίαση αυτή βελτιώθηκε και αποφασίστηκε να τοποθετηθούν τέσσερα βέλη προσδιορισμού κατεύθυνσης, ένα σε κάθε άκρη του επιπέδου (βλ. σχήμα 7.4).

8. ΣΥΝΟΨΗ ΚΑΙ ΜΕΛΛΟΝΤΙΚΕΣ ΕΠΕΚΤΑΣΕΙΣ

Στην παρούσα εργασία υλοποιήθηκε μία εφαρμογή για φορητές συσκευές που δίνει τη δυνατότητα εφαρμογής γεωμετρικών μετασχηματισμών σε μοντέλα στερεών. Με τη χρήση της επαυξημένης πραγματικότητας γίνεται εισαγωγή των μοντέλων στο περιβάλλον μετασχηματισμού μετά από εύρεση της εικόνας-στόχου του επιθυμητού μοντέλου. Η εισαγωγή μοντέλων με αυτόν τον τρόπο απαιτεί σύνδεση της συσκευής στο διαδίκτυο, καθώς τα μοντέλα αποθηκεύονται online για λόγους εξοικονόμησης χώρου της συσκευής. Οι βασικοί μετασχηματισμοί αφορούν την περιστροφή, μεγέθυνση και μετακίνηση του μοντέλου. Μεγαλύτερη έμφαση δόθηκε στην υλοποίηση της τομής ενός μοντέλου σε σχέση με ένα επίπεδο, το οποίο μετακινεί ο χρήστης στην επιθυμητή θέση τομής σε έναν από τους άξονες x, y και z του μοντέλου.

Για λόγους καλύτερης απόδοσης και ορισμένων περιορισμών των αλγορίθμων τομής, οι βασικοί μετασχηματισμοί υλοποιήθηκαν με κατάλληλη διαχείριση της κάμερας του περιβάλλοντος ανάπτυξης της εφαρμογής και όχι με αντίστοιχη διαχείριση του μοντέλου. Επιπλέον λειτουργίες που υλοποιήθηκαν αφορούν την αποθήκευση του μοντέλου, τον υπολογισμό της επιφάνειας, όγκου και βάρους του, καθώς και την προβολή του μηχανολογικού σχεδίου του μοντέλου.

Για τη δημιουργία μίας εφαρμογής που χρησιμοποιεί λειτουργίες επαυξημένης πραγματικότητας, θα πρέπει να επιλεχθεί μία βιβλιοθήκη επαυξημένης πραγματικότητας (**AR SDK**) με δυνατότητα ενσωμάτωσης στο περιβάλλον της Unity. Επειδή η Unity έχει γίνει ιδιαίτερα γνωστή τα τελευταία χρόνια, οι περισσότερες γνωστές βιβλιοθήκες επαυξημένης πραγματικότητας προσφέρουν τη δυνατότητα ενσωμάτωσης στο περιβάλλον της Unity. Μερικές από τις πιο γνωστές βιβλιοθήκες είναι οι **Vuforia**, **Wikitude**, **Kudan**, **EasyAR** και **ARCore**. Οι βιβλιοθήκες αυτές προσφέρουν απλές και προχωρημένες λειτουργίες. Η εφαρμογή της παρούσας εργασίας χρειάστηκε μόνο απλή αναγνώριση εικόνων-στόχων, λειτουργία που υποστηρίζεται από όλες τις βιβλιοθήκες. Η βασική παράμετρος επιλογής της βιβλιοθήκης ήταν η άδεια χρήσης της, μια και η εφαρμογή προορίστηκε για κοινοποίηση στο διαδίκτυο. Έτσι, επιλέχθηκε η βιβλιοθήκη EasyAR, η οποία είναι η μόνη βιβλιοθήκη που υποστηρίζει κοινοποίηση στο διαδίκτυο ακόμα και στην απλή έκδοσή της. Σημαντική διευκόλυνση για την εφαρμογή θα ήταν η αποθήκευση των εικόνων-στόχων (Targets) σε κάποιο cloud για περαιτέρω εξοικονόμηση χώρου στη φορητή συσκευή. Η λειτουργία αυτή διατίθεται μόνο σε επαγγελματικά πακέτα των βιβλιοθηκών επαυξημένης πραγματικότητας, οπότε και δεν μπόρεσε να χρησιμοποιηθεί.

Μελλοντικές επεκτάσεις θα μπορούσαν να είναι η εύρεση χαρακτηριστικών σημείων και αξόνων των μοντέλων, ώστε η επιλογή των επιπέδων τομής να γίνεται σε χαρακτηριστικές θέσεις τους. Επιπλέον η εφαρμογή μπορεί να επεξεργάζεται συναρμολογημένες διατάξεις και όχι απλά στερεά αντικείμενα, προσφέροντας εκτός των ήδη περιγραφόμενων λειτουργιών επιπλέον λειτουργίες όπως animation της συναρμολόγησης, βίντεο και γενικά πλήρη επεξεργασία συναρμολογήσεων.

9. ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Computer-aided Design https://en.wikipedia.org/wiki/Computer-aided_design

[2] Αντωνιάδης Α: Μηχανουργική Τεχνολογία, 3η έκδοση 2017, Εκδόσεις Τζιόλα

[3] Computational Geometry: https://en.wikipedia.org/wiki/Computational_geometry

[4] Τζίμας Ε. (2016). Υποστήριξη Διαδικασιών Προετοιμασίας Εργαλειομηχανών με τη βοήθεια Επαυξημένης Πραγματικότητας

[5] Κόκκας Α. (2018). Χωροταξική Διάταξη και Προσομοίωση Λειτουργίας Ευέλικτου Συστήματος Κατεργασιών με Χρήση Επαυξημένης Πραγματικότητας

[6] de Berg, M., Cheong, O., van Kreveld, M., and Overmars, M. (2008). Computational Geometry: Algorithms and Applications. Springer, 3rd edition.

[7] Bajaj, C. L. and Dey, T. K. (1990). Polygon nesting and robustness.

[8] Scherzinger A., Brix T. and Hinrichs K. (2017). An Efficient Geometric Algorithm for Clipping and Capping Solid Triangle Meshes.

[9] Elberly D. (2002). Clipping a Mesh Against a Plane

[10] Obj Importer: http://wiki.unity3d.com/index.php/ObjImporter

[11] Unity Documentation: https://docs.unity3d.com

[12] Silhouette-Outlined Diffuse Shader: http://wiki.unity3d.com/index.php/Silhouette-Outlined_Diffuse

ΠΑΡΑΡΤΗΜΑ

Στο παράρτημα αυτό, παρατίθεται όλα τα Scripts που δημιουργήθηκαν στην εφαρμογή. Κάθε script αποτελεί και μία νέα κλάση.

Doubly Connected Edge List

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;

// Doubly Connected Edge List Class
public class DCEL {

    private List<Vertex> vertices;
    private List<Vector3> vertex3Dpositions;
    private List<Face> faces;
    private List<HalfEdge> halfEdges;
    private Face unboundedFace;

    public DCEL(){
        vertices = new List<Vertex> ();
        vertex3Dpositions = new List<Vector3> ();
        faces = new List<Face> ();
        halfEdges = new List<HalfEdge> ();
        unboundedFace = new Face ();
    }

    public void SortVertices(){
        VertexComparer vComparer = new VertexComparer ();

        vertices.Sort (vComparer);

    }

    public void Clear(){

        vertices.Clear ();
        faces.Clear ();
        halfEdges.Clear ();
        unboundedFace = null;
    }

    public void AddHalfEdge(HalfEdge e){
        halfEdges.Add (e);
    }

    public void AddVertex(Vertex v){
        vertices.Add (v);
    }

    public void AddVertex3Dposition(Vector3 v){
        vertex3Dpositions.Add (v);
    }

    public void AddFace(Face f){
        faces.Add (f);
    }

    public Vertex GetVertex(int position){
        return vertices[position];
    }

    public Vector3 GetVertex3dPosition(int position){
        return vertex3Dpositions[position];
    }

    public HalfEdge GetHalfEdge(int position){
        return halfEdges[position];
    }

    public Face GetFace(int position){
        return faces[position];
    }

    public List<Vertex> GetVertices(){
        return vertices;
    }

    public List<Face> GetFaces(){
        return faces;
    }

    public List<HalfEdge> GetHalfEdges(){
        return halfEdges;
    }

    public Face GetUnboundedFace(){
        return unboundedFace;
    }


    // Comparer that sorts Vertex objects in descending order according t
o their y and z values
    internal class VertexComparer : IComparer<Vertex>
    {
        public int Compare (Vertex ver1, Vertex ver2)
        {
            Vector2 v1 = ver1.GetPosition ();
            Vector2 v2 = ver2.GetPosition ();

            if (Math.Round(v1.y,5) <Math.Round( v2.y,5) || (Mathf.Approxi
mately (v1.y, v2.y) && Math.Round(v1.x,5) > Math.Round(v2.x,5))) {
                return 1;
            } else if (Math.Round(v1.y,5) >Math.Round( v2.y,5) || (Mathf.
Approximately (v1.y, v2.y) && Math.Round(v1.x,5) < Math.Round(v2.x,5))) {
                return -1;
            } else {
                return 0;
            }
        }
    }
}
```

Face

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Face  {

    private HalfEdge outerComponent;
    private List<HalfEdge> innerComponents;

    public Face(){
        outerComponent = null;
        innerComponents = new List<HalfEdge> ();
    }

    public void AddInnerComponent(HalfEdge innerComponent){
        innerComponents.Add (innerComponent);
    }

    public HalfEdge GetOuterComponent(){
        return outerComponent;
    }

    public List<HalfEdge> GetInnerComponents(){
        return innerComponents;
    }

    public void SetOuterComponent(HalfEdge outerComponent){
        this.outerComponent = outerComponent;
    }
}
```

HalfEdge

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class HalfEdge  {

    private Vertex origin;
    private HalfEdge twin;
    private Face incidentFace;
    private HalfEdge prev;
    private HalfEdge next;
    private Vertex helperVertex;    // Helper Vertex of this edge
    bool ishelperMerge;        // True if helper Vertex is a Merge Vertex
    bool hasBeenTraversed;    // True if HalfEdge has been traversed duri
ng the seperation in monotoneParts phase

    public HalfEdge(){
        origin = null;
        twin = null;
        incidentFace = null;
        next = null;
        prev = null;
        helperVertex = null;
        ishelperMerge = false;
        hasBeenTraversed = false;
    }

    public Vertex GetOrigin(){
        return origin;
    }

    public Vertex GetHelper(){
        return helperVertex;
    }

    public HalfEdge GetTwin(){
        return twin;
    }

    public Face GetIncidentFace(){
        return incidentFace;
    }

    public HalfEdge GetPrev(){
        return prev;
    }

    public HalfEdge GetNext(){
        return next;
    }

    public bool IsHelperMerge(){
        return ishelperMerge;
    }

    public bool HasBeenTraversed(){
        return hasBeenTraversed;
    }


    public void SetHasBeenTraversed(bool value){
        this.hasBeenTraversed = value;
    }


    public void SetIsHelperMerge(bool value){
        this.ishelperMerge = value;
    }


    public void SetOrigin(Vertex origin){
```

```
        this.origin = origin;
    }

    public void SetHelper(Vertex helperVertex){
        this.helperVertex = helperVertex;
    }

    public void SetTwin(HalfEdge twin){
        this.twin = twin;
    }

    public void SetIncidentFace(Face incidentFace){
        this.incidentFace = incidentFace;
    }

    public void SetPrev(HalfEdge prev){
        this.prev = prev;
    }

    public void SetNext(HalfEdge next){
        this.next = next;
    }
}
```

Vertex

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Vertex  {

    private HalfEdge incidentEdge;
    private Vector2 position;
    pri-
vate int initialPositionInList; // Variable holding the position the vert
ex was stored in the list of vertices in the DCEL
    pri-
vate bool isOnRightChain;    // Variable indicating if the Vertex lies on
 the right chain of a y-monotone polygon
    pri-
vate int listIndex;        // Variable holding the position of the vertex
 in the list of Vertices in the triangulation phase. If the Vertex is not
 in the list it is set to -1.
    private int vertexType;     // Indicates the type of this Vertex (0-
>Regular Vertex, 1-> Start Vertex, 2-> End Vertex, 3-> Split Vertex, 4-
> Merge Vertex)

    public Vertex(){
        incidentEdge = null;
        position = Vector2.positiveInfinity;
        listIndex = -1;
    }

    public Vertex(Vector2 position){
        incidentEdge = null;
        this.position = position;
        listIndex = -1;
    }

    public bool IsOnRightChain(){
        return isOnRightChain;
    }


    public Vector2 GetPosition(){
        return position;
    }

    public HalfEdge GetIncidentEdge(){
        return incidentEdge;
    }

    public int GetInitialPositionInList(){
        return initialPositionInList;
    }

    public int GetListIndex(){
        return listIndex;
    }

    public int GetVertexType(){
        return vertexType;
    }

    public void SetVertexType(int vertexType){
        this.vertexType = vertexType;
    }

    public void SetIsOnRightChain(bool value){
        this.isOnRightChain = value;
    }

    public void SetPosition(Vector2 position){
        this.position = position;
    }

    public void SetInitialPositionInList(int initialPositionInList){
        this.initialPositionInList = initialPositionInList;
    }

    public void SetListIndex(int listIndex){
        this.listIndex = listIndex;
    }

    public void SetIncidentEdge(HalfEdge incidentEdge){
        this.incidentEdge = incidentEdge;
    }
}
```

IntersectionEdge

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class IntersectionEdge {

    Vector3 startVertex;
    Vector3 endVertex;

    public IntersectionEdge(Vector3 startV, Vector3 endV){
        this.startVertex=startV;
        this.endVertex = endV;

    }

    public Vector3 GetStartVertex(){
        return startVertex;
    }

    public Vector3 GetEndVertex(){
        return endVertex;
    }

    public void SetStartVertex(Vector3 startVertex){
        this.startVertex = startVertex;
    }

    public void SetEndVertex(Vector3 endVertex){
        this.endVertex = endVertex;
    }
}
```

Loop

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Loop {

    List<IntersectionEdge> edges;
    List<Vector3> vertices;
    bool isTriangulated;        // Variable indicating that this loop has
 already been used in the triangulation algorithm

    public void AddEdge(IntersectionEdge e){
        edges.Add(e);
        vertices.Add (e.GetStartVertex ());
    }

    public void AddVertex(Vector3 v){
        vertices.Add(v);
    }

    public Loop(){
        edges = new List<IntersectionEdge> ();
        vertices = new List<Vector3> ();
        isTriangulated = false;
    }


    public void SetIsTriangulated(bool isTriangulated){
        this.isTriangulated = isTriangulated;
    }

    public IntersectionEdge GetFirstEdge(){
        return edges [0];

    }

    public List<IntersectionEdge> GetEdges(){
        return edges;
    }

    public List<Vector3> GetVertices(){
        return vertices;
    }

    public bool IsTriangulated(){
        return isTriangulated;

    }

    public Vector3 GetVertex(int position){
        return vertices[position];
    }

    public IntersectionEdge GetEdge(int position){

        return edges [position];
    }
}
```

ImageTargetController

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine.Networking;
using UnityEngine;
using UnityEngine.UI;
using System.IO;

public class ImageTargetController : MonoBehaviour {

    bool wasEnabled= false;
    bool objectCreated= false;
    public GameController gameControl;

    void OnEnable(){
        wasEnabled = true;
    }

    void Update(){
        if (wasEnabled && !objectCreated && gameControl.CheckInternetAvai
lability() )         {
            gameCon-
trol.DownloadRequest("http://www.antoniadis.gr/models/"+this.name+".obj",
this.gameObject,this.name);
            objectCreated = true;
            Destroy (this);

        }
```

```
    }
}
```

MenuController

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.EventSystems;
using UnityEngine.SceneManagement;
using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;

public class MenuController : MonoBehaviour {

    public GameObject openPanel;
    public GameObject savedModelsPanel;
    public GameObject modelButtonPrefab;
    public GameObject confirmPanel;

    public Material objectMaterial;

    private string selectedModelName;

    public void ScanModel(){

        StartCoroutine (LoadARScene ());
    }

    IEnumerator LoadARScene()
    {
        AsyncOperation asyncLoad = SceneManager.LoadSceneAsync (2,
LoadSceneMode.Single);

        while (!asyncLoad.isDone)
        {
            yield return null;
        }
    }

    public void OpenSavedModelList(){

        openPanel.SetActive (true);

        DirectoryInfo dInfo = new DirectoryInfo (Application.
persistentDataPath);
        FileInfo[] fInfo = dInfo.GetFiles ();

        // Create as many buttons as the saved models
        for (int i = 0; i < fInfo.Length; i++){

            // ignore png files as they dont correspond to saved models
            if (fInfo [i].Name.Length>=4 && fInfo [i].Name.Substring (
fInfo [i].Name.Length - 4) == ".png") {
                continue;
            }

            GameObject modelButton = Instantiate (modelButtonPrefab,
savedModelsPanel.GetComponent<RectTransform> ());

            Button btn = modelButton.GetComponent<Button> ();
            btn.onClick.AddListener (this.SelectSavedModel);
            //set name
            modelBut-
ton.GetComponentInChildren<Text> ().text = fInfo[i].Name;

        }
    }

    public void SelectSavedModel(){

        GameObject selectedGO = EventSystem.current.
currentSelectedGameObject;
        selectedModelName = selectedGO.GetComponentInChildren<Text> ().
text;

    }

    public void CloseSavedModelList(){
        foreach (Transform child in savedModelsPanel.transform) {
            GameObject.Destroy(child.gameObject);
        }
        openPanel.SetActive (false);
    }

    public void Load(){

        if (File.Exists (Application.persistentDataPath + "/"+
selectedModelName)) {
            BinaryFormatter bf = new BinaryFormatter ();

            SurrogateSelector ss = new SurrogateSelector ();
            Vec-
tor3SerializationSurrogate v3ss = new Vector3SerializationSurrogate ();
            ss.AddSurrogate (typeof(Vector3), new StreamingContext (
StreamingContextStates.All), v3ss);
            bf.SurrogateSelector = ss;

            FileStream file = File.Open (Application.persistentDataPath +
 "/"+selectedModelName, FileMode.Open);

            ObjectMeshData data = (ObjectMeshData)bf.Deserialize (file);

            file.Close ();

            GameOb-
ject objectInCenter = new GameObject ("ObjectInCenter");
            objectInCenter.tag = "ScreenCenter";
            Material mat = objectInCenter.AddComponent<MeshRenderer> ().
material = objectMaterial;
            mat.shader = Shader.Find ("Standard");

            MeshFilter mf = objectInCenter.AddComponent<MeshFilter> ();
            Mesh objectMesh = mf.mesh;

            objectMesh.vertices = data.vertices;
            objectMesh.normals = data.normals;
            objectMesh.triangles = data.triangles;

            objectInCenter.name = data.objectName;

            StartCoroutine(LoadEditorAsync(objectInCenter));

            }
    }


    IEnumerator LoadEditorAsync(GameObject objectToMove)
    {
        AsyncOperation asyncLoad = SceneManager.LoadSceneAsync (1,
LoadSceneMode.Additive);

        while (!asyncLoad.isDone)
        {
            yield return null;
        }

        //objectToMove.SetActive (true);
        SceneManager.MoveGameObjectToScene (objectToMove, SceneManager.
GetSceneByBuildIndex (1));
        SceneManager.UnloadSceneAsync(SceneManager.GetActiveScene());


    }

    public void DeleteSavedModel(){

        confirmPanel.SetActive (true);

    }

    public void RejectDeletion(){

        confirmPanel.SetActive (false);
    }

    public void ConfirmDeletion(){

        if (File.Exists (Application.persistentDataPath + "/" +
selectedModelName)) {

            File.Delete (Application.persistentDataPath + "/" +
selectedModelName);

            // Update Model Panel after deletion
            foreach (Transform child in savedModelsPanel.transform) {
                GameObject.Destroy(child.gameObject);
            }
            DirectoryInfo dInfo = new DirectoryInfo (Application.
persistentDataPath);
            FileInfo[] fInfo = dInfo.GetFiles ();

            // Create as many buttons as the saved models
            for (int i = 0; i < fInfo.Length; i++){

                // ignore png files as they dont correspond to saved mode
ls
                if (fInfo [i].Name.Length>=4 && fInfo [i].Name.
Substring (fInfo [i].Name.Length - 4) == ".png") {
                    continue;
                }

                GameOb-
ject modelButton = Instantiate (modelButtonPrefab, savedModelsPanel.GetCo
mponent<RectTransform> ());
                Button btn = modelButton.GetComponent<Button> ();
                btn.onClick.AddListener (this.SelectSavedModel);
                //set name
                modelButton.GetComponentInChildren<Text> ().text =
fInfo[i].Name;

            }

        }

        confirmPanel.SetActive (false);

    }

    public void ConfirmExit(){
        Application.Quit ();
    }

}
```

