

Ατομική Διπλωματική Εργασία

**ΑΝΙΧΝΕΥΣΗ ΦΩΤΕΙΝΗΣ ΠΗΓΗΣ ΓΙΑ ΦΩΤΙΣΜΟ
ΑΝΤΙΚΕΙΜΕΝΟΥ ΣΕ AUGMENTED REALITY**

Μιχάλης Γιάγκου

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ



ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2010

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ανίχνευση Φωτεινής Πηγής για Φωτισμό Αντικειμένου σε Augmented Reality

Μιχάλης Γιάγκου

Επιβλέπων Καθηγητής
Γιώργος Χρυσάνθου

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των
απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του
Πανεπιστημίου Κύπρου

Μάιος 2010

Περίληψη

Επαυξημένη Πραγματικότητα ή Augmented Reality (AR) ονομάζουμε την τεχνολογία που συνδυάζει τον τρισδιάστατο, πραγματικό κόσμο στον οποίο ζούμε, με τον ψηφιακό μέσω κάποιας συσκευής, δίνοντας τη δυνατότητα σε αυτή τη συσκευή να κάνει render αντικείμενα μέσα στο χώρο ή επιτρέποντας διάδραση μεταξύ χρήστη και του κόσμου μέσω της συσκευής.

Τον τελευταίο καιρό έχουν γίνει βήματα προς την ανάπτυξη τεχνολογιών που χρησιμοποιούν Augmented Reality από πολλές και ανάμεσά τους σημαντικές εταιρίες, όπως για παράδειγμα η Google. Ο λόγος για αυτή την τάση προς αυτή την πλευρά της τεχνολογίας είναι το ότι μέσω του Augmented Reality μπορούμε να προσφέρουμε στο χρήστη μια πιο immersive εμπειρία και η τεχνολογία στις συσκευές μας σήμερα μπορεί να το υποστηρίξει. Χρησιμοποιείται σε πολλούς τομείς, παραδείγματος χάριν από το στρατό, στην αρχαιολογία, την αρχιτεκτονική κ.λπ.

Σημαντικό εμπόδιο στην ανάπτυξη τεχνολογίας Augmented Reality είναι το πάντρεμα του πραγματικού τρισδιάστατου κόσμου με τον δισδιάστατο εικονικό που έχουμε στις συσκευές μας, ο προσδιορισμός δηλαδή της θέσης μας και της θέσης αντικειμένων που βλέπουμε μέσα από την κάμερά μας.

Ο σκοπός μας είναι η δημιουργία μιας εφαρμογής που με τη χρήση Augmented Reality να κάνει render ένα τρισδιάστατο αντικείμενο στο χώρο όπως φαίνεται από τη κάμερα της συσκευής μας και μετά να καθορίσουμε τη θέση της φωτεινής πηγής στο χώρο αυτό για να δώσουμε φυσικό φωτισμό στο αντικείμενο που κάναμε render.

Περιεχόμενα

Κεφάλαιο 1	Εισαγωγή.....	1
	1.1 Τί είναι Επαυξημένη Πραγματικότητα (AR)	1
	1.2 Προβλήματα AR	3
	1.3 Σκοπός μας	4
	1.4 Περίγραμμα ΑΔΕ	4
Κεφάλαιο 2	Augmented Reality.....	6
	2.1 Ιστορική Αναδρομή	6
	2.2 Pipeline του AR	7
	2.3 Φωτισμός (Illumination)	9
	2.4 Είδη AR	15
	2.5 AR Μηχανές	17
	2.6 Σύγκριση Μηχανών	17
	2.7 Επιλογή Μηχανής	19
Κεφάλαιο 3	Θεωρητική Ανάλυση Λύσης.....	20
	3.1 Θεωρία Λύσης	20
	3.2 Κομμάτια Συστήματος	21
Κεφάλαιο 4	Set Up.....	23
	4.1 Στήσιμο Unity	23
	4.2 Στήσιμο Vuforia	25
	4.3 Vumarks, ImageTargets, Object Recognition	26
Κεφάλαιο 5	Ανίχνευση & Τοποθέτηση Φωτεινής Πηγής.....	30
	5.1 Εισαγωγή	30
	5.2 Ανίχνευση Φωτεινής πηγής	31

5.3	Αλγόριθμος Gaussian Blur	32
5.4	Αλγόριθμος Pixelate	35
5.5	Τοποθέτηση Φωτεινής Πηγής	37
Κεφάλαιο 6	Αποτελέσματα.....	39
6.1	Προσθήκη Φωτισμού	39
6.2	Ανάλυση Gaussian Blur	41
6.3	Ανάλυση Pixelate	42
6.4	Σύγκριση Gaussian Blur & Pixelate	42
Κεφάλαιο 7	Συμπεράσματα	45
Βιβλιογραφία		46
Παράρτημα Α.....		A-1

Κεφάλαιο 1

Εισαγωγή

1.1 Τί είναι η Επαυξημένη Πραγματικότητα (AR)	1
1.2 Προβλήματα AR	3
1.3 Σκοπός μας	4
1.4 Περίγραμμα ΑΔΕ	4

1.1 Τί είναι η Επαυξημένη Πραγματικότητα (AR)

Επαυξημένη Πραγματικότητα (Augmented Reality), είναι η σύνδεση του πραγματικού κόσμου με τον εικονικό. Με άλλα λόγια παρεμβάλλει αντικείμενα και εικόνες που δημιουργούνται από τον υπολογιστή στο οπτικό πεδίο του χρήστη, ο οποίος μπορεί να βλέπει τον κόσμο μέσα από την κάμερα του κινητού ή με άλλα digital eyewear, και τους δίνει θέση με τρόπο που να φαίνονται ότι βρίσκονται πραγματικά εκεί.

Παρόλο που η ιδέα του Augmented Reality είναι αρκετά παλιά^[2], τα πρώτα στάδια της υλοποίησής της είναι αρκετά πρόσφατα. Από τότε η ανάπτυξή του AR ήταν ραγδαία και πολύ γρήγορα άρχισε να εξαπλώνεται παντού.

Η ανάπτυξη τέτοιας τεχνολογίας είναι πολύ σημαντική αφού φτάσαμε σε ένα σημείο που η συμβατική τεχνολογία δε μπορεί να μας προσφέρει περισσότερο immersion στον εικονικό κόσμο. Με τη χρήση AR μπορούμε να κάνουμε αυτό το έξτρα βήμα και να πετύχουμε αυτό το καινούριο επίπεδο βοηθώντας το χρήστη να βιώσει ένα ολοκαίνουριο είδος εμπειρίας. Για το λόγο αυτό όλο και περισσότερες εφαρμογές

κάνουν χρήση AR τεχνολογίας και η χρήση της βοηθά όλο και περισσότερο στην επιτυχία νέων εφαρμογών.

Υπάρχουν πολλοί τρόποι για τη χρήση του AR, παραδείγματος χάριν με τη χρήση markers γνωστών διαστάσεων για την εφαρμογή για τον ορισμό συγκεκριμένης θέσης και διαστάσεων για ένα αντικείμενο στο χώρο. Ένας άλλος τρόπος χρήσης του AR είναι με τη χρήση της γεωγραφικής θέσης της συσκευής μας σαν trigger για κάποιο γεγονός. Τέλος μερικές εφαρμογές AR έχουν τη δυνατότητα να αναγνωρίσουν χρώματα και σχήματα όπως επίσης και την τοπολογία του εδάφους, χωρίς να ορίσει κάτι συγκεκριμένο ο προγραμματιστής της εφαρμογής όπως με τα markers, και να χρησιμοποιήσει αυτές τις πληροφορίες για να πετύχει το σκοπό του. Σε αυτή την τελευταία κατηγορία μπάινει και το Face Recognition.

Η χρήση Augmented Reality έχει εξαπλωθεί σε ένα ευρύ φάσμα πεδίων, μερικές εκ των οποίων είναι η αρχαιολογία, η αρχιτεκτονική, η εκπαίδευση, η ιατρική, στο στρατό κ.α. Με τη βοήθεια της AR πετυχαίνουμε καλύτερη πρακτική εξάσκηση σε περιστάσεις που υπο άλλες συνθήκες θα ήταν πολύ επικύνδινες, μια πιο εύκολη και φτηνή λύση για visualization μοντέλων από τη δημιουργία μοκετών, καλύτερη κατανόηση του αντικειμένου που διδάσκεται μέσω αυτής της τεχνολογίας αφού δε δίνεται ωμή πληροφoρία στο μαθητή αλλά ζεί το μάθημα μέσω του AR. Αυτές είναι μόνο μερικές από τις κατηγορίες που βρίκει εφαρμογή το Augmented Reality.

Οι εξελίξεις στην ανάπτυξη εφαρμογών και υλικού για τον τομέα του AR τα τελευταία χρόνια είναι τεράστιες και αυτό αποτελεί ακόμα ένα γεγονός που μαρτυρεί τη σημαντικότητα που αποκτά αυτός ο τομέας στο μέλλον της τεχνολογίας. Μεγάλες εταιρίες όπως η Google με το Magic Leap, η Microsoft με το HoloLens, το Facebook με το Oculus Rift καθώς επίσης και εταιρίες παιχνιδιών όπως η Nintendo και η Sony βλέπουν την προοπτική ανάπτυξης στον τομέα του Augmented Reality και γι'αυτό επεκάθεκαν προς αυτή την κατεύθυνση αναπτύσσοντας υλικό που να υποστηρίζει τέτοια τεχνολογία. Παράλληλα με την ανάπτυξη υλικού δημιουργούν και λογισμικό για τις συσκευές τους και σε μερικές περιπτώσεις δίνουν πρόσβαση σε άτομα εκτός της εταιρίας για τη δημιουργία εφαρμογών που υποστηρίζεται από αυτούς. Με την προσοχή

που μαζεύει λοιπόν αυτός ο τομέας θα μπορούσε κάποιος να πεί ότι έχουμε να κάνουμε με την τεχνολογία του μέλλοντος.

1.2 Προβλήματα AR

Όπως είναι φυσικό μια τόσο νέα τεχνολογία έχει αρκετά προβλήματα. Μερικά από αυτά τα προβλήματα είναι η σωστή και ακριβείς τοποθέτηση των αντικειμένων στη σκηνή για να πετύχουμε το επιθυμητό αποτέλεσμα. Επίσης τα αντικείμενα που τοποθετούμε θέλουμε να δίνουν την αίσθηση στο χρήστη ότι είναι πραγματικά παρόντα στον πραγματικό κόσμο και αυτό είναι αρκετά δύσκολο να επιτευχθεί αφού πρέπει να προγραμματιστούν ώστε να αντιδρούν όταν αντιληφθούν ότι τους δίνεται κάποιο ερέθισμα από τον πραγματικό κόσμο και αυτό δεν είναι πάντα εφικτό αφού αυτό το ερέθισμα μπορεί να έρθει από άπειρα είδη διαδράσεων.