GameController (Augmented Reality Controller)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.Networking;
using UnityEngine.UI;
using System.IO;

public class GameController : MonoBehaviour {

    public GameObject openEditorButton;
    GameObject arObject=null;
    bool canRaycast = true;
    public Material objectMaterial;
    public RadialProgressBar progressBar;
    public Text notificationText;

    private Transform oldParent;

    void Update () {

        // On mouse Click or Touch cast a Ray and check if it intersects
with a projected 3D Object
        if (Input.GetMouseButtonDown(0)) {

            RaycastHit hit;

            Ray ray = Camera.main.ScreenPointToRay (Input.mousePosition);
```

```
            Material mat;
            Color outlineColor;

            // If Raycast is possible (Not touching a UI Element)
            if (!canRaycast) {
                return;
            }

            if (Physics.Raycast(ray, out hit)) {

                // Object Hit
                if (hit.transform.tag == "ARObject") {

                    openEditorButton.SetActive (true);

                    // reset outline of previous selected model if there
is any
                    if (arObject != null) {

                        // Reset parent to old Parent
                        arObject.transform.parent.parent = oldParent;
                        arObject.transform.parent.localPosition =
new Vector3 (0, 1, 0);
                        arObject.transform.parent.localEulerAngles =
new Vector3 (0,0, 0);

                        mat = arObject.GetComponent<Renderer> ().material
;
                        outlineColor = mat.GetColor ("_OutlineColor");
                        outlineColor.a = 0f;
                        mat.SetColor ("_OutlineColor", outlineColor);

                    }

                    //assign new selected Object
                    arObject = hit.transform.gameObject;

                    // Set  Main Camera as the pivot's parent.
                    oldParent = arObject.transform.parent.parent;
                    arObject.transform.parent.parent = Camera.main.
transform;
                    arObject.transform.parent.localPosition =
new Vector3 (0, 0.1f, 2f);
                    arObject.transform.parent.localEulerAngles =
new Vector3 (-40f,40f, 0);

                    //set Outline of selected object as visible
                    mat = arObject.GetComponent<Renderer> ().material;
                    outlineColor = mat.GetColor ("_OutlineColor");
                    outlineColor.a = 255f;
                    mat.SetColor ("_OutlineColor", outlineColor);
                }
            }
            else {
                // Nothing hit
                // reset outline of previous selected model
                if (arObject != null) {

                    // Reset parent to old Parent
                    arObject.transform.parent.parent = oldParent;
                    arObject.transform.parent.localPosition =
new Vector3 (0, 1, 0);
                    arObject.transform.parent.localEulerAngles =
new Vector3 (0,0, 0);

                    mat = arObject.GetComponent<Renderer> ().material;
                    outlineColor = mat.GetColor ("_OutlineColor");
                    outlineColor.a = 0f;
                    mat.SetColor ("_OutlineColor", outlineColor);

                }
                openEditorButton.SetActive (false);

            }
        }
    }

    // Functions used to enable or disable Raycasting depending on touch
position (UI elements should block raycasting)
    public void RaycastOff(){

        canRaycast = false;
    }

    public void RaycastOn(){

        canRaycast = true;
    }

    // Function that loads the selected the object to the editor scene
    public void OpenEditorScene(){

        string name = arObject.name;
        // Create a duplicate of the selected object that will be loaded
to the new scene
        GameObject objectDuplicate = Instantiate (arObject);
        objectDuplicate.name = name;

        // disable meshRenderer as duplicate should not be visible in the
 current scene
        objectDuplicate.GetComponent<MeshRenderer>().enabled=false;
        // Set properties
        objectDuplicate.transform.tag = "ScreenCenter";
        objectDuplicate.transform.localPosition = Vector3.zero;
        objectDuplicate.transform.localScale = Vector3.one;
        Destroy (objectDuplicate.GetComponent<Rotator> ());
        objectDuplicate.transform.localRotation = Quaternion.identity;

        Material mat = arObject.GetComponent<Renderer> ().material;
        mat.shader = Shader.Find ("Standard");
        mat.color = new Color32 (255, 102, 0, 255);

        StartCoroutine(LoadEditorAsync(objectDuplicate));

    }

    // Routine that loads the new scene asynchronously and moves the
duplicate object to that scene
    IEnumerator LoadEditorAsync(GameObject objectToMove)
    {
        AsyncOperation asyncLoad = SceneManager.LoadSceneAsync (1,
LoadSceneMode.Additive);

        while (!asyncLoad.isDone)
        {
            yield return null;
        }

        SceneManager.MoveGameObjectToScene (objectToMove, SceneManager.
GetSceneByBuildIndex (1));
        SceneManager.UnloadSceneAsync(SceneManager.GetActiveScene());


    }

    // Function that returns to editor scene
    public void BackToEditor(){

        StartCoroutine(LoadEditorAsync());
    }

    IEnumerator LoadEditorAsync()
    {
        AsyncOperation asyncLoad = SceneManager.LoadSceneAsync (1,
LoadSceneMode.Additive);

        while (!asyncLoad.isDone)
        {
            yield return null;
        }

        SceneManager.UnloadSceneAsync(SceneManager.GetActiveScene());

    }

    public void DownloadRequest(string url, GameObject parent,
string name){

        progressBar.Activate ();
        StartCoroutine(GetFile(url,parent,name));

    }

    // Routine that sends a download request to the specified url. If the
 re-
quest is successfull the object is downloaded. After proper modification
the object is projected to the current scene
    IEnumerator GetFile(string url, GameObject parent, string name )
    {

        // send request
        using (UnityWebRequest www = UnityWebRequest.Get(url))
        {
            UnityWebRequestAsyncOperation asyncOperation =
www.SendWebRequest ();
            float prevProgress = -1;
            while (!asyncOperation.isDone)    // wait until download is
complete. Track progress to the progress bar
            {
                if (asyncOperation.progress > prevProgress) {
                    prevProgress = asyncOperation.progress;
                    progressBar.UpdateProgress (asyncOperation.progress);
                }

                yield return null;
            }

            if (www.isNetworkError || www.isHttpError)
            {
                Debug.LogError(www.error);    // Print error message if
there is one
            }
            else
            {
                progressBar.UpdateProgress (asyncOperation.progress);

                // Create a file and copy downloaded information to it.
                File.Create (Application.persistentDataPath + name+
".obj").Close ();
                File.WriteAllText (Application.persistentDataPath + name+
".obj", www.downloadHandler.text);

                // Using the ObjImporter class import the object's mesh
and store it to a Mesh class instance
                Mesh importedMesh = new Mesh ();
                ObjImporter importer = new ObjImporter ();

                importedMesh = importer.ImportFile (Application.
persistentDataPath + name+".obj");

                // Create a new object
                GameObject targetModel = new GameObject (name);
                MeshFilter meshFilter = targetModel.AddComponent<
MeshFilter> ();
                MeshRenderer renderer =    targetModel.AddComponent<
MeshRenderer> ();

                //Set object's mesh as the imported mesh
                meshFilter.mesh = importedMesh;

                // Set material with an outline shader
                Material mat = targetModel.GetComponent<Renderer> ().
material;
                mat.shader = Shader.Find ("Outlined/Silhouetted Diffuse")
;
                mat.color = new Color32 (143, 192, 255, 255);
                mat.SetColor ("_OutlineColor", new Color32 (215, 255, 103
, 0));
                mat.SetFloat ("_Outline", 0.0005f);

                //Create a box collider for the object
                targetModel.AddComponent<BoxCollider> ();

                //Set tag of model
                targetModel.tag = "ARObject";

                //Create object pivot and set it to object center;
                GameObject pivot = new GameObject ("Pivot");

                pivot.transform.position = meshFilter.mesh.bounds.center;

                targetModel.transform.parent = pivot.transform;
```

```
                // Rescale object to 0.75 Unity units (by rescaling pivot
 object which is its parent);
                Vector3 boundsSize = meshFilter.mesh.bounds.size;
                float maxDimension = Mathf.Max (boundsSize.x, boundsSize.
y, boundsSize.z);
                float rescaleFactor = 0.75f / maxDimension;

                piv-
ot.transform.localScale = new Vector3 (rescaleFactor, rescaleFactor, resc
aleFactor);

                // Set pivot 1 unit above the image target;
                pivot.transform.localPosition = new Vector3(0,1,0);

                //Set parent of object pivot this image target
                pivot.transform.parent = parent.transform;

                pivot.AddComponent<Rotator> ();

                // Delete Created File in persistent Datapath
                File.Delete(Application.persistentDataPath + name+".obj")
;

            }
        }
    }

    // Function that checks if internet connection is available
    public bool CheckInternetAvailability(){