Όσον αφορά το θέμα της τοποθέτησης των αντικειμένων η λύση αναφέρθηκε πιο πάνω. Μπορούμε να πετύχουμε το σκοπό μας με τη χρήση ενός marker, ο οποίος μπορεί να είναι οτιδήποτε από QR Codes, Custom Markers από το περιβάλλον ανάπτυξης που χρησιμοποιούμε, τρισδιάστατα αντικείμενα ή ακόμα και ένα απλό σχέδιο, φτάνει αυτός ο marker να οριστεί εκ των πρωταίρων από τον προγραμματιστή ή το χρήστη. Αυτή η μέθοδος είναι και η πιο αξιόπιστη. Ένας άλλος τρόπος είναι η χρήση της γεωγραφικής θέσης της συσκευής μας έτσι ώστε όταν έρθουμε σε μία συγκεκριμένη απόσταση από την τοποθεσία που θέλουμε να ενεργοποιείται ένα event για το χρήστη της εφαρμογής. Τέλος μπορεί να χρησιμοποιηθεί πρόγραμμα ανάλυσης εικόνας για την αναγνώριση της τοπολογίας επιφανειών ή την ανίχνευση κάποιου pattern όπως για παράδειγμα ένα πρόσωπο για να τοποθετήσουμε κάτι στη σκηνή.

Το δεύτερο πρόβλημα, δηλαδή η δημιουργία της αίσθησης ότι το εικονικό αντικείμενο ανήκει στον πραγματικό κόσμο, είναι πιο πολύπλοκο και πιο δύσκολο να λυθεί. Ο λόγος γι' αυτό είναι το πλήθος των παραγόντων που πρέπει να λάβουμε υπόψη. Για παράδειγμα δεν πρέπει να υπάρχει καθυστέρηση μεταξύ του ερεθίσματος που δίνεται στο αντικείμενο και της αντίδρασής του. Αυτό δεν είναι πάντα εφικτό αφού όλα τα ερεθίσματα που λαμβάνονται πρέπει να τύχουν επεξεργασίας πρώτα, και όσο πιο απαιτητική είναι αυτή η διαδικασία τόσο ανεβάζουμε το latency μεταξύ ερεθίσματος

και αντίδρασης. Ο φωτισμός του αντικειμένου είναι ένας άλλος σημαντικός παράγοντας που συμβάλλει στην δημιουργία ρεαλιστικών αντικειμένων με τη χρήση υπολογιστή. Για να πετύχουμε σωστό φωτισμό πρέπει πρώτα να βρούμε τη θέση της φωτεινής πηγής και να προσθέσουμε κάτι αντιστοιχο στην εικονική σκηνή μας, κάτι το οποίο δεν είναι καθόλου εύκολο αφού η φωτεινή πηγή δεν είναι πάντα ορατή από τη θέση της συσκευής μας και ο καθορισμός της θέσης της είναι από μόνο του μια πρόκληση. Αυτό είναι επίσης και το θέμα με το οποίο θα ασχοληθούμε.

1.3 Σκοπός μας

Εμείς για αυτή την εργασία θα ασχοληθούμε με το κομμάτι του φωτισμού του αντικειμένου. Σκοπός μας είναι η δημιουργία εφαρμογής κινητού που με τη χρήση AR θα τοποθετεί ένα αντικείμενο στη σκηνή, η οποία θα ορίζεται από την κυρίως κάμερα του κινητού, και στη συνέχεια θα αναζητά τη θέση της φωτεινής πηγής στο δωμάτιο το οποίο βρισκόμαστε έτσι ώστε να τοποθετηθεί μια αντίστοιχη φωτεινή πηγή στη σκηνή μας και να φωτιστεί κατάλληλα το αντικείμενό μας.

Θεωρούμε ότι για την τοποθέτηση του αντικειμένου στο χώρο θα χρησιμοποιήσουμε ένα marker, βάσει του οποίου θα ορίζεται η θέση του αντικειμένου. Επίσης θεωρούμε ότι η συσκευή μας έχει δύο κάμερες, την πίσω-κύρια κάμερα την οποία θα χρησιμοποιεί το AR λογισμικό και την μπροστινή την οποία θα χρησιμοποιεί για εξαγωγή της θέσης της φωτεινής πηγής. Μια ακόμα υπόθεση που κάνουμε είναι ότι το κινητό βρίσκεται ανάμεσα στο αντικείμενο που ανιχνεύουμε και τη φωτεινή πηγή. Και τέλος μια προσέγγιση της θέσης της φωτεινής πηγής μας είναι αρκετή για την υλοποίηση της εφαρμογής.

1.4 Περίγραμμα ΑΔΕ

Το τί θα κάνουμε με περισσότερη λεπτομέρεια σ' αυτή την εργασία είναι ο φωτισμός ενός τρισδιάστατου αντικειμένου σύμφωνα με τη θέση της φωτεινής πηγής του δωματίου στο οποίο βρισκόμαστε, ακριβώς όπως θα φωτιζόταν ένα πραγματικό αντικείμενο. Το αντικείμενο θα τοποθετηθεί σε μια AR σκηνή, θα έχουμε δηλαδή feed από την κάμερα του κινητού, και με τη βοήθεια AR μηχανής θα αναγνωρίζουμε ένα

σημάδι ή εικόνα την οποία θα χρησιμοποιήσουμε σαν anchor point για να ορίσουμε τη θέση του αντικειμένου στο χώρο.

Όταν οριστεί η θέση του αντικειμένου θα γίνει χρήση της μπροστινής κάμερας μέσω της οποίας θα αναγνωρίσουμε τη θέση της φωτεινής πηγής σε σχέση με το κινητό και θα τοποθετηθεί στη σκηνή μας. Η θέση της φωτεινής πηγής θα θεωρήσουμε ότι είναι πάνω από το κινητό και από το αντικείμενό μας, και όχι ενδιάμεσά τους.

Η λήψη εικόνας από την μπροστινή κάμερα για την εύρεση της φωτεινής πηγής θα γίνεται είτε μια φορά για κάθε ένα συγκεκριμένο αριθμό frames ή θα δίνεται σήμα από το χρήστη αγγίζοντας την οθόνη. Ο λόγος γι' αυτό είναι το ότι χρειάζεται να διαβαστεί το κάθε pixel της εικόνας που θα λάβουμε ξεχωριστά. Επίσης θα πρέπει να τύχουν επεξεργασίας και ανάλυσης για να πάρουμε τα αποτελέσματα που θέλουμε. Αυτή η διαδικασία πού πιθανό να έχει πολύ ψηλό κόστος και σε μία εφαρμογή που επικοινωνεί με το χρήστη μέσω εικόνας από την κάμερα δεν μπορούμε να διακόπτουμε την λειτουργία της συνέχεια για υπολογισμούς.

Κεφάλαιο 2

Augmented Reality

2.1 Ιστορική αναδρομή	6
2.2 Pipeline του AR	7
2.3 Φωτισμός (Illumination)	9
2.4 Είδη AR	15
2.5 AR Μηχανές	17
2.6 Σύγκριση Μηχανών	17
2.7 Επιλογή Μηχανής	19

2.1 Ιστορική Αναδρομή

Αναφορά για πρώτη φορά σε Augmented Reality έγινε το 1901 από το συγγραφέα L. Frank Baum^[2] ο οποίος αναφέρεται, σε έργο του, γυαλιά τα οποία προβάλλουν πληροφορίες στον πραγματικό κόσμο. Αργότερα ο σκηνογράφος Morton Heilig δημιούργησε ένα εξομοιωτή με το όνομα Sensorama και χρησιμοποιούσε οπτικά εφέ, ήχο, μυροδιά και δόνηση για να δημιουργήσει μια από τις πρώτες AR εμπειρίες. Σιγά σιγά η τεχνολογία AR άρχισε να εξελίσσεται επιτρέποντας στο χρήστη να αλληλεπιδρά με τα αντικείμενα που προβάλλονταν στον εικονικό κόσμο.

Παράλληλα με την ανάπτυξη του Augmented Reality είχαμε και την ανάπτυξη του Virtual Reality και οι δύο αυτές τεχνολογίες βοηθούσαν η μία την άλλη να εξελιχθούν. Το 1989 ο Jaron Lanier δημιουργεί την πρώτη επιχείρηση που ασχολείται με Virtual Worlds.

Το 1992 ο Louis Rosenberg δημιούργησε το πρώτο λειτουργικό AR σύστημα το οποίο αποτέλεσε απόδειξη για τα πλεονεκτήματα που μπορεί να προσφέρει το Augmented Reality στην εκπαίδευση και τη βελτίωση της ανθρώπινης απόδοσης. Από κεί και πέρα η χρήση AR συστημάτων άρχισε να διαδίδεται σε διάφορους τομείς της επιστήμης και άλλων σπουδών.

Μερικά παραδείγματα της χρήσης Augmented Reality είναι η χρήση της για την αναγνώριση συντριμιών και άλλων αντικειμένων στο διάστημα με την προβολή δορυφορικών γεογραφικών τροχιών σε ζωντανή μετάδοση τηλεσκοπίου. Επίσης σημαντικό παράδειγμα είναι ο συνδυασμός AR και οδήγησης με την πρώτη παρουσίαση να γίνεται το 1993 από τα Loral Western Development Laboratories. Το 1995 στο Massachusetts University άρχισε η ανάπτυξη Object Tracking συστημάτων με τη χρήση καμερών και τεχνολογιών AR, και το 1998 το Augmented Reality εξαπλώθηκε και στον τομέα της χωρομετρίας για μέτρηση διαστάσεων χώρου και όγκου στο North Carolina University.

Τα παραδείγματα συνεχίζουν μέχρι και σήμερα και η εξέλιξη των AR συστημάτων συνεχίζει ακάθεκτη δείχνοντας ότι έχει μεγάλες προοπτικές για ανάπτυξη και επέκταση βιομηχανιών προς το μέρος τους.