        if (Application.internetReachability == NetworkReachability.
NotReachable){
            notificationText.text = "Check Internet Connection!";
            return false;
        }
        else{
            notificationText.text = "";
            return true;
        }
    }
}
```

InterfaceController

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using System;
using UnityEngine.Networking;
using UnityEngine.SceneManagement;
using UnityEngine.EventSystems;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;
using System.Diagnostics;

public class InterfaceController : MonoBehaviour {

    public Text sliderTitle;
    public Text planePosition;

    public Text inputText;
    public Text inputTextSpecificWeight;

    public Text saveOutput;
    public Text notificationText;
    public Text leftPanelInfoText;
    public Text rightPanelInfoText;

    // Main Panels
    public GameObject mainPanel;
    public GameObject viewsPanel;
    public GameObject displayModePanel;
    public GameObject optionsPanel;
    public GameObject clipPanel;
    public GameObject exitPanel;

    // Color palette
    public GameObject colorPalette;
    public GameObject selectedColorIndicator;

    //Properties Panel and Model Weight Panel
    public GameObject propertiesPanel;
    public Text trianglesText;
    public Text verticesText;
    public Text surfaceText;
    public Text dimensionsText;
    public GameObject modelWeightPanel;
    public GameObject specificWeightInput;
    public Dropdown materialDropdown;
    public Text objectVolumeText;

    // Planes
    public GameObject planeX;
    public GameObject planeY;
    public GameObject planeZ;
    public GameObject planeZNormalsOff;
    public GameObject planeSliderObj;

    // Open function gameobjects
    public GameObject openPanel;
    public GameObject savedModelsPanel;
    public GameObject modelButtonPrefab;

    public GameObject confirmPanel;

    // Save Function
    public GameObject savePanel;

    public Material objectMaterial;

    public CameraController cameraControl;

    public Sprite leftArrow;
    public Sprite rightArrow;

    public Image mainPanelHideButton;

    public GameObject pdfImageGameObject;

    private Slider slider;

    private String selectedModelName;

    private Image lastSelectedButton;

    bool planeXinverted =false;
    bool planeYinverted =false;
    bool planeZinverted =false;
    int activePlane; // Indicates which plane is active for clipping.
0-> X, 1-> Y, 2-> Z

    private bool viewsPanelActive=false;
    private bool displayModePanelActive=false;
    private bool clipOperationActive=false;
    private bool mainPanelHidden=false;
    private bool colorPaletteActive=false;
    private bool openPanelActive=false;
    private bool savePanelActive=false;
    private bool optionsPanelActive=false;
    private bool propertiesPanelActive=false;
    private bool modelWeightPanelActive=false;
    private bool modelVolumePanelActive=false;
    private bool pngPanelActive = false;

    private bool canCalculateVolume=true;

    private int messagesCount=0;

    MeshClip clipMeshComponent;

    SwitchButtonImage switchButtonImageController;

    Calculator calculator;

    GameObject objectInCenter;

    float modelVolume;
    float specificWeight=2712f;    // Default specific weight for
Aluminium

    private Vector3[] modelVertices;
    private Vector3[] modelNormals;
    private int[] modelTriangles;

    // Use this for initialization
    void Start () {
        activePlane = -1;
        modelVolume = -1;
        clipMeshComponent = GetComponent<MeshClip> ();
        switchButtonImageController = GetComponent<SwitchButtonImage> ();
        calculator = GetComponent<Calculator> ();

        slider = planeSliderObj.GetComponent<Slider> ();

        objectIn-
Center = GameObject.FindGameObjectWithTag ("ScreenCenter");

        if (objectInCenter != null) {
            UnityEn-
gine.Debug.Log("Object Found! " + objectInCenter.name);
            objectInCenter.GetComponent<MeshRenderer>().enabled=true;

            Mesh objMesh = objectInCenter.GetComponent<MeshFilter> ().
mesh;
            modelVertices = objMesh.vertices;
            modelNormals = objMesh.normals;
            modelTriangles = objMesh.triangles;

            ReadjustPlanes ();

        }
    }

    public void OpenExitPanel(){
        exitPanel.SetActive (true);
        switchButtonImageController.SwitchExitButton ();
    }

    public void ConfirmExit(){
        Application.Quit ();
    }

    public void RejectExit(){
        exitPanel.SetActive (false);
        switchButtonImageController.SwitchExitButton ();
    }

    public void togglePNGPanel(){

        if (pngPanelActive) {
            leftPanelInfoText.text = "";
            pdfImageGameObject.SetActive (false);
            pngPanelActive = false;
        } else {
            leftPanelInfoText.text = "Open PNG";
            OpenPngFile ();
            pngPanelActive = true;
        }

    }

    public void ClosePNGPanel(){
        leftPanelInfoText.text = "";
        pdfImageGameObject.SetActive (false);
        pngPanelActive = false;
        switchButtonImageController.SwitchPNGImage();

    }

    public void OpenPngFile(){
        StartCoroutine (OpenPng ());

    }

    private Sprite LoadSprite(string path)
    {
        if (string.IsNullOrEmpty(path)) return null;
        if (System.IO.File.Exists(path))
```

```
        {
            byte[] bytes = System.IO.File.ReadAllBytes(path);
            Texture2D texture = new Texture2D(1, 1);
            texture.LoadImage(bytes);
            Sprite sprite = Sprite.Create(texture, new Rect(0, 0,
texture.width, texture.height), new Vector2(0.5f, 0.5f));
            return sprite;
        }
        return null;
    }

    IEnumerator OpenPng(){
        if (objectInCenter == null) {
            notificationText.text = "No Object is opened!";
            yield return new WaitForSeconds (3);
            ClosePNGPanel ();
            notificationText.text = "";
        }
        else if (File.Exists (Application.persistentDataPath + "/" +
objectInCenter.name +".png")) {

            Sprite mainSprite = LoadSprite (Application.
persistentDataPath + "/" + objectInCenter.name +".png");

            pdfImageGameOb-
ject.GetComponent<RectTransform> ().sizeDelta = new Vector2 (mainSprite.t
exture.width, mainSprite.texture.height);
            pdfImageGameObject.GetComponent<Image>().sprite = mainSprite;
            pdfImageGameObject.GetComponent<Image>().preserveAspect =
true;
            pdfImageGameObject.SetActive (true);

        }
        else if (Application.internetReachability == NetworkReachability.
NotReachable) {
            notificationText.text = "Check Internet Connection!";
            yield return new WaitForSeconds (3);
            ClosePNGPanel ();
            notificationText.text = "";

        } else {
            notificationText.text = "";

            UnityWebRequest www = UnityWebRequest.Get ("http://www.
antoniadis.gr/models/" + objectInCenter.name +".png");

            UnityWebRequestAsyncOperation asyncOperation =
www.SendWebRequest ();
            while (!asyncOperation.isDone) {
                yield return null;
            }

            if (www.isNetworkError || www.isHttpError) {
                UnityEngine.Debug.LogError (www.error);
            } else {

                File.Create (Application.persistentDataPath + "/" +
objectInCenter.name +".png").Close ();
                File.WriteAllBytes (Application.persistentDataPath + "/"
+ objectInCenter.name +".png", www.downloadHandler.data);

                Sprite mainSprite = LoadSprite (Application.
persistentDataPath + "/" + objectInCenter.name +".png");

                pdfImageGameObject.GetComponent<RectTransform> ().
sizeDelta = new Vector2 (mainSprite.texture.width, mainSprite.texture.
height);
                pdfImageGameObject.GetComponent<Image>().sprite =
mainSprite;
                pdfImageGameOb-
ject.GetComponent<Image>().preserveAspect = true;
                pdfImageGameObject.SetActive (true);
            }

        }
    }

    void disableAllPanels(){
        viewsPanel.SetActive (false);
        viewsPanelActive = false;

        displayModePanel.SetActive (false);
        displayModePanelActive = false;

        clipPanel.SetActive (false);
        DisablePlanesAndSlider ();
        clipOperationActive = false;

    }

    void DisablePlanesAndSlider(){
        planeX.SetActive (false);
        planeY.SetActive (false);
        planeZ.SetActive (false);
        planeSliderObj.SetActive (false);
        switchButtonImageController.SetDefaultClipButtons ();
        activePlane = -1;
    }

    public void ToggleMainPanel(){

        if (mainPanelHidden) {
            mainPanel.GetComponent<RectTransform> ().anchoredPosition =
new Vector2 (-0.5f, 0);
            mainPanelHideButton.sprite = leftArrow;
            mainPanelHidden = false;
        } else {
            mainPanel.GetComponent<RectTransform> ().anchoredPosition =
new Vector2 (-80, 0);
            mainPanelHideButton.sprite = rightArrow;
            mainPanelHidden = true;
        }

    }


    public void toggleViewsPanel(){

        if (viewsPanelActive) {
            leftPanelInfoText.text = "";
            viewsPanel.SetActive (false);
            viewsPanelActive = false;
        } else {
            disableAllPanels ();
            leftPanelInfoText.text = "Views";
            viewsPanel.SetActive (true);
            viewsPanelActive = true;
        }

    }

    public void toggleDisplayModePanel(){

        if (displayModePanelActive) {
            leftPanelInfoText.text = "";
            this.ShowMessagePermanent ("");
            displayModePanel.SetActive (false);
            displayModePanelActive = false;
            switchButtonImageController.CloseModelVolumeImage ();
            planeZNormalsOff.SetActive (false);
        } else {
            disableAllPanels ();
            leftPanelInfoText.text = "Display";
            displayModePanel.SetActive (true);
            displayModePanelActive = true;
        }

    }

    public void toggleOptionsPanel(){

        if (optionsPanelActive) {
            leftPanelInfoText.text = "";
            optionsPanel.SetActive (false);
            optionsPanelActive = false;
        } else {
            leftPanelInfoText.text = "Settings";
            optionsPanel.SetActive (true);
            optionsPanelActive = true;
        }

    }

    public void CloseOptionsPanel(){
        leftPanelInfoText.text = "";
        optionsPanel.SetActive (false);
        optionsPanelActive = false;
        switchButtonImageController.SwitchSettingsImage();

    }

    public void toggleClipOperation(){

        if (clipOperationActive) {
            leftPanelInfoText.text = "";
            clipPanel.SetActive (false);
            DisablePlanesAndSlider ();
            switchButtonImageController.SetDefaultClipButtons ();
            clipOperationActive = false;
        } else {
            disableAllPanels ();
            leftPanelInfoText.text = "Sections";
            clipPanel.SetActive (true);
            clipOperationActive = true;
        }

    }

    public void toggleColorPalette(){

        if (colorPaletteActive) {
            colorPalette.SetActive (false);
            colorPaletteActive = false;
        } else {
            colorPalette.SetActive (true);
            colorPaletteActive = true;
        }

    }

    public void CloseColorPalette(){
        colorPalette.SetActive (false);
        colorPaletteActive = false;
        switchButtonImageController.SwitchColorImage ();

    }

    public void togglePropertiesPanel(){

        if (propertiesPanelActive) {
            propertiesPanel.SetActive (false);
            propertiesPanelActive = false;
        } else {
            propertiesPanel.SetActive (true);
            propertiesPanelActive = true;
            UpdateProperties ();
        }

    }

    public void toggleModelVolumePanel(){

        if (modelVolumePanelActive) {
            this.ShowMessagePermanent ("");
            modelVolumePanelActive = false;
            objectVolumeText.enabled = false;
        } else {
            this.ShowMessagePermanent ("Model Volume");
            if (modelWeightPanelActive) {

                modelWeightPanelActive = false;
                modelWeightPanel.SetActive (false);
                switchButtonImageController.SwitchModelWeightImage ();

            }
            objectVolumeText.enabled = true;
            modelVolumePanelActive = true;

            if (canCalculateVolume) {
                calculator.CalculateVolumeAnimate (objectInCenter, 100,
false, 0);
            }
```

```
        }
    }

    public void SetCanCalculateVolume(bool value){
        canCalculateVolume = value;

    }

    public void toggleModelWeightPanel(){

        if (modelWeightPanelActive) {
            modelWeightPanel.SetActive (false);
            modelWeightPanelActive = false;
            objectVolumeText.enabled = false;
        } else {
            if (modelVolumePanelActive) {
                modelVolumePanelActive = false;
                switchButtonImageController.SwitchModelVolumeImage ();
                objectVolumeText.enabled = false;
            }
            modelWeightPanel.SetActive (true);
            modelWeightPanelActive = true;

        }
    }

    public void ClosePropertiesPanel(){
        propertiesPanel.SetActive (false);
        propertiesPanelActive = false;
        switchButtonImageController.SwitchPropertiesImage ();

    }

    void UpdateProperties(){
        if (objectInCenter != null){
            Vec-
tor3 boundsSize = objectInCenter.GetComponent<MeshFilter> ().mesh.bounds.
size;

            trianglesT-
ext.text = objectInCenter.GetComponent<MeshFilter> ().mesh.triangles.Leng
th.ToString ();
            verticesText.text = objectInCenter.GetComponent<MeshFilter>
().mesh.vertices.Length.ToString ();
            surfaceT-
ext.text = CalculateSurface ().ToString () + " mm\u00B2";

            dimensionsText.text = Math.Round(boundsSize.x,1) + " x " +
Math.Round(boundsSize.y,1) + " x " + Math.Round(boundsSize.z,1) + " mm";
        }
    }

    public void ChangeMaterial (){

        string selectedOption = materialDropdown.captionText.text;

        switch (selectedOption) {

        case "Aluminium":
            specificWeight = 2712f;
            specificWeightInput.SetActive (false);
            CalculateWeight ();
            break;

        case "Silver":
            specificWeight = 10490f;
            specificWeightInput.SetActive (false);
            CalculateWeight ();
            break;

        case "Brass":
            specificWeight = 8746f;
            specificWeightInput.SetActive (false);
            CalculateWeight ();
            break;

        case "Lead":
            specificWeight = 11340f;
            specificWeightInput.SetActive (false);
            CalculateWeight ();
            break;

        case "Nickel":
            specificWeight = 8800f;
            specificWeightInput.SetActive (false);
            CalculateWeight ();
            break;

        case "Iron":
            specificWeight = 7850f;
            specificWeightInput.SetActive (false);
            CalculateWeight ();
            break;

        case "Titanium":
            specificWeight = 4500f;
            specificWeightInput.SetActive (false);
            CalculateWeight ();
            break;

        case "Copper":
            specificWeight = 8930f;
            specificWeightInput.SetActive (false);
            CalculateWeight ();
            break;

        case "Steel":
            specificWeight = 7850f;
            specificWeightInput.SetActive (false);
            CalculateWeight ();
            break;

        case "Gold":
            specificWeight = 19320f;
            specificWeightInput.SetActive (false);
            CalculateWeight ();
            break;

        case "Chromium":
            specificWeight = 7190f;
            specificWeightInput.SetActive (false);
            CalculateWeight ();

            break;

        case "Grey Iron":
            specificWeight = 7300f;
            specificWeightInput.SetActive (false);
            CalculateWeight ();

            break;

        case "Custom":
            specificWeightInput.SetActive (true);
            break;

        }

    }

    public void CalculateWeight(){

        if (objectInCenter != null) {
            objectVolumeText.enabled = true;
            // Calculate weight of selected Material
            if (materialDropdown.captionText.text != "Custom") {

                if (canCalculateVolume) {
                    calculator.CalculateVolumeAnimate (objectInCenter,
100, true, specificWeight);
                }
            }
            // Calculate weight of custom Material
            else {
                string specificWeightString = inputTextSpecificWeight.
text;
                float specificWeightFloat = float.Parse (
specificWeightString);

                if (canCalculateVolume) {
                    calculator.CalculateVolumeAnimate (objectInCenter,
100, true, specificWeightFloat);
                }
            }
        }
    }

    public void CloseModelWeightPanel(){
        modelWeightPanel.SetActive (false);
        modelWeightPanelActive = false;
        objectVolumeText.enabled = false;
        switchButtonImageController.SwitchModelWeightImage ();

    }

    public float CalculateSurface(){

        return calculator.CalculateSurface (objectInCenter);

    }

    public float CalculateVolume(){

        return calculator.CalculateVolume (objectInCenter,100);

    }

    public void ScanModel(){

        StartCoroutine (LoadARScene ());
    }

    IEnumerator LoadARScene()
    {
        AsyncOperation asyncLoad = SceneManager.LoadSceneAsync (2,
LoadSceneMode.Single);

        while (!asyncLoad.isDone)
        {
            yield return null;
        }

    }

    public void ClipX(){
        StartCoroutine(ShowInfoMessageTemp ("Section X"));
        if (activePlane==0) {
            DisablePlanesAndSlider ();
        } else {
            DisablePlanesAndSlider ();
            planeSliderObj.SetActive (true);

            Bounds objectMeshBounds = objectInCenter.GetComponent<
MeshFilter> ().mesh.bounds;
            slider.minValue = objectMeshBounds.center.x -
objectMeshBounds.extents.x - 1;
            slider.maxValue = objectMeshBounds.center.x +
objectMeshBounds.extents.x + 1;

            slider.value = planeX.transform.position.x;
            sliderTitle.text = "X :";

            planePosi-
tion.text = Math.Round (slider.value, 2).ToString ();
            planeX.SetActive (true);
            //X plane Active (0)
            activePlane = 0;
        }


    }

    public void ClipY(){
        StartCoroutine(ShowInfoMessageTemp ("Section Y"));
        if (activePlane==1) {
            DisablePlanesAndSlider ();
        } else {
            DisablePlanesAndSlider ();
            planeSliderObj.SetActive (true);

            Bounds objectMeshBounds = objectInCenter.GetComponent<
```

```
MeshFilter> ().mesh.bounds;
            slider.minValue = objectMeshBounds.center.y -
objectMeshBounds.extents.y - 1;
            slider.maxValue = objectMeshBounds.center.y +
objectMeshBounds.extents.y + 1;

            slider.value = planeY.transform.position.y;
            sliderTitle.text = "Y :";

            planePosition.text = Math.Round (slider.value, 2).ToString();
            planeY.SetActive (true);
            //Y plane Active (1)
            activePlane = 1;
        }

    }

    public void ClipZ(){
        StartCoroutine(ShowInfoMessageTemp ("Section Z"));
        if (activePlane==2) {
            DisablePlanesAndSlider ();
        } else {
            DisablePlanesAndSlider ();
            planeSliderObj.SetActive (true);

            Bounds objectMeshBounds = objectInCenter.GetComponent<
MeshFilter> ().mesh.bounds;
            slider.minValue = objectMeshBounds.center.z -
objectMeshBounds.extents.z - 1;
            slider.maxValue = objectMeshBounds.center.z +
objectMeshBounds.extents.z + 1;

            slider.value = planeZ.transform.position.z;
            sliderTitle.text = "Z :";

            planePosition.text = Math.Round (
-slider.value, 2).ToString();
            planeZ.SetActive (true);
            //X plane Active (2)
            activePlane = 2;
        }

    }

    public void Clip(){
        StartCoroutine(ShowInfoMessageTemp ("Perform Section"));
        if (activePlane == 0) {
            switchButtonImageController.SwitchClipX ();
            if (planeXinverted) {
                clipMeshComponent.ClipMesh (Vector3.left, planeX.
transform.position);
            } else {
                clipMeshComponent.ClipMesh (Vector3.right, planeX.
transform.position);
            }
        }
        else if (activePlane == 1) {
            switchButtonImageController.SwitchClipY ();
            if (planeYinverted) {
                clipMeshComponent.ClipMesh (Vector3.down, planeY.
transform.position);
            } else {
                clipMeshComponent.ClipMesh (Vector3.up, planeY.
transform.position);
            }
        }
        else if (activePlane == 2) {
            switchButtonImageController.SwitchClipZ ();
            if (planeZinverted) {
                clipMeshComponent.ClipMesh (Vector3.forward, planeZ.
transform.position);
            } else {
                clipMeshComponent.ClipMesh (Vector3.back, planeZ.
transform.position);
            }

        }
        modelVolume = -1;
        UpdateProperties ();
        DisablePlanesAndSlider ();

    }

    public void InvertPlane(){
        StartCoroutine(ShowInfoMessageTemp ("Invert Axis"));
        if (activePlane == 0) {
            planeXinverted = !planeXinverted;
            planeX.transform.eulerAngles = new Vector3 (0, planeX.
transform.eulerAngles.y - 180, 0);

        }
        else if (activePlane == 1) {
            planeYinverted = !planeYinverted;
            planeY.transform.rotation = Quaternion.Inverse (planeY.
transform.rotation);

        }
        else if (activePlane == 2) {
            planeZinverted = !planeZinverted;
            planeZ.transform.rotation = Quaternion.Inverse (planeZ.
transform.rotation);

        }
    }

    public void UpdatePlanePosition(){

        if (activePlane == 0) {
            Vector3 position = planeX.transform.position;
            position.x = slider.value;
            planeX.transform.position = position;

            planePosition.text = Math.Round (slider.value, 2).ToString();
        }
        else if (activePlane == 1) {
            Vector3 position = planeY.transform.position;
            position.y = slider.value;
            planeY.transform.position = position;

            planePosition.text = Math.Round (slider.value, 2).ToString();
        }
        else if (activePlane == 2) {
            Vector3 position = planeZ.transform.position;
            position.z = slider.value;
            planeZ.transform.position = position;

            planePosition.text = Math.Round (-
slider.value, 2).ToString();

        }

    }

    // Function that adjusts the clipping plane positions and scales in t
he respect to the ObjectInCenter scale and position
    void ReadjustPlanes(){

        Vector3 boundsSize = objectInCenter.GetComponent<MeshFilter> ().
mesh.bounds.size;
        float maxDimension = Mathf.Max (boundsSize.x, boundsSize.y,
boundsSize.z);
        int planeSize = Mathf.RoundToInt (maxDimension) + 1;

        planeX.transform.position = objectInCenter.GetComponent<
MeshFilter> ().mesh.bounds.center;
        planeY.transform.position = objectInCenter.GetComponent<
MeshFilter> ().mesh.bounds.center;
        planeZ.transform.position = objectInCenter.GetComponent<
MeshFilter> ().mesh.bounds.center;

        planeX.transform.localScale = new Vector3 (planeSize/ 100f,
planeSize, planeSize);
        planeY.transform.localScale = new Vector3 (planeSize/ 100f,
planeSize, planeSize);
        planeZ.transform.localScale = new Vector3 (planeSize/ 100f,
planeSize, planeSize);

    }

    public void ChangeObjectColor(){
        GameObject selectedGO = EventSystem.current.
currentSelectedGameObject;
        Color selectedColor = selectedGO.GetComponent<Image> ().color;
        Material mat = objectInCenter.GetComponent<Renderer> ().material;
        mat.color = selectedColor;

        selectedColorIndicator.GetComponent<RectTransform> ().
anchoredPosition = selectedGO.GetComponent<RectTransform> ().
anchoredPosition;

    }

    public void ChangeTextColor(Color32 color){
        notificationText.color = color;
        leftPanelInfoText.color = color;
        rightPanelInfoText.color = color;
        sliderTitle.color = color;
        planePosition.color = color;
        objectVolumeText.color = color;

    }


    public void toggleSavePanel(){

        if (savePanelActive) {
            leftPanelInfoText.text = "";
            savePanel.SetActive (false);
            savePanelActive = false;

        } else {
            leftPanelInfoText.text = "Save Model";
            if (openPanelActive) {

                foreach (Transform child in savedModelsPanel.transform) {
                    GameObject.Destroy(child.gameObject);
                }
                openPanel.SetActive (false);
                openPanelActive = false;
                switchButtonImageController.SwitchOpenImage ();
            }

            savePanel.SetActive (true);
            savePanelActive = true;
            saveOutput.text = "";
            inputText.text = "";
        }

    }

    public void CloseSavePanel(){
        leftPanelInfoText.text = "";
        savePanel.SetActive (false);
        savePanelActive = false;
        switchButtonImageController.SwitchSaveImage ();
    }

    public void RestoreOriginalModel(){
        StartCoroutine (ShowInfoMessageTemp ("Restore Original Model"));
        if (objectInCenter != null) {

            Mesh objectMesh = objectInCenter.GetComponent<MeshFilter> ().
mesh;
            objectMesh.Clear ();

            objectMesh.vertices = modelVertices;
            objectMesh.normals = modelNormals;
            objectMesh.triangles = modelTriangles;

            UpdateProperties ();
            clipMeshComponent.InitializePlanePositions ();
        }
    }


    public void Save(){

        String modelName = inputText.text;

        if (File.Exists (Application.persistentDataPath + "/" +
```

```
modelName)) {
            saveOutput.text = "Name already Exists!";
        } else if (objectInCenter == null) {
            saveOutput.text = "There is no model in editor!";
        }
        else {
            BinaryFormatter bf = new BinaryFormatter ();

            SurrogateSelector ss = new SurrogateSelector ();
            Vector3SerializationSurrogate v3ss = new
Vector3SerializationSurrogate ();
            ss.AddSurrogate (typeof(Vector3), new StreamingContext (
StreamingContextStates.All), v3ss);
            bf.SurrogateSelector = ss;

            FileStream file = File.Create (Application.
persistentDataPath + "/"+modelName);

            MeshFilter objectMeshFilter = objectInCenter.
GetComponent<MeshFilter> ();
            Mesh objectMesh = objectMeshFilter.mesh;

            ObjectMeshData data = new ObjectMeshData ();
            data.vertices = objectMesh.vertices;
            data.normals = objectMesh.normals;
            data.triangles = objectMesh.triangles;
            data.objectName = objectInCenter.name;

            saveOutput.text = "Save Successful!";

            bf.Serialize (file, data);
            file.Close ();
        }
    }

    public void OpenSavedModelList(){

        if (openPanelActive) {
            leftPanelInfoText.text = "";
            foreach (Transform child in savedModelsPanel.transform) {
                GameObject.Destroy(child.gameObject);
            }
            openPanel.SetActive (false);
            openPanelActive = false;
        } else {

            leftPanelInfoText.text = "Open Saved Model";

            if (savePanelActive) {
                savePanelActive = false;
                switchButtonImageController.SwitchSaveImage ();
            }

            openPanel.SetActive (true);
            openPanelActive = true;

            DirectoryInfo dInfo = new DirectoryInfo (Application.
persistentDataPath);
            FileInfo[] fInfo = dInfo.GetFiles ();

            // Create as many buttons as the saved models
            for (int i = 0; i < fInfo.Length; i++){

                // ignore png files as they dont correspond to saved
models
                if (fInfo [i].Name.Length>=4 && fInfo [i].Name.
Substring (fInfo [i].Name.Length - 4) == ".png") {
                    continue;
                }

                GameOb-
ject modelButton = Instantiate (modelButtonPrefab, savedModelsPanel.GetCo
mponent<RectTransform> ());

                Button btn = modelButton.GetComponent<Button> ();
                btn.onClick.AddListener (this.SelectSavedModel);
                //set name
                modelButton.GetComponentInChildren<Text> ().text =
fInfo[i].Name;

            }

        }

    }

    public void CloseSavedModelList(){
        leftPanelInfoText.text = "";
        foreach (Transform child in savedModelsPanel.transform) {
            GameObject.Destroy(child.gameObject);
        }
        openPanel.SetActive (false);
        openPanelActive = false;
        switchButtonImageController.SwitchOpenImage ();

    }


    public void SelectSavedModel(){

        GameObject selectedGO = EventSystem.current.
currentSelectedGameObject;
        selectedModelName = selectedGO.GetComponentInChildren<Text> ().
text;

    }

    public void DeleteSavedModel(){

        confirmPanel.SetActive (true);

    }

    public void ConfirmDeletion(){

        if (File.Exists (Application.persistentDataPath + "/" +
selectedModelName)) {

            File.Delete (Application.persistentDataPath + "/" +
selectedModelName);

            // Update Model Panel after deletion
            foreach (Transform child in savedModelsPanel.transform) {
                GameObject.Destroy(child.gameObject);
            }
            DirectoryInfo dInfo = new DirectoryInfo (Application.
persistentDataPath);
            FileInfo[] fInfo = dInfo.GetFiles ();

            // Create as many buttons as the saved models
            for (int i = 0; i < fInfo.Length; i++){

                // ignore png files as they dont correspond to saved
models
                if (fInfo [i].Name.Length>=4 && fInfo [i].Name.Substring
(fInfo [i].Name.Length - 4) == ".png") {
                    continue;
                }

                GameOb-
ject modelButton = Instantiate (modelButtonPrefab, savedModelsPanel.GetCo
mponent<RectTransform> ());
                Button btn = modelButton.GetComponent<Button> ();
                btn.onClick.AddListener (this.SelectSavedModel);
                //set name
                modelButton.GetComponentInChildren<Text> ().text =
fInfo[i].Name;

            }

        }

        confirmPanel.SetActive (false);

    }

    public void RejectDeletion(){

        confirmPanel.SetActive (false);
    }

    public void Load(){

        if (File.Exists (Application.persistentDataPath + "/"+
selectedModelName)) {
            BinaryFormatter bf = new BinaryFormatter ();

            SurrogateSelector ss = new SurrogateSelector ();
            Vec-
tor3SerializationSurrogate v3ss = new Vector3SerializationSurrogate ();
            ss.AddSurrogate (typeof(Vector3), new StreamingContext (
StreamingContextStates.All), v3ss);
            bf.SurrogateSelector = ss;

            FileStream file = File.Open (Application.persistentDataPath +
 "/"+selectedModelName, FileMode.Open);

            ObjectMeshData data = (ObjectMeshData)bf.Deserialize (file);

            modelVertices = data.vertices;
            modelNormals = data.normals;
            modelTriangles = data.triangles;

            file.Close ();

            if (objectInCenter != null) {
                MeshFilter objectMeshFilter = objectInCenter.
GetComponent<MeshFilter> ();
                Mesh objectMesh = objectMeshFilter.mesh;

                objectMesh.Clear ();

                objectMesh.vertices = data.vertices;
                objectMesh.normals = data.normals;
                objectMesh.triangles = data.triangles;
                objectInCenter.name = data.objectName;
            } else {

                objectInCenter = new GameObject ("ObjectInCenter");
                objectInCenter.tag = "ScreenCenter";
                Materi-
al mat = objectInCenter.AddComponent<MeshRenderer> ().material = objectMa
terial;
                mat.shader = Shader.Find ("Standard");

                MeshFilter mf = objectInCenter.AddComponent<MeshFilter>
();
                Mesh objectMesh = mf.mesh;

                objectMesh.vertices = data.vertices;
                objectMesh.normals = data.normals;
                objectMesh.triangles = data.triangles;
                objectInCenter.name = data.objectName;

            }

            ReadjustPlanes ();
            cameraControl.UpdateObjectInCenter (objectInCenter);

            foreach (Transform child in savedModelsPanel.transform) {
                GameObject.Destroy(child.gameObject);
            }

            openPanel.SetActive (false);
            openPanelActive = false;

            switchButtonImageController.SwitchOpenImage ();
            UpdateProperties ();
            modelVolume = -1;

        }

    }

    public void ShowMessagePermanent(string msg){
        rightPanelInfoText.text = msg;
    }

    public void CallShowMessage(string msg){
        StartCoroutine(ShowInfoMessageTemp (msg));
```

```
    }

    IEnumerator ShowInfoMessageTemp(string msg){

        messagesCount++;
        rightPanelInfoText.text = msg;
        yield return new WaitForSeconds (2f);
        messagesCount--;

        if (messagesCount == 0) {
            rightPanelInfoText.text = "";
        }
    }

[Serializable]
class ObjectMeshData
{
    public Vector3[] vertices;
    public Vector3[] normals;
    public int[] triangles;
    public string objectName;
}

sealed class Vector3SerializationSurrogate : ISerializationSurrogate {

    // Method called to serialize a Vector3 object
    public void GetObjectData(System.Object obj,
        SerializationInfo info, StreamingContext context) {

        Vector3 v3 = (Vector3) obj;
        info.AddValue("x", v3.x);
        info.AddValue("y", v3.y);
        info.AddValue("z", v3.z);

    }

    // Method called to deserialize a Vector3 object
    public System.Object SetObjectData(System.Object obj,
        SerializationInfo info, StreamingContext context,
        ISurrogateSelector selector) {

        Vector3 v3 = (Vector3) obj;
        v3.x = (float)info.GetSingle ("x");
        v3.y = (float)info.GetSingle ("y");
        v3.z = (float)info.GetSingle ("z");
        obj = v3;
        return obj;
    }
}
```