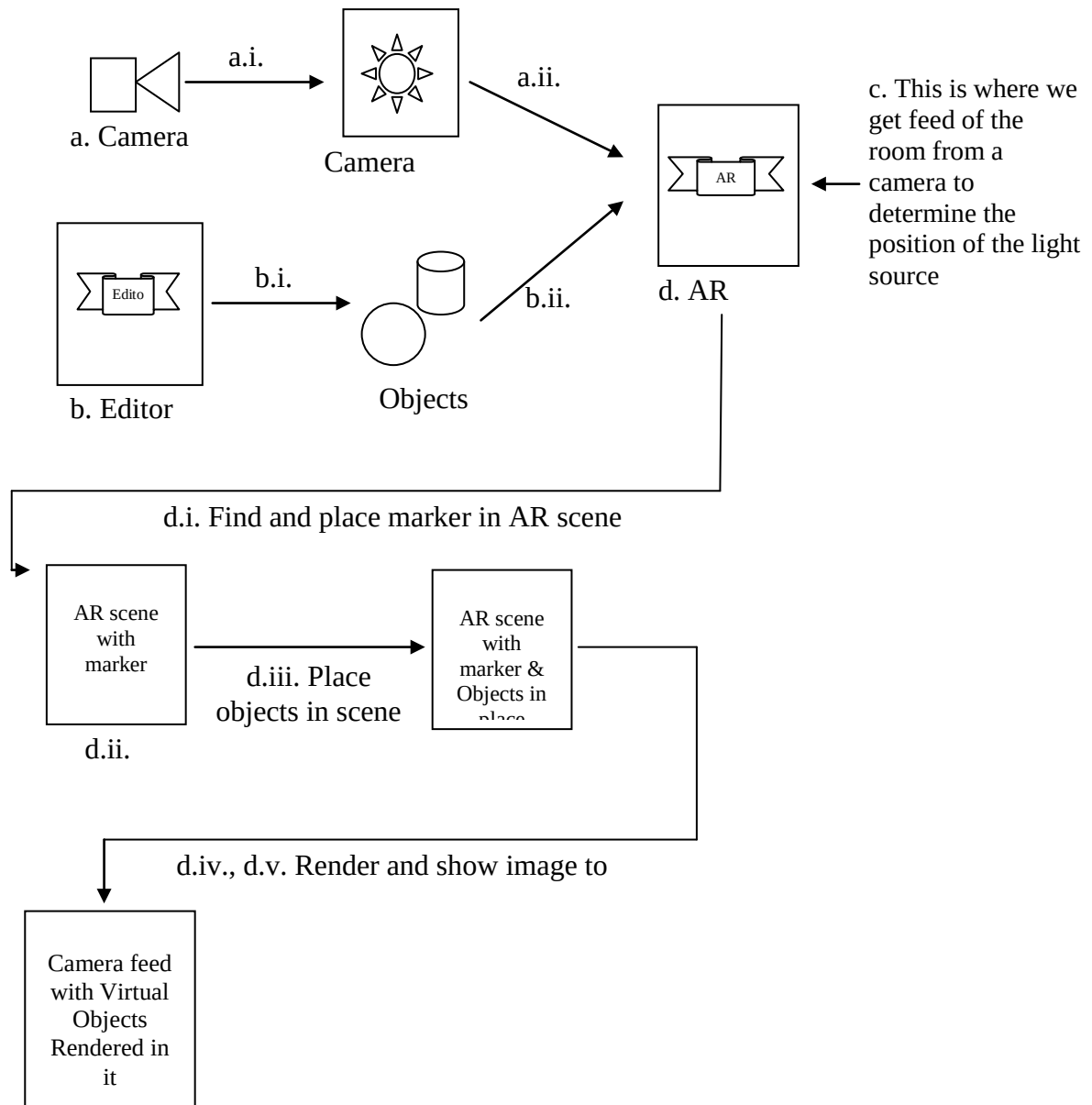
2.2 Pipeline του AR

Ο τρόπος που λειτουργεί ένα AR σύστημα με τη χρήση κάμερας έχει ως εξής:

- a) Από την πλευρά της κάμερας:
 - i. Παίρνουμε την εικόνα που βλέπει η κάμερα.
 - ii. Τη στέλνουμε στο AR section του προγράμματός μας για επεξεργασία.
- b) Από την πλευρά του Editor μου:
 - i. Δημιουργώ το αντικείμενο.
 - ii. Το στέλνω στο AR section του προγράμματός μου.
- c) Εδώ παίρνουμε το feed από άλλη κάμερα για αναγνώριση της θέσης της φωτεινής πηγής και το στέλνω στο AR section του προγράμματός μου.
- d) Στο AR section του προγράμματός μου:

- i. Μόλις λάβω το feed της κάμερας και τις παραμέτρους του αντικειμένου αναλύω την εικόνα ψάχνοντας το marker μου ο οποίος θα χρησιμοποιηθεί ως σημείο αναφοράς για την τοποθέτηση του αντικειμένου μου.
- ii. Μόλις βρεθεί ο marker σύμφωνα με το μέγεθός του και τις διαστάσεις του βρίσκω την απόστασή του και το orientation του σε σχέση με την κάμερα και τα ορίζω μέσα στην AR σκηνή μου.
- iii. Η θέση και το orientation του αντικειμένου σε σχέση με τον marker δίνεται ήδη από το πρόγραμμα οπότε το μόνο που έχουμε να κάνουμε είναι να το τοποθετήσουμε στην AR σκηνή.
- iv. Στο background της σκηνής μας πέζει το feed από τη κάμερα δίνοντας τη ψευδαίσθηση ότι το αντικείμενο τοποθετήθηκε στον πραγματικό κόσμο.
- v. Η συνολική εικόνα δίνεται στο χρήστη από την οθόνη της συσκευής του.

Σχήμα του Pipeline δίνεται πιο κάτω (βλέπε Σχήμα 2.1)



Σχήμα 2.1 – AR Pipeline

2.3 Φωτισμός (Illumination)^[4]

Στην πραγματικότητα το φως ξεκινά από τη φωτεινή πηγή και διαχέεται προς όλες τις κατευθύνσεις, αναπηδά και απορροφάται πάνω σε διάφορες επιφάνειες μέχρι που στο τέλος φτάνει στο ανθρώπινο μάτι. Αναλόγως με το μονοπάτι που ακολούθησε κάθε ακτίνα φωτός που αντιλαμβανόμαστε μας μεταφέρει και κάποια πληροφορία για τα αντικείμενα με τα οποία ήρθε σε επαφή.

Στα γραφικά έχουμε δύο είδη φωτισμού για να προσομοιώσουμε αυτό, τον γενικό φωτισμό (Global Illumination) και τον τοπικό φωτισμό (Local Illumination). Ο γενικός φωτισμός δε λαμβάνει υπόψη μόνο τις ακτίνες που έρχονται κατευθείαν από τη φωτεινή πηγή και καταλήγουν στο μάτι του θεατή αλλά και αυτές που ανακλάστηκαν προηγουμένως από άλλες επιφάνειες. Κάτι τέτοιο έχει εξαιρετικά μεγάλο κόστος γι' αυτό δεν χρησιμοποιείται συχνά εκτός και αν θέλουμε να κατασκευάσουμε μια πολύ υψηλής ποιότητας εικόνα και έχουμε το χρόνο να κάνουμε όλους τους υπολογισμούς που χρειάζεται. Ο τοπικός φωτισμός από την άλλη υπολογίζει μόνο τις ακτίνες που καταλήγουν στο μάτι του θεατή και έρχονται κατ' ευθείαν από τη φωτεινή πηγή γλιτώνοντας μας πολύ χρόνο και κόστος σε υπολογιστική δύναμη.

Για να πετύχουμε μια καλύτερη προσέγγιση του γενικού φωτισμού χρησιμοποιώντας τοπικό, εισάγουμε τον έμμεσο φωτισμό (Ambient Light) στη σκηνή μας. Ο έμμεσος φωτισμός αντιπροσωπεύει το "αδέσποτο" φώς στη σκηνή μας που προέρχεται από την αντανάκλαση φωτός από άλλα αντικείμενα. Η τιμή του είναι σταθερή σε όλη τη σκηνή και φωτίζει όλα τα αντικείμενα ακόμα κι αν δεν φωτίζονται κατ' ευθείαν από κάποια φωτεινή πηγή. Κατα τον υπολογισμό του έμμεσου φωτισμού λαμβάνουμε υπόψη την έντασή του (I_a) και τον συντελεστή έμμεσης ανάκλασης (k_a) και η εξίσωσή του είναι:

$$\Rightarrow I_{r,a} = I_a * k_a$$

Αυτός ο υπολογισμός πρέπει να γίνει ξεχωριστά για κάθε ένα από τα RGB και το αποτέλεσμα του φαίνεται στο πιο κάτω σχήμα (Βλέπε Σχήμα 2.2).

Το πιο συνηθισμένο μοντέλο φωτισμού στα γραφικά, και το μοντέλο που θα χρησιμοποιήσουμε, είναι το μοντέλο Phong το οποίο λαμβάνει υπόψη δύο απλές μορφές ανακλάσεων, την κατευθυνόμενη ανάκλαση (Specular Reflection) και την διάχυτη ανάκλαση (Diffuse Reflection).

Για την κατευθυνόμενη ανάκλαση θεωρούμε ότι όταν μια ακτίνα φωτός χτυπά ένα αντικείμενο έχει ως αποτέλεσμα την ανάκλαση αυτής της ακτίνας προς μια άλλη κατεύθυνση (βλέπε Σχήμα 2.3). Έχοντας υπόψη τις παραμέτρους E, N, L και H (Βλέπε Σχήμα 2.5) όπου E είναι η κατεύθυνση προς το μάτι, N είναι η κάθετος στο σημείο που έχει επαφή η ακτίνα με την επιφάνεια του αντικειμένου μας, L είναι η κατεύθυνση προς

τη φωτεινή πηγή και H η διχοτόμος των H και L καταλήγουμε στην εξίσωση για κατευθυνόμενο φωτισμό:

$$\Rightarrow I_{r,s} = K_s * I^*(H.N)^m \text{ (Βλέπε Σχήμα 2.10)}$$

Όπου το k_s είναι ο συντελεστής κατευθυνόμενης ανάκλασης, και εξαρτάται με το υλικό της επιφάνειάς μας (Βλέπε Σχήμα 2.6), I είναι το intensity της ακτίνας μας και το m καθορίζει τα χαρακτηριστικά του highlight. Ψηλό m σημαίνει ότι το highlight είναι συγκεντρωμένο γύρω από το σημείο ανάκλασης ενώ χαμηλό m σημαίνει πιο πλατύ και θολό highlight γύρω από το σημείο ανάκλασης (Βλέπε Σχήμα 2.7).

Για την διάχυτη ανάκλαση θεωρούμε ότι όταν μια ακτίνα χτυπήσει μια επιφάνεια το φως αντανακλάται εξίσου προς όλες τις κατευθύνσεις (Βλέπε Σχήμα 2.4). Το ποσοστό του φωτός που αντανακλάται διάχυτα εξαρτάται από το συντελεστή διάχυτης ανάκλασης του αντικείμενου k_d και την κατεύθυνση και ένταση της εισερχόμενης ακτίνας φωτός. Επίσης ακολουθεί το νόμο του Lambert. Ο νόμος του Lambert λέει ότι η διάχυτη ανάκλαση του φωτός είναι ανάλογη του $\cos \theta$ όπου θ είναι η γωνία μεταξύ της κατεύθυνσης της φωτεινής πηγής (L) και της καθέτου του σημείου στο οποίο συναντά η ακτίνα το αντικείμενο (N) (Βλέπε Σχήμα 2.8). Η εξίσωση για το διάχυτο φωτισμό είναι:

$$\Rightarrow I_{r,d} = k_d * I * \cos \theta \quad \text{ή} \quad I_r = k_d * I^*(N.L) \text{ (Βλέπε Σχήμα 2.9)}$$