SwitchButtonImage

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;


public class SwitchButtonImage : MonoBehaviour {

    public Button scanButton;
    public Sprite scanOn;
    public Sprite scanOff;

    public Button saveButton;
    public Sprite saveOn;
    public Sprite saveOff;

    public Button openButton;
    public Sprite openOn;
    public Sprite openOff;

    public Button pngButton;
    public Sprite pngOn;
    public Sprite pngOff;

    public Button displayButton;
    public Sprite displayOn;
    public Sprite displayOff;

    public Button viewsButton;
    public Sprite viewsOn;
    public Sprite viewsOff;

    public Button clipButton;
    public Sprite clipOn;
    public Sprite clipOff;

    public Button settingButton;
    public Sprite settingOn;
    public Sprite settingOff;

    public Button exitButton;
    public Sprite exitOn;
    public Sprite exitOff;

    public Button colorButton;
    public Sprite colorOn;
    public Sprite colorOff;

    public Button propertiesButton;
    public Sprite propertiesOn;
    public Sprite propertiesOff;

    public Button modelWeightButton;
    public Sprite modelWeightOn;
    public Sprite modelWeightOff;

    public Button modelVolumeButton;
    public Sprite modelVolumeOn;
    public Sprite modelVolumeOff;

    public Button normalModeButton;
    public Sprite normalModeOn;
    public Sprite normalModeOff;

    public Button highlightModeButton;
    public Sprite highlightModeOn;
    public Sprite highlightModeOff;

    public Button clipXButton;
    public Sprite clipXOn;
    public Sprite clipXOff;

    public Button clipYButton;
    public Sprite clipYOn;
    public Sprite clipYOff;

    public Button clipZButton;
    public Sprite clipZOn;
    public Sprite clipZOff;


    void SetDefaultImages(){
        scanButton.image.sprite = scanOff;
        displayButton.image.sprite = displayOff;
        viewsButton.image.sprite = viewsOff;
        clipButton.image.sprite = clipOff;
    }

    public void SwitchScanImage(){

        if (scanButton.image.sprite == scanOn) {

            scanButton.image.sprite = scanOff;
        } else {
            SetDefaultImages ();
            scanButton.image.sprite = scanOn;
        }

    }

    public void SwitchSaveImage(){

        if (saveButton.image.sprite == saveOn) {
            saveButton.image.sprite = saveOff;
        } else {

            saveButton.image.sprite = saveOn;
        }

    }


    public void SwitchOpenImage(){

        if (openButton.image.sprite == openOn) {
            openButton.image.sprite = openOff;
        } else {

            openButton.image.sprite = openOn;
        }

    }

    public void SwitchPNGImage(){

        if (pngButton.image.sprite == pngOn) {
            pngButton.image.sprite = pngOff;
        } else {

            pngButton.image.sprite = pngOn;
        }

    }

    public void SwitchDisplayImage(){

        if (displayButton.image.sprite == displayOn) {

            displayButton.image.sprite = displayOff;
        } else {
            SetDefaultImages ();
            displayButton.image.sprite = displayOn;
        }

    }

    public void SwitchViewsImage(){

        if (viewsButton.image.sprite == viewsOn) {

            viewsButton.image.sprite = viewsOff;
        } else {
            SetDefaultImages ();
            viewsButton.image.sprite = viewsOn;
        }

    }

    public void SwitchClipImage(){

        if (clipButton.image.sprite == clipOn) {
            clipButton.image.sprite = clipOff;
        } else {
            SetDefaultImages ();
            clipButton.image.sprite = clipOn;
        }

    }

    public void SwitchSettingsImage(){

        if (settingButton.image.sprite == settingOn) {
            settingButton.image.sprite = settingOff;
        } else {
            settingButton.image.sprite = settingOn;
        }

    }

    public void SwitchColorImage(){

        if (colorButton.image.sprite == colorOn) {
            colorButton.image.sprite = colorOff;
        } else {
            colorButton.image.sprite = colorOn;
        }

    }
```

```
    public void SwitchPropertiesImage(){

        if (propertiesButton.image.sprite == propertiesOn) {
            propertiesButton.image.sprite = propertiesOff;
        } else {
            propertiesButton.image.sprite = propertiesOn;
        }

    }

    public void SwitchModelVolumeImage(){

        if (modelVolumeButton.image.sprite == modelVolumeOn) {
            modelVolumeButton.image.sprite = modelVolumeOff;
        } else {
            modelVolumeButton.image.sprite = modelVolumeOn;
        }

    }

    public void CloseModelVolumeImage(){

        modelVolumeButton.image.sprite = modelVolumeOff;
    }

    public void SwitchModelWeightImage(){
        if (modelWeightButton.image.sprite == modelWeightOn) {
            modelWeightButton.image.sprite = modelWeightOff;
        } else {
            modelWeightButton.image.sprite = modelWeightOn;
        }

    }

    void DisableDisplayModes(){
        normalModeButton.image.sprite = normalModeOff;
        highlightModeButton.image.sprite = highlightModeOff;

    }


    public void SwitchNormalMode(){

        if (normalModeButton.image.sprite == normalModeOff) {
            DisableDisplayModes ();
            normalModeButton.image.sprite = normalModeOn;
        }
    }

    public void SwitchHighlightMode(){

        if (highlightModeButton.image.sprite == highlightModeOff) {
            DisableDisplayModes ();
            highlightModeButton.image.sprite = highlightModeOn;
        }
    }

    void DisableAxisImages(){
        clipXButton.image.sprite = clipXOff;
        clipYButton.image.sprite = clipYOff;
        clipZButton.image.sprite = clipZOff;
    }

    public void SwitchClipX(){

        if (clipXButton.image.sprite == clipXOn) {
            clipXButton.image.sprite = clipXOff;
        } else {
            DisableAxisImages ();
            clipXButton.image.sprite = clipXOn;
        }

    }

    public void SwitchClipY(){

        if (clipYButton.image.sprite == clipYOn) {
            clipYButton.image.sprite = clipYOff;
        } else {
            DisableAxisImages ();
            clipYButton.image.sprite = clipYOn;
        }
    }

    public void SwitchClipZ(){

        if (clipZButton.image.sprite == clipZOn) {
            clipZButton.image.sprite = clipZOff;
        } else {
            DisableAxisImages ();
            clipZButton.image.sprite = clipZOn;
        }

    }

    public void SwitchExitButton(){

        if (exitButton.image.sprite == exitOn) {
            exitButton.image.sprite = exitOff;
        } else {
            exitButton.image.sprite = exitOn;
        }
    }

    public void SetDefaultClipButtons(){
        clipXButton.image.sprite = clipXOff;
        clipYButton.image.sprite = clipYOff;
        clipZButton.image.sprite = clipZOff;
    }
}
```

CameraController

```
using System.Collections;
using System.Collections.Generic;
using System;
using UnityEngine;
using UnityEngine.EventSystems;
using UnityStandardAssets.ImageEffects;
using UnityEngine.UI;

public class CameraController : MonoBehaviour {

    public float minFieldOfView;
    public float maxFieldOfView;
    public float zoomSensitivity;
    public float distanceOffset;

    public float moveSpeed;

    private Touch initTouch = new Touch ();

    private float rotationX = 0f;
    private float rotationY = 0f;
    private Vector3 originalRotation;

    public float rotationSpeed;
    public float direction = -1;

    Camera cameraComponent;
    Transform cameraPivot;
    public Transform axisCameraPivot;
    public Camera axisCamera;
    public Transform axisObject;
    public GameObject axisRenderer;

    private bool swipePossible =true;
    private bool infoMessageOn = false;

    GameObject objectInCenter;

    public Slider zoomSensitivitySlider;
    public Text zoomSliderValue;
    public Slider moveSensitivitySlider;
    public Text moveSliderValue;
    public Slider rotateSensitivitySlider;
    public Text rotateSliderValue;
    public Slider edgeSensitivitySlider;
    public Text edgeSliderValue;

    public Dropdown backgroundDropdown;

    public Toggle perspectiveToggle;

    public InterfaceController interfaceController;

    void Start () {

        cameraComponent = GetComponent<Camera> ();
        cameraPivot =  this.transform.parent;

        originalRotation = cameraPivot.eulerAngles;
        rotationX = originalRotation.x;
        rotationY = originalRotation.y;

        objectIn-
Center = GameObject.FindGameObjectWithTag ("ScreenCenter");

        LoadPlayerPrefs ();

        FitObjectInCameraView ();
        DrawAxisObject ();

    }

    // Update is called once per frame
    void Update () {

        if (swipePossible) {
            // Camera Rotation
            if (Input.touchCount == 1) {

                Touch touch = Input.GetTouch (0);

                if (touch.phase == TouchPhase.Began) {

                    initTouch = touch;
                } else if (touch.phase == TouchPhase.Moved) {

                    if (infoMessageOn) {
                        interfaceController.ShowMessagePermanent ("");
                    }

                    float deltaX = initTouch.position.x -
 touch.position.x;
                    float deltaY = initTouch.position.y -
 touch.position.y;

                    rotationX -
= deltaY * Time.deltaTime * rotationSpeed * direction;
                    rota-
tionY += deltaX * Time.deltaTime * rotationSpeed * direction;

                    cameraPivot.eulerAngles = new Vector3 (rotationX,
rotationY, 0f);

                } else if (touch.phase == TouchPhase.Ended) {

                    initTouch = new Touch ();
                }

            }

            // Camera Zoom
            if (Input.touchCount == 2) {

                Touch touch0 = Input.GetTouch (0);
                Touch touch1 = Input.GetTouch (1);

                Vector2 touch0PrevPos = touch0.position -
 touch0.deltaPosition;
                Vector2 touch1PrevPos = touch1.position -
 touch1.deltaPosition;

                float prevTouchDeltaMag = (touch0PrevPos -
 touch1PrevPos).magnitude;
                float touchDeltaMag = (touch0.position -
 touch1.position).magnitude;
```

```
                float deltaMagnitudeDiff = prevTouchDeltaMag -
 touchDeltaMag;

                // Object's max Bound used to find zoom and move
sensitivity;
                Vector3 boundsSize = objectInCenter.GetComponent<
MeshFilter> ().mesh.bounds.size;
                float maxBound = Mathf.Max (boundsSize.x, boundsSize.y,
boundsSize.z);

                // Move Camera
                if (Mathf.Abs (deltaMagnitudeDiff) < 8f && Input.
GetTouch (0).phase == TouchPhase.Moved && Input.GetTouch (1).phase ==
TouchPhase.Moved) {
                    Vec-
tor2 touchDelta = Input.GetTouch (0).deltaPosition;

                    cameraPivot.Translate (new Vector3 (-
touchDelta.x * maxBound * 0.005f * moveSpeed, -
touchDelta.y * maxBound * 0.005f * moveSpeed, 0));


                } else {

                    if (cameraComponent.orthographic) {

                        cameraComponent.orthographicSize +=
deltaMagnitudeDiff * maxBound * 0.005f * zoomSensitivity;
                        cameraComponent.orthographicSize = Mathf.Clamp (
cameraComponent.orthographicSize, 0.1f, float.PositiveInfinity);
                    } else {
                        this.transform.localPosition -
= new Vector3 (0, 0, deltaMagnitudeDiff * maxBound * 0.005f *
zoomSensitivity);

                    }
                }
            }
        }

        DrawAxisObject ();
    }


    public void LoadPlayerPrefs(){

        zoomSensitivity = zoomSensitivitySlider.value = PlayerPrefs.
GetFloat ("ZoomSensitivity",1);
        zoomSliderVal-
ue.text = Math.Round (zoomSensitivity, 2).ToString();

        moveSpeed = moveSensitivitySlider.value = PlayerPrefs.GetFloat
("MoveSpeed",1);
        moveSliderValue.text = Math.Round (moveSpeed, 2).ToString();

        rotationSpeed = rotateSensitivitySlider.value = PlayerPrefs.
GetFloat ("RotateSpeed", 2);
        rotateSliderVal-
ue.text = Math.Round (rotationSpeed, 2).ToString();

        this.GetComponent<EdgeDetection> ().sensitivityDepth =
edgeSensitivitySlid-
er.value = PlayerPrefs.GetFloat ("EdgesSensitivity", 30);
        edgeSliderVal-
ue.text = Math.Round (rotateSensitivitySlider.value, 2).ToString();

        background-
Dropdown.value = PlayerPrefs.GetInt ("BackgroundColor",0);
        if (backgroundDropdown.value == 0) {
            cameraCompo-
nent.backgroundColor = new Color32 (182, 188, 195, 255);
            interfaceController.ChangeTextColor (Color.black);
        } else if (backgroundDropdown.value == 1) {
            cameraComponent.backgroundColor = Color.white;
            interfaceController.ChangeTextColor (Color.black);
        } else {
            cameraComponent.backgroundColor = new Color32 (38, 38, 38,
255);
            interfaceController.ChangeTextColor (Color.white);
        }

        if (PlayerPrefs.GetInt ("CameraProjection", 1) == 0) {
            cameraComponent.orthographic = false;
            perspectiveToggle.isOn = true;
        } else {
            cameraComponent.orthographic = true;
            this.transform.localPosition = new Vector3 (0, 0, -900);
            perspectiveToggle.isOn = false;

        }

    }


    public void RestoreDefaultOptions(){
        zoomSensitivity = zoomSensitivitySlider.value = 1;
        zoomSliderValue.text = zoomSensitivity.ToString ();
        PlayerPrefs.SetFloat ("ZoomSensitivity", zoomSensitivity);

        moveSpeed = moveSensitivitySlider.value = 1;
        moveSliderValue.text = moveSpeed.ToString ();
        PlayerPrefs.SetFloat ("MoveSpeed", moveSpeed);

        rotationSpeed = rotateSensitivitySlider.value = 2;
        rotateSliderValue.text = rotationSpeed.ToString();
        PlayerPrefs.SetFloat ("RotateSpeed", rotationSpeed);

        this.GetComponent<EdgeDetection> ().sensitivityDepth =
edgeSensitivitySlider.value = 30;
        edgeSliderVal-
ue.text = Math.Round (rotateSensitivitySlider.value, 2).ToString();
        PlayerPrefs.SetFloat ("EdgesSensitivity", rotationSpeed);

        cameraCompo-
nent.backgroundColor = new Color32 (182, 188, 195, 255);
        interfaceController.ChangeTextColor (Color.black);
        backgroundDropdown.value = 0;
        PlayerPrefs.SetInt ("BackgroundColor", 0);

        cameraComponent.orthographic = true;
        this.transform.localPosition = new Vector3 (0, 0, -900);
        perspectiveToggle.isOn = false;
        PlayerPrefs.SetInt ("CameraProjection", 1);