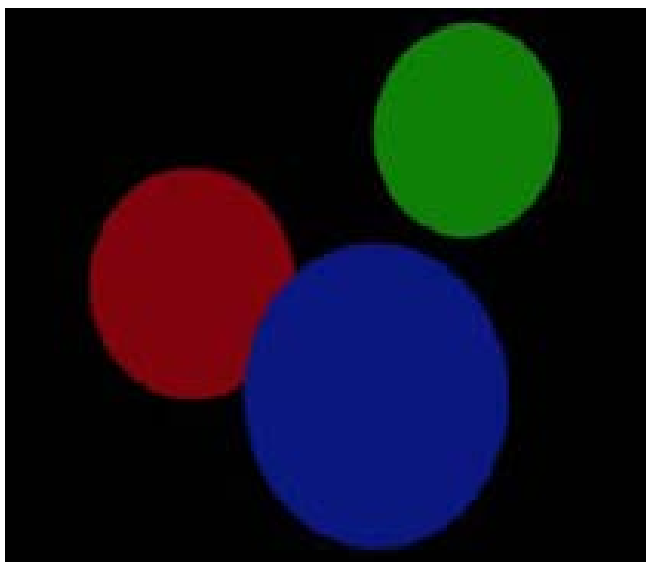
Βάζοντας μαζί όλες τις εξισώσεις που βρήκαμε καταλήγουμε στο μοντέλο φωτισμού Phong και η εξίσωσή ακτίνας με βάση αυτό το μοντέλο είναι:

$$\Rightarrow I_r = I_a * k_a + I^*(k_d * (N.L) + k_s * (H.N)^m)$$

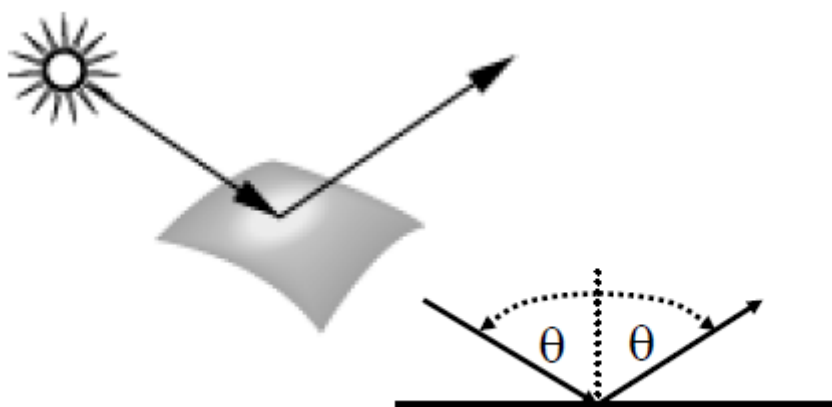
Στην περίπτωση που έχουμε πολλαπλές φωτεινές πηγές η πιο πάνω εξίσωση γίνεται:

$$\Rightarrow I_r = I_a * k_a + \sum_{j=1}^p I_j^*(k_d * (N.L_j) + k_s * (H_j.N)^m)$$

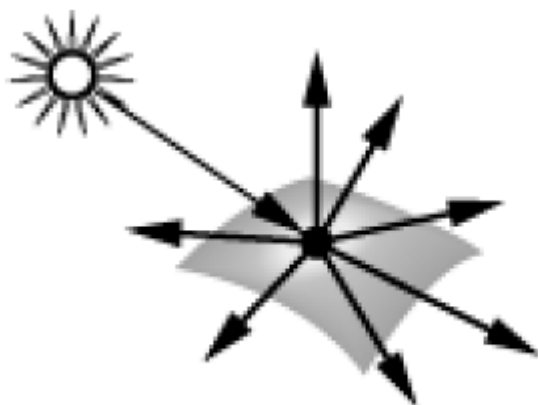
Όπου j ο αριθμός της κάθε φωτεινής πηγής. Το I_r πρέπει να υπολογιστεί ξεχωριστά για κάθε μια από τις τιμές RGB.



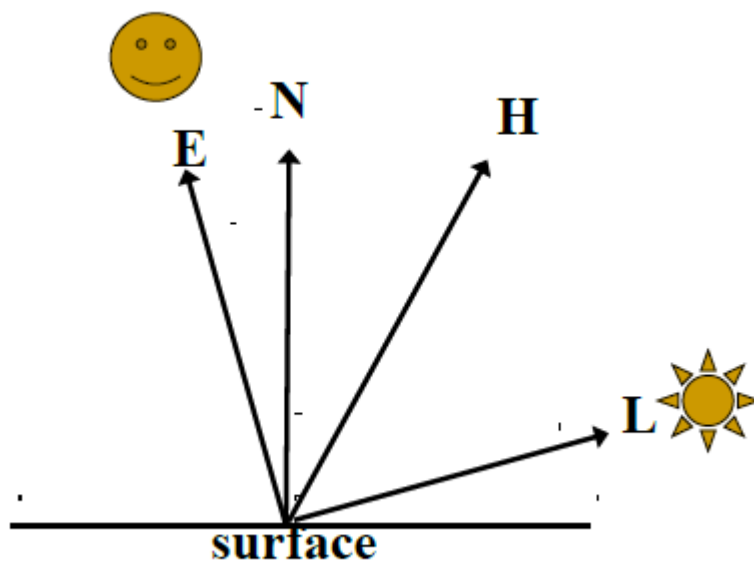
Σχήμα 2.2 – Φωτισμός σκηνής μόνο με Ambient Light