    }


    public void ChangeBackgroundColor(){

        string selectedOption = backgroundDropdown.captionText.text;

        switch (selectedOption) {

        case "Skybox":
            cameraCompo-
nent.backgroundColor = new Color32 (182, 188, 195, 255);
            interfaceController.ChangeTextColor (Color.black);
            PlayerPrefs.SetInt ("BackgroundColor", 0);
            break;

        case "White":
            cameraComponent.backgroundColor = Color.white;
            interfaceController.ChangeTextColor (Color.black);
            PlayerPrefs.SetInt ("BackgroundColor", 1);
            break;

        case "Black":

            cameraComponent.backgroundColor = new Color32 (38, 38, 38,
255);
            interfaceController.ChangeTextColor (Color.white);
            PlayerPrefs.SetInt ("BackgroundColor", 2);
            break;

        }
    }

    public void ChangeCameraProjection(){

        if (perspectiveToggle.isOn) {
            cameraComponent.orthographic = false;
            this.GetComponent<EdgeDetection> ().sensitivityDepth = 1f;
            FitObjectInCameraView ();
            PlayerPrefs.SetInt ("CameraProjection", 0);
        } else {
            cameraComponent.orthographic = true;
            this.GetComponent<EdgeDetection> ().sensitivityDepth = 30f;
            this.transform.localPosition = new Vector3 (0, 0, -900);
            FitObjectInCameraView ();
            PlayerPrefs.SetInt ("CameraProjection", 1);
        }

    }

    public void    UpdateZoomSensitivity(){
        zoomSensitivity = zoomSensitivitySlider.value;
        zoomSliderVal-
ue.text = Math.Round (zoomSensitivity, 2).ToString();

        PlayerPrefs.SetFloat ("ZoomSensitivity", zoomSensitivity);
    }

    public void    UpdateMoveSensitivity(){
        moveSpeed = moveSensitivitySlider.value;
        moveSliderValue.text = Math.Round (moveSpeed, 2).ToString();

        PlayerPrefs.SetFloat ("MoveSpeed", moveSpeed);
    }

    public void    UpdateRotateSensitivity(){
        rotationSpeed = rotateSensitivitySlider.value;
        rotateSliderVal-
ue.text = Math.Round (rotationSpeed, 2).ToString();

        PlayerPrefs.SetFloat ("RotateSpeed", rotationSpeed);
    }

    public void    UpdateEdgesSensitivity(){
        this.GetComponent<EdgeDetection> ().sensitivityDepth =
edgeSensitivitySlider.value;

        edgeSliderValue.text = Math.Round (edgeSensitivitySlider.value,
2).ToString();

        PlayerPrefs.SetFloat ("EdgesSensitivity", edgeSensitivitySlider.
value);

    }

    public void  SwipeOn(){

        swipePossible = true;
    }

    public void  SwipeOff(){

        swipePossible = false;
    }

    public void NormalShadedMode(){

        this.GetComponent<EdgeDetection> ().enabled = false;
        interfaceController.CallShowMessage ("Normal Shaded Mode");
    }

    public void HighlightEdgesMode(){

        this.GetComponent<EdgeDetection> ().enabled = true;
        interfaceController.CallShowMessage ("Highlight Edges Mode");

    }

    public void PerspectiveView(){
        infoMessageOn = true;
        interfaceController.ShowMessagePermanent ("Perspective View");
        rotationX = 40;
        rotationY = -40;
        cameraPivot.eulerAngles = new Vector3 (rotationX, rotationY, 0);
        DrawAxisObject ();

    }
```

```
    public void TopView(){
        infoMessageOn = true;
        interfaceController.ShowMessagePermanent ("Top View");
        rotationX = 90;
        rotationY = 0;
        cameraPivot.eulerAngles = new Vector3 (rotationX, rotationY, 0);
        DrawAxisObject ();

    }

    public void BottomView(){
        infoMessageOn = true;
        interfaceController.ShowMessagePermanent ("Bottom View");
        rotationX = -90;
        rotationY = 0;
        cameraPivot.eulerAngles = new Vector3 (rotationX, rotationY, 0);
        DrawAxisObject ();

    }

    public void LeftView(){
        infoMessageOn = true;
        interfaceController.ShowMessagePermanent ("Left View");
        rotationX = 0;
        rotationY = 90;
        cameraPivot.eulerAngles = new Vector3 (rotationX, rotationY, 0);
        DrawAxisObject ();

    }

    public void RightView(){
        infoMessageOn = true;
        interfaceController.ShowMessagePermanent ("Right View");
        rotationX = 0;
        rotationY = -90;
        cameraPivot.eulerAngles = new Vector3 (rotationX, rotationY, 0);
        DrawAxisObject ();

    }

    public void FrontView(){
        infoMessageOn = true;
        interfaceController.ShowMessagePermanent ("Front View");
        rotationX = 0;
        rotationY = 0;
        cameraPivot.eulerAngles = new Vector3 (rotationX, rotationY, 0);
        DrawAxisObject ();

    }

    public void BackView(){
        infoMessageOn = true;
        interfaceController.ShowMessagePermanent ("Rear View");
        rotationX = 0;
        rotationY = 180;
        cameraPivot.eulerAngles = new Vector3 (rotationX, rotationY, 0);
        DrawAxisObject ();

    }

    void DrawAxisObject(){

        axisCameraPivot.eulerAngles = cameraPivot.eulerAngles;
        axisCameraPivot.position = cameraPivot.position;

        Vector3 position;
        if (cameraComponent.orthographic) {
            axisCamera.orthographic = true;
            axisCamera.orthographicSize = cameraComponent.
orthographicSize;
            axisObject.localScale = new Vector3 (axisCamera.
orthographicSize * 0.25f, axisCamera.orthographicSize * 0.25f,
axisCamera.orthographicSize * 0.25f);
            position = axisCamera.ViewportToWorldPoint (new Vector3 (
0.81f, 0.22f, axisCamera.farClipPlane - 100));
            SetRect (axisRenderer.GetComponent<RectTransform> (), 0,0,0,0
);
        }
        else{
            axisCamera.orthographic = false;
            axisCamera.transform.localPosition = cameraComponent.
transform.localPosition;
            axisObject.localScale = new Vector3 (25, 25, 25);
            position = axisCamera.ViewportToWorldPoint (new Vector3 (
0.6f, 0.4f, axisCamera.farClipPlane - 1800));

            SetRect (axisRenderer.GetComponent<RectTransform> (), 265,
125, -265, -125);

        }

        axisObject.position = position;
    }

    pub-
lic static void SetRect(RectTransform trs, float left, float top, float r
ight, float bottom)
    {
        trs.offsetMin = new Vector2(left, bottom);
        trs.offsetMax = new Vector2(-right, -top);
    }


    public void FitObjectInCameraView(){

        if (objectInCenter != null) {

            cameraPivot.transform.position = objectInCenter.
GetComponent<MeshFilter> ().mesh.bounds.center;
            Vec-
tor3 boundsSize = objectInCenter.GetComponent<MeshFilter> ().mesh.bounds.
size;
            float maxBound = Mathf.Max (boundsSize.x, boundsSize.y,
boundsSize.z);

            if (cameraComponent.orthographic) {
                cameraComponent.orthographicSize = maxBound / 1.75f +
distanceOffset;

            } else {
                float distance = maxBound * 0.5f / Mathf.Tan (0.5f *
cameraComponent.fieldOfView * Mathf.Deg2Rad);

                this.transform.localPosition = new Vector3 (this.
transform.localPosition.x, this.transform.localPosition.y, -
distance * distanceOffset);
            }
        }
    }

    public void UpdateObjectInCenter(GameObject newObject){
        objectInCenter = newObject;
        FitObjectInCameraView ();
        DrawAxisObject ();
    }
}
```

MeshClip

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;

public class MeshClip : MonoBehaviour {

    private GameObject objectToClip;

    public GameObject instGO;
    public GameObject instGO2;

    private Plane clipPlane;

    private MeshFilter objectMeshFilter;
    private Mesh objectMesh;

    // Neccesary for Undo Option
    private Vector3[] oldVertices;
    private Vector3[] oldNormals;
    private int[] oldTriangles;

    private List<Vector3> newVertices;
    private int[,] vertexInfo;
    private List<int> newTriangles;
    private List<Vector3> newNormals;
    private List<Vector2> newUVs;

    private Vector3[] oldPlanePositions;

    private int listIndex;

    List<IntersectionEdge> intersectionEdges;
    List<Loop> loops;

    void Start () {
        newVertices = new List<Vector3>();
        newTriangles = new List<int> ();
        newNormals = new List<Vector3> ();
        //newUVs = new List<Vector2> ();
        intersectionEdges = new List<IntersectionEdge> ();
        loops = new List<Loop> ();

        // Initialize old positions of clipping planes
        oldPlanePositions = new Vector3[6];    // Array storing 6 plane
positions as: X plane, X plane inverted, Y plane, Y plane inverted,
Z plane, Z plane inverted

        InitializePlanePositions ();
    }

    public void UndoClip(){

        objectMesh.Clear ();

        objectMesh.vertices = oldVertices;
        objectMesh.normals = oldNormals;
        //objectMesh.uv = newUVs.ToArray ();
        objectMesh.triangles = oldTriangles;

        // Initialize old positions of clipping planes
        InitializePlanePositions();

    }

    public void InitializePlanePositions() {
        for (int i = 0; i < oldPlanePositions.Length; i++) {
            oldPlanePositions[i] = Vector3.positiveInfinity;
        }

    }


    void InitializeMeshData(Vector3 planeNormal, Vector3 planePosition){
        // Get Object to Clip Mesh

        objectToClip = GameObject.FindGameObjectWithTag ("ScreenCenter");
        objectMeshFilter = objectToClip.GetComponent<MeshFilter> ();
        if (objectMeshFilter == null) {
            objectMeshFil-
ter = objectToClip.GetComponentInChildren<MeshFilter>();
        }
        objectMesh = objectMeshFilter.mesh;

        // Get Mesh info

        oldVertices = objectMesh.vertices;
        oldTriangles = objectMesh.triangles;
        oldNormals = objectMesh.normals;

        //Clear Lists that will store the final Mesh Info after Clipping

        newVertices.Clear ();
        newTriangles.Clear ();
        newNormals.Clear ();

        if (intersectionEdges != null) {
            intersectionEdges.Clear ();
        } else {
            intersectionEdges = new List<IntersectionEdge> ();
```

```
        }

        loops.Clear ();

        // Initialize Array
        // First Row -
 int Array storing info about which Vertex has already
        // been added to the new Vertex List, thus avoiding Doubles  (-
1 -> not added , 1-> added);
        // Second row -
 specifies the position of the Vertex in the new Vertex List
        vertexInfo = new int[oldVertices.Length, 2];

        for (int i = 0; i<vertexInfo.GetLength (0); i++) {
            vertexInfo [i,0] = -1;
            vertexInfo [i,1] = 0;
        }
        // new Vertice list index
        listIndex = 0;

        // Initialize the Clipping Plane
        clipPlane = new Plane(planeNormal,planePosition);

    }

    public float ClipMesh (Vector3 planeNormal,Vector3 planePosition){

        // Check if planePosition is the same as the previous clipping op
eration. If so, no clipping is necessary.
        if (planeNormal == Vector3.left && (oldPlanePositions [0] ==
Vector3.positiveInfinity || oldPlanePositions [0] != planePosition)) {
            oldPlanePositions [0] = planePosition;
        } else if (planeNormal == Vector3.right && (oldPlanePositions [1]
 == Vector3.positiveInfinity || oldPlanePositions [1] != planePosition))
{
            oldPlanePositions [1] = planePosition;
        } else if (planeNormal == Vector3.down && (oldPlanePositions [2]
== Vector3.positiveInfinity || oldPlanePositions [2] != planePosition)) {
            oldPlanePositions [2] = planePosition;
        } else if (planeNormal == Vector3.up && (oldPlanePositions [3] ==
 Vector3.positiveInfinity || oldPlanePositions [3] != planePosition)) {
            oldPlanePositions [3] = planePosition;
        } else if (planeNormal == Vector3.back && (oldPlanePositions [4]
== Vector3.positiveInfinity || oldPlanePositions [4] != planePosition)) {
            oldPlanePositions [4] = planePosition;
        } else if (planeNormal == Vector3.forward && (oldPlanePositions [
5] == Vector3.positiveInfinity || oldPlanePositions [5] != planePosition)
) {
            oldPlanePositions [5] = planePosition;
        } else {
            De-
bug.Log ("Same plane position .. No Clipping is necessary!");
            return -1;
        }


        InitializeMeshData (planeNormal,planePosition);
        // Arrays storing indices of intersecting Triangle Vertices

        List<int> positive = new List<int> ();
        List<int> negative = new List<int> ();

        // the 3 points of each triangle
        int[] vertexIndices = new int[3];
        Vector3[] points = new Vector3[3];
        Vector3[] pointsRounded = new Vector3[3];
        float[] pointDistances = new float[3];

        //check position of each triangle in relation to the clipPlane

        for (int i = 0; i < oldTriangles.Length; i += 3) {

            //get Point Indices of current Triangle

            vertexIndices[0] = oldTriangles[i];
            vertexIndices[1] = oldTriangles[i+1];
            vertexIndices[2] = oldTriangles[i+2];

            //get Points of current Triangle
            points[0] = oldVertices[oldTriangles[i]];
            points[1] = oldVertices[oldTriangles[i+1]];
            points[2] = oldVertices[oldTriangles[i+2]];

            // keep only 5 frictional digits of new Vectors, resulting in
 correct float comparison with zero
            pointsRounded [0].Set ((float)Math.Round (points [0].x, 5),
(float)Math.Round (points [0].y, 5), (float)Math.Round (points [0].z, 5))
;
            //pointsGlobal [1] = objectToClip.transform.TransformPoint
(points [1]);
            pointsRounded [1].Set ((float)Math.Round (points [1].x, 5),
(float)Math.Round (points [1].y, 5), (float)Math.Round (points [1].z, 5))
;
            //pointsGlobal [2] = objectToClip.transform.TransformPoint
(points [2]);
            pointsRounded [2].Set ((float)Math.Round (points [2].x, 5),
(float)Math.Round (points [2].y, 5), (float)Math.Round (points [2].z, 5))
;

            // Calculate Distance from point to clipPlane (distance=0
if a point lies on the plane)
            pointDistances [0] = clipPlane.GetDistanceToPoint
(pointsRounded [0]);
            pointDistances [1] = clipPlane.GetDistanceToPoint
(pointsRounded [1]);
            pointDistances [2] = clipPlane.GetDistanceToPoint
(pointsRounded [2]);

            // Case 1 - whole triangle on positive side of the plane
            // Also handles special cases when 1 or 2 points of the
triangle lie on the plane

            if (pointDistances [0] >= 0f && pointDistances [1] >= 0f &&
pointDistances [2] >= 0f) {

                // if Triangle is on positive Side -
> Retain Triangle ( Add Triangle to new Triangle List)

                // if whole triangle lies on the clip plane then there is
 no need to add any intersection edges
                if (pointDistances [0] == 0f && pointDistances [1] ==
0f && pointDistances [2] == 0f) {
                    continue;
                }

                // If two points lie on the clip plane then Add
Intersection Edge to list
                if (pointDistances [0] == 0f && pointDistances [1] == 0f)
 {
                    intersectionEdges.Add (new IntersectionEdge (
points [0], points [1]));
                }
                else if (pointDistances [0] == 0f && pointDistances [2]
== 0f) {
                    intersectionEdg-
es.Add(new IntersectionEdge(points[2],points[0]));
                }
                else if (pointDistances [1] == 0f && pointDistances [2] =
= 0f) {
                    intersectionEdg-
es.Add(new IntersectionEdge(points[1],points[2]));
                }

                AddNewTriangle (vertexIndices);

            }
            // Triangle is on the negative Side -
> Discard Triangle (Do not add to new Triangle List)
            else if (pointDistances [0] <= 0f && pointDistances [1] <=
0f && pointDistances [2] <= 0f) {
                continue;
            }
             //Triangle intersects with the Plane
            else {

                positive.Clear ();
                negative.Clear ();

                // Find in which halfspace each Triangle vertex belongs
                for (int j = 0; j < 3; j++) {
                    if (pointDistances [j] > 0f) {
                        positive.Add(vertexIndices [j]);
                    } else if (pointDistances [j] <= 0f) {
                        negative.Add(vertexIndices [j]);
                    }
                }

                // Adjust vertice lists in CCW order
                if (positive.Count == 1 && pointDistances [1] > 0) {
                    negative.Reverse ();
                }
                else if(negative.Count==1 && pointDistances [1] > 0){
                    positive.Reverse ();
                }

                AddIntersectingTriangle (positive.ToArray(),negative.
ToArray());

            }
        }

        ConstructLoops ();


        List<Vector3> newVerticesOut;
        List<Vector3> newNormalsOut;
        List<int> newTrianglesOut;

        Triangulator tr = GetComponent<Triangulator> ();

        float objectSurface = 0;

        //First Find every polygon that contains holes and triangulate it
        foreach (Loop polygon in loops) {

            if (polygon.IsTriangulated ()) {
                continue;
            }

            List<Loop> polygonHoles = new List<Loop> ();

            foreach (Loop polygon2 in loops) {

                if (System.Object.ReferenceEquals (polygon, polygon2)) {
                    continue;
                }

                if (IsPointInPolygon (polygon, polygon2.GetEdge (0).
GetStartVertex (),planeNormal)) {
                    polygon2.SetIsTriangulated (true);
                    polygonHoles.Add (polygon2);
                }

            }

            if (polygonHoles.Count > 0) {

                polygon.SetIsTriangulated (true);
                objectSurface += tr.PartitionIntoMonotonePieces (
polygon.GetEdges(),out newVerticesOut,out newNormalsOut, out
newTrianglesOut,polygonHoles, ref listIndex, planeNormal);

                newVertices.AddRange (newVerticesOut);
                newNormals.AddRange (newNormalsOut);
                newTriangles.AddRange (newTrianglesOut);

            }
        }


        foreach (Loop polygon in loops) {

            if (!polygon.IsTriangulated ()) {
                polygon.SetIsTriangulated (true);

                objectSur-
face += tr.PartitionIntoMonotonePieces (polygon.GetEdges(),out newVertice
sOut,out newNormalsOut, out newTrianglesOut,
null, ref listIndex, planeNormal);

                newVertices.AddRange (newVerticesOut);
                newNormals.AddRange (newNormalsOut);
```

```
                newTriangles.AddRange (newTrianglesOut);
            }
        }

        ConstructFinalMesh ();

        return objectSurface;
    }

// Function tha checks if a given  point lies inside a given polygon
// The point is given in 3D space and is converted to 2D space
    bool IsPointInPolygon(Loop polygon, Vector3 point3D, Vector3
planeNormal){

        List<IntersectionEdge> polygonEdges = polygon.GetEdges();
        bool oddIntersections = false;

        Quaternion rotation;
        if (planeNormal == Vector3.forward) {
            rotation = Quaternion.identity;
        } else {
            rotation = Quaternion.FromToRotation (planeNormal, Vector3.
forward);
        }

        Vector2 point = (Vector2)(rotation * point3D);

        foreach (IntersectionEdge e in polygonEdges) {

            Vec-
tor2 endPoint1 = (Vector2)(rotation * e.GetStartVertex ());
            Vector2 endPoint2 = (Vector2)(rotation * e.GetEndVertex ());

            if     (  (endPoint1.y<point.y && endPoint2.y>=point.y)
                || (endPoint2.y<point.y && endPoint1.y>=point.y) ){

                Vector2 direction = endPoint2 - endPoint1;
                float xDifference = direction.x;
                float slope=0;
                if (xDifference!=0){
                    slope = direction.y / direction.x ;
                }
                float intercept = endPoint1.y - slope*endPoint1.x;

                if (   (xDifference==0 && endPoint1.x < point.x )
                    || (slope > 0 && point.y < (slope * point.x +
intercept))
                    || (slope < 0 && point.y > (slope * point.x +
intercept))) {

                    oddIntersections = !oddIntersections;
                }
            }
        }

        return oddIntersections;
    }

    void ConstructLoops(){

        while (intersectionEdges.Count != 0) {
            Loop newLoop = new Loop ();
            Vector3 loopStartVertex;
            IntersectionEdge currentEdge = intersectionEdges [0];
            newLoop.AddEdge (currentEdge);
            intersectionEdges.RemoveAt (0);
            loopStartVertex = newLoop.GetFirstEdge ().GetStartVertex ();


// Equality comparison accounts for floating point
// inaccuracies..
            while (currentEdge.GetEndVertex () != loopStartVertex) {

                IntersectionEdge successorOfCurrent=null;
                int successorPosition = -1;

                for (int i=0; i<intersectionEdges.Count;i++){

                    if (currentEdge.GetEndVertex() ==
intersectionEdges[i].GetStartVertex()){

                        successorOfCurrent=intersectionEdges[i];
                        successorPosition = i;
                        break;
                    }
                }

                if (successorOfCurrent != null) {
                    currentEdge = successorOfCurrent;
                    newLoop.AddEdge (currentEdge);
                    intersectionEdges.RemoveAt (successorPosition);

                } else {
                    Debug.LogError ("ERROR: NO SUCCESOR FOUND!!");
                    break;
                }
            }

            loops.Add (newLoop);
        }
    }

    void AddIntersectingTriangle (int[] positive, int[] negative) {

        float ray1Distance = 0;
        float ray2Distance = 0;
        float ray1DistanceNormalized = 0;
        float ray2DistanceNormalized = 0;

        // Vertice positions at Local Space
        Vector3 positiveLocal1;
        Vector3 positiveLocal2;
        Vector3 negativeLocal1;
        Vector3 negativeLocal2;

        // First intersection point
        Vector3 newVertex1;
        Vector3 newNormal1;

        // Second intersection point
        Vector3 newVertex2;
        Vector3 newNormal2;

        if (positive.Length == 1) {

            // Vertice positions at Local Space
            positiveLocal1 = oldVertices[positive[0]];
            negativeLocal1 = oldVertices[negative[0]];
            negativeLocal2 =  oldVertices[negative[1]];

            // Ray (line) going from point at the positive side to the
first point at the negative side of the plane
            Ray ray = new Ray (positiveLocal1, (negativeLocal1 -
 positiveLocal1).normalized);

            clipPlane.Raycast (ray, out ray1Distance);
            ray1DistanceNormalized = ray1Distance / (negativeLocal1 -
 positiveLocal1).magnitude;

            //first intersection point with plane
            newVertex1 = Vector3.Lerp (positiveLocal1, negativeLocal1,
ray1DistanceNormalized);
            newNormal1 = Vector3.Lerp (oldNormals[positive[0]],
oldNormals[negative[0]], ray1DistanceNormalized);

            // Ray (line) going from point at the positive side to the
second point at the negative side of the plane
            Ray ray2 = new Ray (positiveLocal1, (negativeLocal2 -
 positiveLocal1).normalized);
            clipPlane.Raycast (ray2, out ray2Distance);
            ray2DistanceNormalized = ray2Distance / (negativeLocal2 -
 positiveLocal1).magnitude;

            //second intersection point with plane
            newVertex2 = Vector3.Lerp (positiveLocal1, negativeLocal2,
ray2DistanceNormalized);
            newNormal2 = Vector3.Lerp (oldNormals[positive[0]],
oldNormals[negative[1]], ray2DistanceNormalized);

            // Add new Triangle
            // First Vertex
            CheckAndAddOldVertex (positive [0]);

            // Second Vertex
            if (ray1DistanceNormalized == 1) {
                CheckAndAddOldVertex (negative [0]);

            } else {
                newVertices.Add (newVertex1);
                newNormals.Add (newNormal1);
                //newUVs.Add (newUV1);
                newTriangles.Add (listIndex);
                listIndex++;
            }

            // Third Vertex
            if (ray2DistanceNormalized == 1) {
                CheckAndAddOldVertex (negative [1]);

            } else {
                newVertices.Add (newVertex2);
                newNormals.Add (newNormal2);
                //newUVs.Add (newUV2);
                newTriangles.Add (listIndex);
                listIndex++;
            }

            intersectionEdges.Add(new IntersectionEdge(newVertex1,
newVertex2));

        } else {

            // Vertice positions at Local Space
            positiveLocal1 = oldVertices[positive[0]];
            positiveLocal2 = oldVertices[positive[1]];
            negativeLocal1 =  oldVertices[negative[0]];

            // Ray (line) going from the first point at the positive side
 to the point at the negative side of the plane
            Ray ray = new Ray (positiveLocal1, (negativeLocal1 -
 positiveLocal1).normalized);

            clipPlane.Raycast (ray, out ray1Distance);
            ray1DistanceNormalized = ray1Distance / (negativeLocal1 -
 positiveLocal1).magnitude;

            //first intersection point with plane
            newVer-
tex1 = Vector3.Lerp (positiveLocal1, negativeLocal1, ray1DistanceNormaliz
ed);
            newNor-
mal1 = Vector3.Lerp (oldNormals[positive[0]], oldNormals[negative[0]], ra
y1DistanceNormalized);

            // Ray (line) going from the second point at the positive sid
e to the point at the negative side of the plane
            Ray ray2 = new Ray (positiveLocal2, (negativeLocal1 -
 positiveLocal2).normalized);
            clipPlane.Raycast (ray2, out ray2Distance);
            ray2DistanceNormalized = ray2Distance / (negativeLocal1 -
 positiveLocal2).magnitude;

            //second intersection point with plane
            newVer-
tex2 = Vector3.Lerp (positiveLocal2, negativeLocal1, ray2DistanceNormaliz
ed);
            newNor-
mal2 = Vector3.Lerp (oldNormals[positive[1]], oldNormals[negative[0]], ra
y2DistanceNormalized);

            //Create First Triangle
            CheckAndAddOldVertex (positive [0]);

            newVertices.Add (newVertex1);
            newNormals.Add (newNormal1);
            newTriangles.Add (listIndex);
            listIndex++;

            newVertices.Add (newVertex2);
            newNormals.Add (newNormal2);
            newTriangles.Add (listIndex);
            listIndex++;
```

```
            //Create Second Triangle
            newTriangles.Add (vertexInfo [positive[0], 1]);
            newTriangles.Add (listIndex-1);

            CheckAndAddOldVertex (positive [1]);

            intersectionEdg-
es.Add(new IntersectionEdge(newVertex1,newVertex2));

        }
    }

    // Function Adding a Triangle which  all of its vertices lie in the p
ositive halfspace
    void AddNewTriangle(int[] vertexIndices){

        int index;

        for (int i = 0; i < 3; i++) {
            index = vertexIndices [i];
            CheckAndAddOldVertex (index);

        }

    }

    // Function that checks if a given vertex has already been added to n
ewVer-
tex List and sets a new triangle index. If not, the vertex is added to th
e list and the triangle index points at its position.
    void CheckAndAddOldVertex(int vertexIndex){
        if (vertexInfo [vertexIndex, 0] != 1) {
            newVertices.Add (oldVertices[vertexIndex]);
            newNormals.Add (oldNormals [vertexIndex]);
            newTriangles.Add (listIndex);

            vertexInfo [vertexIndex, 0] = 1;
            vertexInfo [vertexIndex, 1] = listIndex;

            listIndex++;
        } else {
            newTriangles.Add (vertexInfo [vertexIndex, 1]);

        }
    }

    void ConstructFinalMesh(){

        objectMesh.Clear ();

        if (newTriangles.Count % 3 == 0) {
            objectMesh.vertices = newVertices.ToArray ();
            objectMesh.normals = newNormals.ToArray ();
            objectMesh.triangles = newTriangles.ToArray ();
        } else {
            objectMesh.vertices = oldVertices;
            objectMesh.normals = oldNormals;
            objectMesh.triangles = oldTriangles;

        }
    }
}
```

Triangulator

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;

public class Triangulator : MonoBehaviour {



    // Function that partitions a polygon in monotone polygons and then T
riangu-
lates it. Output is list of Vectors, a list of normals and a list of tria
ngles for the output mesh.

    public float PartitionIntoMonotonePieces(List<IntersectionEdge>
polygonEdges, out List<Vector3> newVerticesOut, out List<Vector3>
newNormalsOut, out List<int> newTrianglesOut,List<Loop> holes, ref int
offSet, Vector3 planeNormal){

        List<int> polygonStartEndIndices;

        DCEL dcel = ConstructDCEL(polygonEdges,holes,out polygonStartEndI
ndices,planeNormal);

        // Sort Vertices in descending order according to their y coordin
ates
        dcel.SortVertices();
        List<Vertex> dcelVertices = dcel.GetVertices ();
        List<HalfEdge> candidateLeftEdges = new List<HalfEdge> ();

        Vertex currentVertex;

        int i = 0;
        while (i<dcelVertices.Count) {

            currentVertex = dcelVertices [i];

            i++;
            switch (FindVertexType (currentVertex)) {

                case 0:
                HandleRegularVertex (currentVertex, candidateLeftEdges,
polygonStartEndIndices, dcel);
                    break;

                case 1:
                HandleStartVertex (currentVertex, candidateLeftEdges);
                    break;

                case 2:
                HandleEndVertex (currentVertex, candidateLeftEdges,
dcel);
                    break;

                case 3:
                HandleSplitVertex (currentVertex, candidateLeftEdges,
dcel);
                    break;

                case 4:
                HandleMergeVertex (currentVertex, candidateLeftEdges,
dcel);
                    break;

            }

        }

        List<HalfEdge> hEdges = dcel.GetHalfEdges();


        // Find every y-
Mono-
tone polygon that was created and triangulate it. There may be only 1 if
the original Polygon was already y-monotone.

        DCEL monotoneDCEL = new DCEL();

        int indexOffset = offSet;
        //int indexOffset=0;
        int listIndex=0;
        List<Vector3> newVertices = new List<Vector3> ();
        List<Vector3> newNormals = new List<Vector3> ();
        List<int> newTriangles = new List<int> ();

        foreach (HalfEdge e in hEdges){

            if (e.HasBeenTraversed ()) {
                continue;
            }
            // New y-monotone polygon
            else {

                HalfEdge polygonEdge = e;
                Vector2 startPosition = polygonEdge.GetOrigin ().
GetPosition ();
                Color randomColor = UnityEngine.Random.ColorHSV ();
                // Build Monotone Polygon by traversing every edge of it
and construct its DCEL
                do{
                    monotoneDCEL.AddHalfEdge(polygonEdge);
                    monotoneDCEL.AddVertex(polygonEdge.GetOrigin());

                    //Edge has been traversed
                    polygonEdge.SetHasBeenTraversed(true);
                    //Correct origin's incident edge
                    polygonEdge.GetOrigin().SetIncidentEdge(polygonEdge);
                    //Initialize origin being on the right Chain of the
polygon
                    polygonEdge.GetOrigin().SetIsOnRightChain(true);

                    polygonEdge = polygonEdge.GetNext();

                }while(polygonEdge.GetOrigin().GetPosition()!=
startPosition);

                // If y-
Mono-
tone Polygon has only 3 Vertices, its already a triangle. No need to tria
ngulate. Just update vertice and triangle lists of exported Mesh
                if (monotoneDCEL.GetVertices ().Count == 3) {

                    for (int j = 2; j >= 0; j--) {
                        Vertex vertex = monotoneDCEL.GetVertex (j);

                        if (vertex.GetListIndex () == -1) {

                            newVertices.Add (dcel.GetVertex3dPosition(
vertex.GetInitialPositionInList ()));
                            newNormals.Add (-planeNormal);
                            vertex.SetListIndex (listIndex);

                            newTriangles.Add (listIndex + indexOffset);
                            listIndex++;

                        } else {
                            newTriangles.Add (vertex.GetListIndex () +
indexOffset);
                        }
                    }

                } else {
                    // Call function that triangulates the y-
monotone polygon
                    TriangulateMono-
tone (monotoneDCEL,dcel, newVertices, newNormals, newTriangles,planeNorma
l, indexOffset, ref listIndex);
                }

                monotoneDCEL.Clear ();
            }
        }

        Calculator calculator = this.GetComponent <Calculator>();
        float capSurface = calculator.CalculateSurface (newTriangles.
ToArray(), newVertices.ToArray(),indexOffset);


        //update Offset of listIndex
        offSet += listIndex;

        //Assign out Variables
        newVerticesOut = newVertices;
        newTrianglesOut = newTriangles;
        newNormalsOut = newNormals;

        // Clear Doubly Connected Edge List
        dcel.Clear ();

        return capSurface;
    }

    // Function that constructs a Doubly Connected Edge List given a list
 of polygon edges
    // If there are holes in the polygon they are also added to the DCEL
```

```
    DCEL ConstructDCEL(List<IntersectionEdge> edges, List<Loop> holes,
out List<int> polygonStartEndIndicesOut, Vector3 planeNormal){

        DCEL dcel = new DCEL ();
        Face polygonFace = new Face ();
        List<int> polygonStartEndIndices = new List<int> ();

        Quaternion rotation;
        if (planeNormal == Vector3.forward) {
            rotation = Quaternion.identity;
        } else {
            rotation = Quaternion.FromToRotation (planeNormal,
Vector3.forward);
        }

        dcel.AddFace (polygonFace);

        AddPolygonToDCEL(edges, dcel, polygonStartEndIndices, rotation,
false);

        // Add holes to DCEL, if there are any
        if (holes!=null) {
            for (int i = 0; i < holes.Count; i++) {
                AddPolygonToDCEL (holes [i].GetEdges (), dcel,
polygonStartEndIndices,rotation, true);
            }
        }

        polygonStartEndIndicesOut = polygonStartEndIndices;
        return dcel;
    }

    void AddPolygonToDCEL(List<IntersectionEdge> edges, DCEL dcel,
List<int> polygonStartEndIndices,Quaternion rotation, bool isHole){

        Face polygonFace = dcel.GetFace (0);
        Face unboundedFace = dcel.GetUnboundedFace ();

        // Add first polygon Edge to DCEL
        IntersectionEdge e = edges [0];

        Vertex v1 = new Vertex ((Vector2)(rotation*e.GetStartVertex()));
        Vertex v2 = new Vertex ((Vector2)(rotation*e.GetEndVertex()));
        HalfEdge he1 = new HalfEdge ();
        HalfEdge he2 = new HalfEdge ();

        int vertexCount=dcel.GetVertices().Count;
        int halfEdgeCount=dcel.GetHalfEdges().Count;

        // Variables indicating the position of first Vertex and halfEdge
 of current polygon in the DCEL
        int startVertexIndex = vertexCount;
        int firstHalfEdgeIndex = halfEdgeCount;

        //Add start index of current polygon to list as well as the end
index of previous polygon added to DCEL
        if (startVertexIndex != 0) {
            polygonStartEndIndices.Add (startVertexIndex);
            polygonStartEndIndices.Add (startVertexIndex - 1);
        }

        v1.SetIncidentEdge (he1);
        v1.SetInitialPositionInList (vertexCount);
        v2.SetIncidentEdge (he2);
        v2.SetInitialPositionInList (vertexCount+1);

        he1.SetOrigin (v1);
        he1.SetTwin (he2);
        he1.SetIncidentFace (polygonFace);

        he2.SetOrigin (v2);
        he2.SetTwin (he1);
        he2.SetIncidentFace (unboundedFace);
        he2.SetHasBeenTraversed (true);

        if (isHole) {
            //If polygon constitutes a hole then add first halfEdge as
an inner component of polygonFace
            polygonFace.AddInnerComponent (he1);
        } else {
            unboundedFace.SetOuterComponent (he2);
            polygonFace.SetOuterComponent (he1);
        }

        dcel.AddVertex (v1);
        dcel.AddVertex3Dposition (e.GetStartVertex());
        dcel.AddVertex (v2);
        dcel.AddVertex3Dposition (e.GetEndVertex());
        dcel.AddHalfEdge (he1);
        dcel.AddHalfEdge (he2);

        vertexCount += 2;
        halfEdgeCount += 2;

        HalfEdge prevhe1;
        HalfEdge prevhe2;

        // Add the rest polygon edges to DCEL except the last one
        for (int i = 1; i < edges.Count-1; i++) {
            e = edges [i];

            v2 = new Vertex ((Vector2)(rotation*e.GetEndVertex()));
            he1 = new HalfEdge ();
            he2 = new HalfEdge ();

            v2.SetInitialPositionInList (vertexCount);

            prevhe1 = dcel.GetHalfEdge (halfEdgeCount - 2);
            prevhe2 = dcel.GetHalfEdge (halfEdgeCount - 1);

            dcel.GetVertex (vertexCount - 1).SetIncidentEdge (he1);

            he1.SetOrigin (dcel.GetVertex(vertexCount-1));
            he1.SetTwin (he2);
            he1.SetIncidentFace (polygonFace);
            he1.SetPrev (prevhe1);

            he2.SetOrigin (v2);
            he2.SetTwin (he1);
            he2.SetIncidentFace (unboundedFace);
            he2.SetHasBeenTraversed (true);
            he2.SetNext (prevhe2);

            prevhe1.SetNext (he1);
            prevhe2.SetPrev (he2);

            dcel.AddVertex (v2);
            dcel.AddVertex3Dposition (e.GetEndVertex());
            dcel.AddHalfEdge (he1);
            dcel.AddHalfEdge (he2);

            vertexCount++;
            halfEdgeCount += 2;

        }

        // Add last polygon edge to DCEL and make final changes to pointe
rs of the first 2 HalfEdges

        he1 = new HalfEdge ();
        he2 = new HalfEdge ();

        prevhe1 = dcel.GetHalfEdge (halfEdgeCount - 2);
        prevhe2 = dcel.GetHalfEdge (halfEdgeCount - 1);

        dcel.GetVertex (vertexCount - 1).SetIncidentEdge (he1);

        he1.SetOrigin (dcel.GetVertex(vertexCount-1));
        he1.SetTwin (he2);
        he1.SetIncidentFace (polygonFace);
        he1.SetPrev (prevhe1);
        he1.SetNext (dcel.GetHalfEdge (firstHalfEdgeIndex));

        he2.SetOrigin (dcel.GetVertex(startVertexIndex));
        he2.SetTwin (he1);
        he2.SetIncidentFace (unboundedFace);
        he2.SetHasBeenTraversed (true);
        he2.SetNext (prevhe2);
        he2.SetPrev (dcel.GetHalfEdge (firstHalfEdgeIndex+1));

        prevhe1.SetNext (he1);
        prevhe2.SetPrev (he2);

        dcel.GetHalfEdge (firstHalfEdgeIndex).SetPrev (he1);
        dcel.GetHalfEdge (firstHalfEdgeIndex+1).SetNext (he2);

        dcel.AddHalfEdge (he1);
        dcel.AddHalfEdge (he2);

    }


    void HandleRegularVertex(Vertex vertex, List<HalfEdge>
candidateLeftEdges,List<int> polygonStartEndIndices, DCEL dcel ){
        bool interiorRight=false;
        int vertexIndex = vertex.GetInitialPositionInList ();

        // First find if interior of Polygon lies to the left or to the
right of vertex
        // This is Found by checking the indices of its upper and lower
neighbors
        Vertex nextNeighbor = vertex.GetIncidentEdge ().GetNext ().
GetOrigin ();
        Vertex prevNeighbor = vertex.GetIncidentEdge ().GetPrev ().
GetOrigin ();

        Vector2 nextNeighborPos = nextNeighbor.GetPosition();
        Vector2 prevNeighborPos = prevNeighbor.GetPosition();

        int upperNeighborIndex;
        int lowerNeighborIndex;

        if (Vector1_IsBelow_Vector2 (nextNeighborPos, prevNeighborPos)) {
            upperNeighborIn-
dex = prevNeighbor.GetInitialPositionInList ();
            lowerNeighborIn-
dex = nextNeighbor.GetInitialPositionInList ();
        } else {
            lowerNeighborIn-
dex = prevNeighbor.GetInitialPositionInList ();
            upperNeighborIn-
dex = nextNeighbor.GetInitialPositionInList ();
        }


        if (vertexIndex != 0 && vertexIndex != dcel.GetVertices().Count -
1 && !polygonStartEndIndices.Contains(vertexIndex)) {
            if (lowerNeighborIndex > upperNeighborIndex) {
                interiorRight = true;
            } else {
                interiorRight = false;
            }
        } else {
            if (lowerNeighborIndex > upperNeighborIndex) {
                interiorRight = false;
            } else {
                interiorRight = true;
            }
        }

        // If interior lies to the right
        if (interiorRight) {

            HalfEdge previousEdge = vertex.GetIncidentEdge ().GetPrev ();

            if (previousEdge.IsHelperMerge ()) {
                // Add 2 new HalfEdges which represent the new Diagonal
                // Set corresponding pointers of each halfEdge
                HalfEdge diagonalHalfEdge1;
                HalfEdge diagonalHalfEdge2;

                CreateDiagonal (vertex, previousEdge.GetHelper (), out
diagonalHalfEdge1, out diagonalHalfEdge2, false);

                dcel.AddHalfEdge (diagonalHalfEdge1);
                dcel.AddHalfEdge (diagonalHalfEdge2);
            }

            candidateLeftEdges.Remove (previousEdge);

            HalfEdge vertexEdge = vertex.GetIncidentEdge ();
```

```
            vertexEdge.SetHelper (vertex);
            vertexEdge.SetIsHelperMerge (false);

            candidateLeftEdges.Add (vertexEdge);


        }
        // If interior lies to the left
        else {

            // Binary search to find edge left of vi
            HalfEdge directlyLeftEdge=findDirectLeftEdge(
candidateLeftEdges,vertex);

            if (directlyLeftEdge == null) {
                return;
            }

            if (directlyLeftEdge.IsHelperMerge ()) {
                // Add 2 new HalfEdges which represent the new Diagonal
                // Set corresponding pointers of each halfEdge
                HalfEdge diagonalHalfEdge1;
                HalfEdge diagonalHalfEdge2;

                CreateDiago-
nal (vertex, directlyLeftEdge.GetHelper(), out diagonalHalfEdge1, out dia
gonalHalfEdge2, false);

                // In this Case we need to alter the Incident Edge of
Current Vertex, since this regular vertex may be connected with a
diagonal later as it becomes a helper
                // So the new Incident Edge becomes the first HalfEdge
created for this diagonal
                vertex.SetIncidentEdge(diagonalHalfEdge1);

                //Add halfEdges to DCEL
                dcel.AddHalfEdge (diagonalHalfEdge1);
                dcel.AddHalfEdge (diagonalHalfEdge2);

            }

            directlyLeftEdge.SetHelper (vertex);
            directlyLeftEdge.SetIsHelperMerge (false);

        }


    }

    void HandleStartVertex(Vertex vertex, List<HalfEdge>
candidateLeftEdges){

        HalfEdge vertexEdge = vertex.GetIncidentEdge ();
        vertexEdge.SetHelper (vertex);
        vertexEdge.SetIsHelperMerge (false);

        candidateLeftEdges.Add (vertexEdge);

    }

    void HandleEndVertex(Vertex vertex, List<HalfEdge> candidateLeftEdges
, DCEL dcel){
        HalfEdge previousEdge = vertex.GetIncidentEdge ().GetPrev ();

        if (previousEdge.IsHelperMerge ()) {

            // Add 2 new HalfEdges which represent the new Diagonal
            // Set corresponding pointers of each halfEdge
            HalfEdge diagonalHalfEdge1;
            HalfEdge diagonalHalfEdge2;

            CreateDiagonal (vertex, previousEdge.GetHelper (), out
diagonalHalfEdge1, out diagonalHalfEdge2, false);

            dcel.AddHalfEdge (diagonalHalfEdge1);
            dcel.AddHalfEdge (diagonalHalfEdge2);

        }
        candidateLeftEdges.Remove (previousEdge);
    }

    void HandleSplitVertex(Vertex vertex,List<HalfEdge>
candidateLeftEdges,DCEL dcel ){
        // Find Direct Left Edge of given Vertex
        HalfEdge directlyLeftEdge=findDirectLeftEdge(candidateLeftEdges,
vertex);

        if (directlyLeftEdge == null) {
            return;
        }
        // Add 2 new HalfEdges which represent the new Diagonal
        // Set corresponding pointers of each halfEdge
        HalfEdge diagonalHalfEdge1;
        HalfEdge diagonalHalfEdge2;

        CreateDiagonal (vertex, directlyLeftEdge.GetHelper(), out
diagonalHalfEdge1, out diagonalHalfEdge2, false);

        // In this Case we need to alter the Incident Edge of Current
Ver-
tex, since this split vertex may be connected with a diagonal later as it
 becomes a helper
        // So the new Incident Edge becomes the first HalfEdge created
for this diagonal


        dcel.AddHalfEdge (diagonalHalfEdge1);
        dcel.AddHalfEdge (diagonalHalfEdge2);

        directlyLeftEdge.SetHelper (vertex);
        directlyLeftEdge.SetIsHelperMerge (false);

        HalfEdge vertexEdge = vertex.GetIncidentEdge ();
        vertexEdge.SetHelper (vertex);
        vertexEdge.SetIsHelperMerge (false);

        vertex.SetIncidentEdge(diagonalHalfEdge1);

        candidateLeftEdges.Add (vertexEdge);

    }

    void HandleMergeVertex(Vertex vertex, List<HalfEdge>
candidateLeftEdges,DCEL dcel ){

        HalfEdge previousEdge = vertex.GetIncidentEdge ().GetPrev ();

        //Case of connecting this merge vertex with another merge vertex
lying to the right
        if (previousEdge.IsHelperMerge ()) {

            HalfEdge diagonalHalfEdge1;
            HalfEdge diagonalHalfEdge2;

            CreateDiagonal (vertex, previousEdge.GetHelper(), out
diagonalHalfEdge1, out diagonalHalfEdge2, false);

            dcel.AddHalfEdge (diagonalHalfEdge1);
            dcel.AddHalfEdge (diagonalHalfEdge2);

        }

        //Remove previous edge from candidate list
        candidateLeftEdges.Remove (previousEdge);
        // Find Edge directly left of vertex
        HalfEdge directlyLeftEdge=findDirectLeftEdge(candidateLeftEdges,
vertex);

        if (directlyLeftEdge == null) {
            return;
        }
        //Case of connecting this merge vertex with another merge vertex
lying to the left
        if (directlyLeftEdge.IsHelperMerge ()) {
            // Add 2 new HalfEdge which represent the new Diagonal
            // Set corresponding pointers of each halfEdge
            HalfEdge diagonalHalfEdge1;
            HalfEdge diagonalHalfEdge2;

            CreateDiagonal (vertex, directlyLeftEdge.GetHelper(), out
diagonalHalfEdge1, out diagonalHalfEdge2, false);

            // In this Case we need to alter the Incident Edge of Current
 Vertex, since this merge vertex will be connected with a new diagonal
later
            // So the new Incident Edge becomes the first HalfEdge
created for this diagonal
            vertex.SetIncidentEdge(diagonalHalfEdge1);

            //Add halfEdges to DCEL
            dcel.AddHalfEdge (diagonalHalfEdge1);
            dcel.AddHalfEdge (diagonalHalfEdge2);

        }
        directlyLeftEdge.SetHelper (vertex);
        directlyLeftEdge.SetIsHelperMerge (true);
    }

    // Function that creates a new diagonal formed by 2 new HalfEdges and
 sets corresponding pointers
    void CreateDiagonal(Vertex diagStart, Vertex diagEnd, out HalfEdge
diagonalHE1, out HalfEdge diagonalHE2, bool triangulationPhase ){

        HalfEdge diagonalHalfEdge1 = new HalfEdge ();
        HalfEdge diagonalHalfEdge2 = new HalfEdge ();

        diagonalHalfEdge1.SetOrigin (diagStart);
        diagonalHalfEdge1.SetTwin (diagonalHalfEdge2);
        diagonalHalfEdge1.SetPrev (diagStart.GetIncidentEdge ().GetPrev
());

        diagonalHalfEdge2.SetOrigin (diagEnd);
        diagonalHalfEdge2.SetTwin (diagonalHalfEdge1);
        diagonalHalfEdge2.SetNext (diagStart.GetIncidentEdge ());

        // Degenarate Case during the partitioning phase. If diagonal end
 is a split vertex and diagonal start is at the right of it,
        // then incidentEdge of diagonal End has been altered since a
nother diagonal has been added to that split vertex.
        // So pointers need to be handled differently
        if (triangulationPhase==false && diagEnd.GetVertexType () == 3 &&
 diagStart.GetPosition ().x > diagEnd.GetPosition ().x) {
            //Debug.Log ("SPECIAL CASE");
            diagonal-
HalfEdge1.SetNext (diagEnd.GetIncidentEdge ().GetTwin ().GetNext ());

            diagonal-
HalfEdge2.SetPrev (diagEnd.GetIncidentEdge ().GetTwin ());

            diagEnd.GetIncidentEdge ().GetTwin ().GetNext ().SetPrev
(diagonalHalfEdge1);
            diagEnd.GetIncidentEdge ().GetTwin ().SetNext
(diagonalHalfEdge2);

        } else {
            diagonalHalfEdge1.SetNext (diagEnd.GetIncidentEdge ());

            diagonal-
HalfEdge2.SetPrev (diagEnd.GetIncidentEdge ().GetPrev ());

            diagEnd.GetIncidentEdge ().GetPrev ().SetNext
(diagonalHalfEdge2);
            diagEnd.GetIncidentEdge ().SetPrev (diagonalHalfEdge1);

        }


        diagStart.GetIncidentEdge ().GetPrev ().SetNext
(diagonalHalfEdge1);
        diagStart.GetIncidentEdge ().SetPrev (diagonalHalfEdge2);

        diagonalHE1 = diagonalHalfEdge1;
        diagonalHE2 = diagonalHalfEdge2;
    }

    // Function that finds the edge directly left of given Vertex;
    HalfEdge findDirectLeftEdge(List<HalfEdge> candidateLeftEdges,
Vertex vertexObj){
        //Debug.Log ("Vertex type: " + vertexObj.GetVertexType());
        HalfEdge directlyLeftEdge=null;
        List<HalfEdge> candidates = new List<HalfEdge> ();
        Vector2 vertex = vertexObj.GetPosition ();
```

```
        float yValueStart;
        float yValueEnd;

        // First find all edges such as given vertex is between the edge
vertices and at the right of the edge : y of first vertex higher than y
of given vertex and y of second vertex is lower than y of given vertex
        for (int i = 0; i < candidateLeftEdges.Count; i++) {

            yValueStart = candidateLeftEdges [i].GetOrigin ().GetPosition
 ().y;
            yValueEnd = candidateLeftEdges [i].GetTwin ().GetOrigin ().
GetPosition().y;
//              zValueStart = candidateLeftEdges [i].GetOrigin ().
GetPosition ().z;
//              zValueEnd = candidateLeftEdges [i].GetTwin ().GetOrigin ().
GetPosition().z;

//              if ((yValueStart > vertex.y || Mathf.Approximately
(yValueStart, vertex.y)) && yValueEnd < vertex.y
//                  && (vertex.z > zValueStart || vertex.z > zValueEnd)) {
            if ((yValueStart > vertex.y || Mathf.Approximately
(yValueStart, vertex.y)) && yValueEnd < vertex.y     ) {

                Vector2 direction = candidateLeftEdges [i].GetTwin ().
GetOrigin ().GetPosition() -
 candidateLeftEdges [i].GetOrigin ().GetPosition ();
                float xDifference = direction.x;
                float slope=0;
                if (xDifference!=0){
                    slope = direction.y / direction.x ;
                }
                float intercept = candidateLeftEdges [i].GetOrigin ().
GetPosition ().y -
 slope*candidateLeftEdges [i].GetOrigin ().GetPosition ().x;

                if ((xDifference == 0 && candidateLeftEdges [i].
GetOrigin ().GetPosition ().x < vertex.x)
                    || (slope > 0 && vertex.y < (slope * vertex.x +
intercept))
                    || (slope < 0 && vertex.y > (slope * vertex.x +
intercept))) {

                    candidates.Add (candidateLeftEdges [i]);
                }
            }
        }

        // Finally find which of these edges is the rightmost one; This
should be the edge directly left of given vertex
        if (candidates.Count > 0) {
            directlyLeftEdge = candidates [0];
        } else {
            Debug.LogError ("No Direct left edge Found!!!! Vertex: "+
vertexObj.GetPosition());
            return null;

        }

        float zValueStart_leftEdge = directlyLeftEdge.GetOrigin().
GetPosition().x;
        float zValueEnd_leftEdge= directlyLeftEdge.GetTwin().GetOrigin().
GetPosition().x;

        float zValueStart_candidate;
        float zValueEnd_candidate;

        for (int i = 1; i < candidates.Count; i++) {

            zValueStart_candidate = candidates [i].GetOrigin().
GetPosition().x;
            zValueEnd_candidate= candidates [i].GetTwin().GetOrigin().
GetPosition().x;

            if (    (zValueStart_candidate>zValueStart_leftEdge || Mathf.
Approximately(zValueStart_leftEdge,zValueStart_candidate))
                && (zValueEnd_candidate>zValueEnd_leftEdge || Mathf.
Approximately(zValueEnd_leftEdge,zValueEnd_candidate)) ){

                directlyLeftEdge = candidates [i];
                zValueStart_leftEdge = zValueStart_candidate;
                zValueEnd_leftEdge = zValueEnd_candidate;
            }
        }

        return directlyLeftEdge;
    }

    // Function that finds the type of given vertex. 5 Possible Outcomes:
  0->Regular Vertex, 1-> Start Vertex, 2-> End Vertex,
3-> Split Vertex, 4-> Merge Vertex

    int FindVertexType(Vertex curVertex){

        Vector2 currentVertex = curVertex.GetPosition ();
        Vertex nextNeighbor = curVertex.GetIncidentEdge ().GetNext ().
GetOrigin ();
        Vertex prevNeighbor = curVertex.GetIncidentEdge ().GetPrev ().
GetOrigin ();
        int vertexType;
        // Find Vertex type depending on positions of neighboring
vertices.

        // If Both neighbors lie below the vertex, then vertex is either
a start or a split vertex
        if (Vector1_IsBelow_Vector2 (nextNeighbor.GetPosition(),
currentVertex) && Vector1_IsBelow_Vector2 (prevNeighbor.GetPosition(),
currentVertex)) {

            //Vector which has a direction from current Vertex to
previous Vertex;
            Vector2 prevDirectedVector = prevNeighbor.GetPosition () -
  curVertex.GetPosition ();
            //Vector which has a direction from current Vertex to next
Vertex;
            Vector2 nextDirectedVector = nextNeighbor.GetPosition () -
 curVertex.GetPosition ();

            // Find directed angle from nextDirectedVector to
prevDirectedVector (interior angle of polygon)
            float angle = Mathf.Atan2 (prevDirectedVector.y, prevDirected
Vector.x) - Mathf.Atan2 (nextDirectedVector.y, nextDirectedVector.x);
            if (angle < 0) {
                angle += 2 * Mathf.PI;
            }
            angle = angle * Mathf.Rad2Deg;

            if (angle < 180) {
                curVertex.SetVertexType (1);
                vertexType = 1;
            } else {
                curVertex.SetVertexType (3);
                vertexType = 3;
            }

        }
        // If Both neighbors lie above the vertex, then vertex is either
a merge or an end vertex

        else if (Vector1_IsAbove_Vector2 (nextNeighbor.GetPosition(),
currentVertex) && Vector1_IsAbove_Vector2 (prevNeighbor.GetPosition(),
currentVertex)) {

            //Vector which has a direction from current Vertex to
previous Vertex;
            Vector2 prevDirectedVector = prevNeighbor.GetPosition () -
  curVertex.GetPosition ();
            //Vector which has a direction from current Vertex to next
Vertex;
            Vector2 nextDirectedVector = nextNeighbor.GetPosition () -
 curVertex.GetPosition ();

            // Find directed angle from nextDirectedVector to
prevDirectedVector (interior angle of polygon)
            float angle = Mathf.Atan2 (prevDirectedVector.y,
prevDirectedVector.x) -
 Mathf.Atan2 (nextDirectedVector.y, nextDirectedVector.x);
            if (angle < 0) {
                angle += 2 * Mathf.PI;
            }
            angle = angle * Mathf.Rad2Deg;

            if (angle < 180) {
                curVertex.SetVertexType (2);
                vertexType = 2;
            } else {
                curVertex.SetVertexType (4);
                vertexType = 4;
            }
        }
        // Otherwise, vertex is regular
        else {
            curVertex.SetVertexType (0);
            vertexType = 0;
        }

        return vertexType;
    }

    // Implementation of a greedy Algorithm that Triangulates a
y-Monotone Polygon.
    void TriangulateMonotone(DCEL monotoneDcel, DCEL dcel, List<Vector3>
newVertices, List<Vector3> newNormals, List<int> newTriangles,Vector3
planeNormal, int indexOffset, ref int listIndex){

        //Sort vertices on decreasing y-coordinate.
        monotoneDcel.SortVertices ();
        List<Vertex> vertices = monotoneDcel.GetVertices ();

        // Find all Vertices which belong to the left chain of
y-monotone Polygon
        // Initially it is assumed that all Vertices belong to the right
chain (IsOnRightChain Variable of vertices is true).
        // Concurrently verify that input polygon is monotone
        Vector2 bottomVertexPosition = vertices [vertices.Count-
1].GetPosition ();
        Vertex currentVertex = vertices [0];
        do {
            if (currentVertex.GetPosition().y < currentVertex.
GetIncidentEdge ().GetNext ().GetOrigin ().GetPosition().y){

                Debug.LogError("Input Polygon is not Monotone!");
                return;
            }
            currentVertex = currentVertex.GetIncidentEdge ().GetNext ().
GetOrigin ();
            currentVertex.SetIsOnRightChain (false);

        } while(currentVertex.GetPosition () != bottomVertexPosition);

        //y-Monotone test for right Chain
        currentVertex  = vertices [0];
        do {
            if (currentVertex.GetPosition().y < currentVertex.
GetIncidentEdge ().GetPrev ().GetOrigin ().GetPosition().y){

                Debug.LogError("Input Polygon is not Monotone!");
                return;
            }
            currentVertex = currentVertex.GetIncidentEdge ().GetPrev ().
GetOrigin ();
        } while(currentVertex.GetPosition () != bottomVertexPosition);

        // Initialize Stack and push the first 2 vertices.
        Stack<Vertex> stack = new Stack<Vertex> ();

        if (vertices.Count < 2) {
            return;
        }

        stack.Push (vertices [0]);
        stack.Push (vertices [1]);

        Vertex stackVertex;
        Vertex lastPoppedStackVertex;
        for (int i = 2; i < vertices.Count - 1; i++) {
            currentVertex = vertices [i];
            stackVertex = stack.Peek ();

            // if current vertex and vertex on top of stack are on
different chains
            if (currentVertex.IsOnRightChain () != stackVertex.
```

```
IsOnRightChain ()) {

                bool firstIteration = true;
                HalfEdge firstDiagonalHE2=null;
                // Add a diagonal from currentVertex to each vertex in
the Stack, except the last one
                do {
                    stackVertex = stack.Pop ();

                    HalfEdge diagonalHalfEdge1;
                    HalfEdge diagonalHalfEdge2;

                    CreateDiagonal(currentVertex,stackVertex,out
diagonalHalfEdge1,out diagonalHalfEdge2, true);

                    if (!currentVertex.IsOnRightChain()){
                        currentVertex.SetIncidentEdge(diagonalHalfEdge1);
                    }

                    if (firstIteration){
                        firstIteration=false;
                        firstDiagonalHE2=diagonalHalfEdge2;
                    }

                    monotoneDcel.AddHalfEdge (diagonalHalfEdge1);
                    monotoneDcel.AddHalfEdge (diagonalHalfEdge2);

                } while(stack.Count != 1);

                // We need to alter the Incident Edge vertex lying on the
 right chain, since this vertex may be connected with a diagonal as it is
 pushed in the stack later

                // The new Incident Edge becomes the first diagonal added
 if current Vertex is on the right chain
                if (currentVertex.IsOnRightChain()){
                    currentVertex.SetIncidentEdge (vertices
[i-1].GetIncidentEdge().GetPrev());
                }
                // Or The new Incident Edge becomes the second HalfEdge
if stack Vertex is on the right chain
                else{
                    currentVertex.SetIncidentEdge (firstDiagonalHE2.
GetNext());
                    vertices [i - 1].SetIncidentEdge(firstDiagonalHE2);
                }

                // Pop last vertex from Stack
                stack.Pop ();
                // Push previous and current vertex to the Stack since
they may be connected with diagonals later
                stack.Push (vertices [i-1]);
                stack.Push (currentVertex);
            }
            // if current vertex and vertex on top of stack are on the
same chain
            else {
                lastPoppedStackVertex = stack.Pop ();

                if (currentVertex.IsOnRightChain ()) {

                    do {
                        if (stack.Count == 0) {
                            break;
                        }
                        stackVertex = stack.Peek ();

                        // Line from current vertex to vertex on top of
stack

                        Vector2 direction = stackVertex.GetPosition() -
 currentVertex.GetPosition();
                        float xDifference = direction.x;
                        float slope=0;
                        if (xDifference!=0){
                            slope = direction.y / direction.x ;
                        }
                        float intercept = currentVertex.GetPosition().y -
 slope*currentVertex.GetPosition().x;

                        // Check if a diagonal can be added to the vertex
 on Top of Stack.
                        // If a diagonal can be added then the last
popped vertex must be to the right of the line from currentVertex to the
vertex on top of stack
                        if (   (xDifference==0 && stackVertex.
GetPosition ().x < lastPoppedStackVertex.GetPosition ().x )
                            || (slope>0 &&  lastPoppedStackVertex.
GetPosition ().y < (slope*lastPoppedStackVertex.GetPosition ().x +
intercept))
                            || (slope<0 &&  lastPoppedStackVertex.
GetPosition ().y > (slope*lastPoppedStackVertex.GetPosition ().x +
intercept))    ){
                            // pop and insert diagonal

                            lastPoppedStackVertex = stack.Pop ();

                            HalfEdge diagonalHalfEdge1;
                            HalfEdge diagonalHalfEdge2;

                            CreateDiagonal (currentVertex, stackVertex,
out diagonalHalfEdge1, out diagonalHalfEdge2, true);

                            // We need to alter the Incident Edge vertex
lying on the right chain, since this vertex may be connected with a
diagonal as it is pushed in the stack later

                            // The new Incident Edge becomes the first
HalfEdge since current Vertex is on the right chain
                            currentVertex.SetIncidentEdge (
diagonalHalfEdge1);

                            monotoneDcel.AddHalfEdge (diagonalHalfEdge1);
                            monotoneDcel.AddHalfEdge (diagonalHalfEdge2);

                        } else {
                            break;
                        }

                    } while(true);

                } else {

                    do {
                        if (stack.Count == 0) {
                            break;
                        }
                        stackVertex = stack.Peek ();

                        // Line from current vertex to vertex on top of
stack

                        Vector2 direction = stackVertex.GetPosition() -
 currentVertex.GetPosition();
                        float xDifference = direction.x;
                        float slope=0;
                        if (xDifference!=0){
                            slope = direction.y / direction.x ;
                        }
                        float intercept = currentVertex.GetPosition().y -
 slope*currentVertex.GetPosition().x;

                        // Check if a diagonal can be added to the vertex
 on Top of Stack.
                        // If a diagonal can be added then the last
popped vertex must be to the left of the line from currentVertex to the
vertex on top of stack
                        if (   (xDifference==0 && stackVertex.GetPosition
 ().x > lastPoppedStackVertex.GetPosition ().x )
                            || (slope>0 &&  lastPoppedStackVertex.GetPosi
tion ().y > (slope*lastPoppedStackVertex.GetPosition ().x+intercept))
                            || (slope<0 &&  lastPoppedStackVertex.GetPosi
tion ().y < (slope*lastPoppedStackVertex.GetPosition ().x+intercept))
){
                            // pop and insert diagonal
                            lastPoppedStackVertex = stack.Pop ();

                            HalfEdge diagonalHalfEdge1;
                            HalfEdge diagonalHalfEdge2;

                            CreateDiago-
nal(currentVertex,stackVertex,out diagonalHalfEdge1,out diagonalHalfEdge2
, true);

                            // We need to alter the Incident Edge vertex
ly-
ing on the right chain, since this vertex may be connected with a diagona
l as it is pushed in the stack later

                            // The new Incident Edge becomes the first Ha
lfEdge since current Vertex is on the right chain
                            stackVertex.SetIncidentEdge
(diagonalHalfEdge2);

                            monotoneDcel.AddHalfEdge (diagonalHalfEdge1);
                            monotoneDcel.AddHalfEdge (diagonalHalfEdge2);

                        }
                        else{
                            break;
                        }

                    } while(true);
                }

                stack.Push (lastPoppedStackVertex);
                stack.Push (currentVertex);
            }
        }
        // Finally add a diagonal from the bottom vertex to every vertex
in the stack except the first and last one
        currentVertex = vertices [vertices.Count-1];

        //Pop first vertex
        stack.Pop ();

        //Add a diagonal until one vertex remains in the stack
        while(stack.Count != 1){
            stackVertex = stack.Pop ();

            HalfEdge diagonalHalfEdge1;
            HalfEdge diagonalHalfEdge2;

            CreateDiagonal(currentVertex,stackVertex,out
diagonalHalfEdge1,out diagonalHalfEdge2, true);

            // Alter incident edge of current vertex
            if (stackVertex.IsOnRightChain()){
                currentVertex.SetIncidentEdge(diagonalHalfEdge1);
            }

            monotoneDcel.AddHalfEdge (diagonalHalfEdge1);
            monotoneDcel.AddHalfEdge (diagonalHalfEdge2);

        }

        List<HalfEdge> hEdges = monotoneDcel.GetHalfEdges();

        // Build Triangle and Vertice arrays by traversing every triangle
 in CW order

        foreach (HalfEdge e in hEdges){

            if (e.HasBeenTraversed ()) {
                continue;
            }
            // New Triangle
            else {
                int iterations = 0;
                HalfEdge triangleEdge = e;
                Vector2 startPosition = triangleEdge.GetOrigin ().
GetPosition ();
                Color randomColor = UnityEngine.Random.ColorHSV ();
                // Traverse every edge of the triangle. Build Vertice and
 Triangle lists of Mesh
                do{
                    //Edge has been traversed
                    triangleEdge.SetHasBeenTraversed(true);

                    // Check if Origin Vertex is in the list of Vertices
of exported Mesh
```

```
                if (triangleEdge.GetOrigin().GetListIndex()==-1){

                    newVertices.Add(dcel.GetVertex3dPosition(
triangleEdge.GetOrigin().GetInitialPositionInList()));
                    newNormals.Add (-planeNormal);
                    triangleEdge.GetOrigin().SetListIndex(listIndex);

                    newTriangles.Add(listIndex + indexOffset);
                    iterations++;
                    listIndex++;

                }
                else{
                    newTriangles.Add(triangleEdge.GetOrigin().
GetListIndex() + indexOffset);
                    iterations++;
                }

                triangleEdge = triangleEdge.GetPrev();

            }while(triangleEdge.GetOrigin().GetPosition()!=
startPosition);

            De-
bug.Assert (iterations == 3,"Iterations = "+iterations);
            if (iterations != 3) {

                List<HalfEdge> hEdges3 = monotoneDcel.GetHalfEdges();

                foreach (HalfEdge e1 in hEdges3){
                    //Instantiate (instGO, e.GetOrigin ().GetPosition
 (), Quaternion.identity);

                    Debug.DrawLine (e1.GetOrigin().GetPosition(),
e1.GetTwin().GetOrigin().GetPosition(), Color.red, float.PositiveInfinity
);

                }
                break;
            }
        }
    }
}

bool Vector1_IsBelow_Vector2(Vector2 v1, Vector2 v2){

    if (Math.Round(v1.y,5) <Math.Round( v2.y,5) || (Mathf.Approximate
ly (v1.y, v2.y) && Math.Round(v1.x,5) > Math.Round(v2.x,5))) {
        return true;
    } else {
        return false;
    }
}

bool Vector1_IsAbove_Vector2(Vector2 v1, Vector2 v2){

    if (Math.Round(v1.y,5) >Math.Round( v2.y,5) || (Mathf.Approximate
ly (v1.y, v2.y) && Math.Round(v1.x,5) < Math.Round(v2.x,5)) ) {

        return true;
    } else {
        return false;
    }
}
}
```

Calculator

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using System;

public class Calculator : MonoBehaviour {

    MeshClip meshClip;
    InterfaceController interfaceController;

    public GameObject planeZ;
    public Text objectVolumeText;

    void Start(){
        meshClip = this.GetComponent<MeshClip>();
        interfaceController = this.GetComponent<InterfaceController>();
    }

    // Calculate Object's Surface
    public float CalculateSurface(GameObject obj){

        Mesh mesh = obj.GetComponent<MeshFilter> ().mesh;
        int[] triangles = mesh.triangles;
        Vector3[] vertices = mesh.vertices;

        float objectSurface = 0;
        //Debug.Log ("surface is: " + objectSurface + " m^2");
        for (int i = 0; i < triangles.Length; i += 3) {

            // Triangle points
            Vector3 a = vertices [triangles[i]];
            Vector3 b = vertices [triangles[i+1]];
            Vector3 c = vertices [triangles [i + 2]];

            float triangleSurface;
            // Distance from b to c
            float triangleBase = Vector3.Distance(b,c);

            //Calculate the distance from point a to the line from b to c
;
            // direction Vectors
            Vector3 ca = a - c;
            Vector3 cb = b - c;

            // calculate distance
            float triangleHeight = (Vector3.Cross (cb, ca).magnitude) /
cb.magnitude;
            triangleSurface = (triangleBase * triangleHeight) / 2;

            if (!float.IsNaN (triangleSurface)) {
                objectSurface += triangleSurface;
            }
        }
        return objectSurface;
    }

    public float CalculateSurface(int[] triangles, Vector3[] vertices,
int offset){

        float objectSurface = 0;
        //Debug.Log ("surface is: " + objectSurface + " m^2");
        if (triangles.Length % 3 != 0) {
            return 0;
        }

        for (int i = 0; i < triangles.Length; i += 3) {

            // Triangle points
            Vector3 a = vertices [triangles[i]-offset];
            Vector3 b = vertices [triangles[i+1]-offset];
            Vector3 c = vertices [triangles [i + 2]-offset];

            float triangleSurface;
            // Distance from b to c
            float triangleBase = Vector3.Distance(b,c);

            //Calculate the distance from point a to the line from b to c
;
            // direction Vectors
            Vector3 ca = a - c;
            Vector3 cb = b - c;

            // calculate distance
            float triangleHeight = (Vector3.Cross (cb, ca).magnitude) /
cb.magnitude;
            triangleSurface = (triangleBase * triangleHeight) / 2;

            if (!float.IsNaN (triangleSurface)) {
                objectSurface += triangleSurface;
            }
        }
        return objectSurface;
    }

    public float CalculateVolume(GameObject obj, int iterations){

        Mesh mesh = obj.GetComponent<MeshFilter> ().mesh;
        Bounds bounds = mesh.bounds;

        Vector3[] originalVertices = mesh.vertices;
        Vector3[] originalNormals = mesh.normals;
        int[] originalTriangles = mesh.triangles;

        float volumePartition = bounds.size.z / iterations;

        float lowerLimit = bounds.center.z -
 bounds.extents.z + volumePartition / 2;

        Vector3 clipPosition = Vector3.zero;

        float objectVolume=0;

        for (int i = 0; i <iterations; i++) {

            clipPosition.Set (bounds.center.x, bounds.center.y,
lowerLimit);
            lowerLimit += volumePartition;

            objectVolume += meshClip.ClipMesh (Vector3.forward,
clipPosition) * volumePartition;

            mesh.Clear ();
            mesh.vertices = originalVertices;
            mesh.normals = originalNormals;
            mesh.triangles = originalTriangles;

        }
        return objectVolume;
    }

    public void CalculateVolumeAnimate(GameObject obj, int iterations,
bool weightCalculation, float specificWeight){

        StartCoroutine (DoIterations (iterations, 1f, obj,
weightCalculation,specificWeight));

    }

    IEnumerator DoIterations(float iterations, float totalTime,
GameObject obj, bool weightCalculation, float specificWeight){

        interfaceController.SetCanCalculateVolume (false);

        Mesh mesh = obj.GetComponent<MeshFilter> ().mesh;
        Bounds bounds = mesh.bounds;

        Vector3[] originalVertices = mesh.vertices;
        Vector3[] originalNormals = mesh.normals;
        int[] originalTriangles = mesh.triangles;

        float volumePartition = bounds.size.z / iterations;

        float lowerLimit = bounds.center.z -
 bounds.extents.z + volumePartition / 2;

        Vector3 clipPosition = Vector3.zero;

        float objectVolume=0;

        if (weightCalculation) {
            objectVolumeText.text = objectVolume.ToString ();
        } else {
            objectVolume-
Text.text = objectVolume.ToString () + " mm\u00B3";
        }
        float iterationTime = totalTime / iterations;

        planeZ.SetActive (true);
        Vector3 boundsSize = bounds.size;
        float maxDimension = Mathf.Max (boundsSize.x, boundsSize.y,
```

```
boundsSize.z);
        int planeSize = Mathf.RoundToInt (maxDimension) + 1;
        planeZ.transform.localScale = new Vector3 (planeSize/ 100f,
planeSize, planeSize);

        for (int i = 0; i <iterations; i++) {

            clipPosition.Set (bounds.center.x, bounds.center.y,
lowerLimit);

            planeZ.transform.position = new Vector3 (bounds.center.x,
bounds.center.y, lowerLimit);
            lowerLimit += volumePartition;

            objectVolume += meshClip.ClipMesh (Vector3.forward,
clipPosition) * volumePartition;

            if (weightCalculation) {

                if (Math.Truncate (specificWeight * objectVolume /
1000000) == 0) {
                    objectVolumeText.text = Math.Round(specificWeight *
objectVolume / 1000000,3).ToString () + " g";
                } else {
                    objectVolumeText.text = Math.Truncate (
specificWeight * objectVolume / 1000000).ToString () + " g";
                }

            } else {

                if (Math.Truncate(objectVolume) == 0) {
                    objectVolumeText.text = Math.Round(objectVolume,3).
ToString () + " mm\u00B3";
                } else {
                    objectVolumeText.text = Math.Truncate(objectVolume).
ToString () + " mm\u00B3";
                }
            }

            mesh.Clear ();
            mesh.vertices = originalVertices;
            mesh.normals = originalNormals;
            mesh.triangles = originalTriangles;

            yield return new WaitForSeconds (iterationTime);
        }
        planeZ.SetActive (false);
        interfaceController.SetCanCalculateVolume (true);
    }
}
```

RadialProgressBar

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class RadialProgressBar : MonoBehaviour {

    public Image loadingBar;
    public Text progressText;
    public Text loadingText;

    public void UpdateProgress(float progress){

        loadingBar.fillAmount = progress;
        progressText.text = Mathf.RoundToInt (progress * 100f).ToString
() + "%";

        if (progress == 1) {
            loadingText.enabled = false;
            this.gameObject.SetActive (false);
        }
    }

    public void Activate(){

        this.gameObject.SetActive (true);
    }
}
```