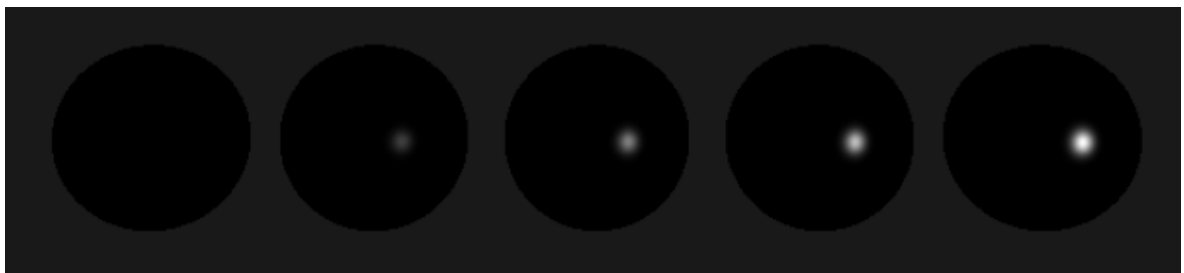
Σχήμα 2.3 – Κατευθυνόμενη Ανάκλαση (Specular Reflection)



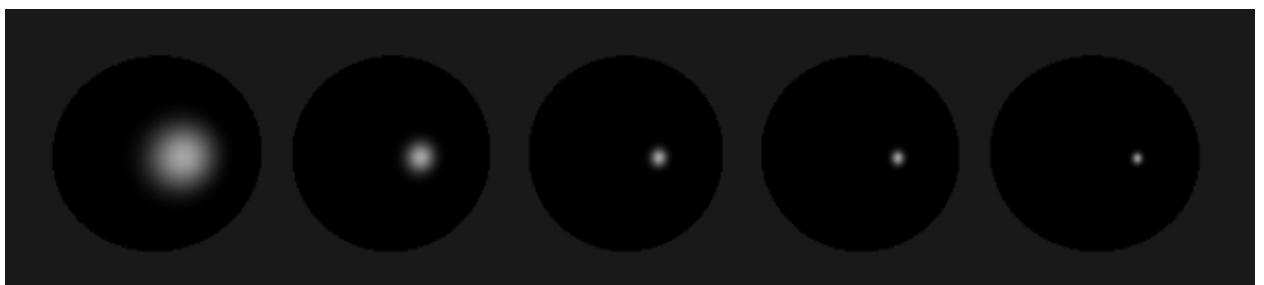
Σχήμα 2.4 – Διάχυτη Ανάκλαση (Diffuse Reflection)



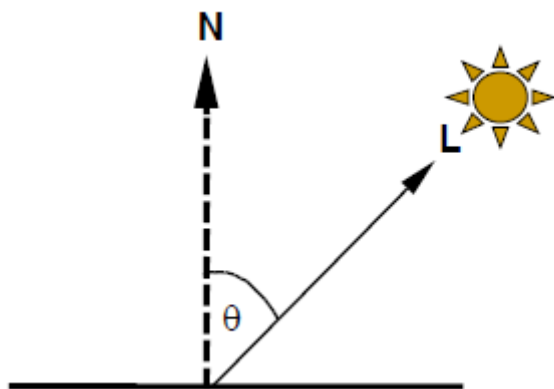
Σχήμα 2.5 – Ατελής κατευθυνόμενη ανάκλαση



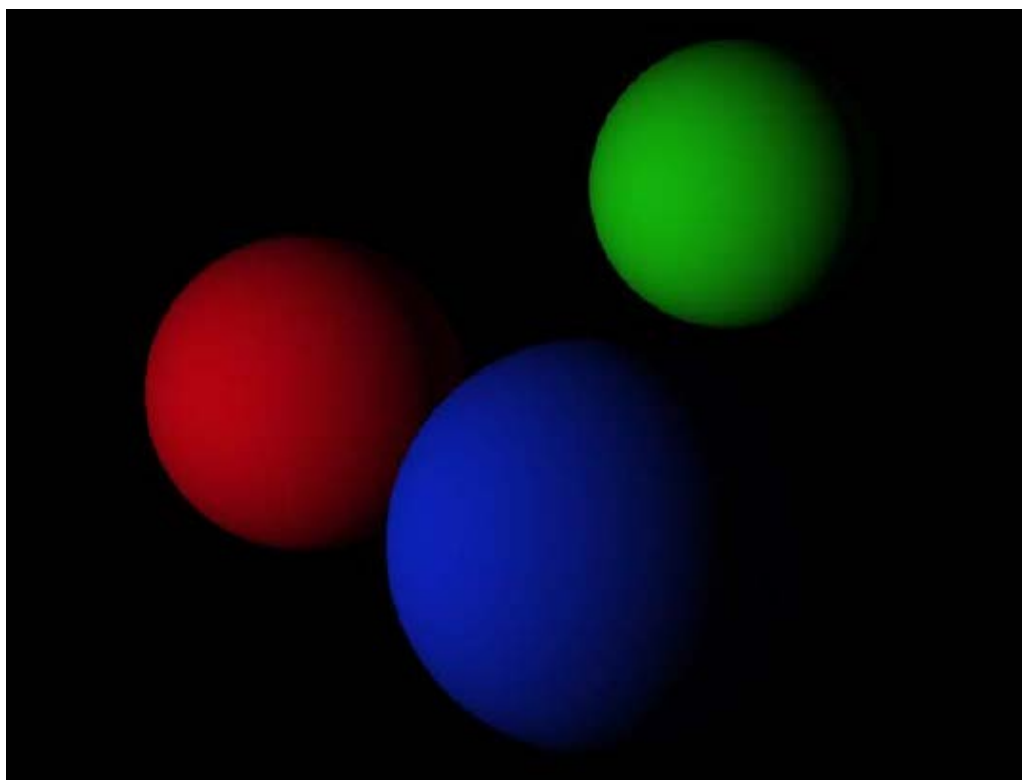
Σχήμα 2.6 – Ανάκλαση για τιμές k_s 0, 0.25, 0.5, 0.75, 1



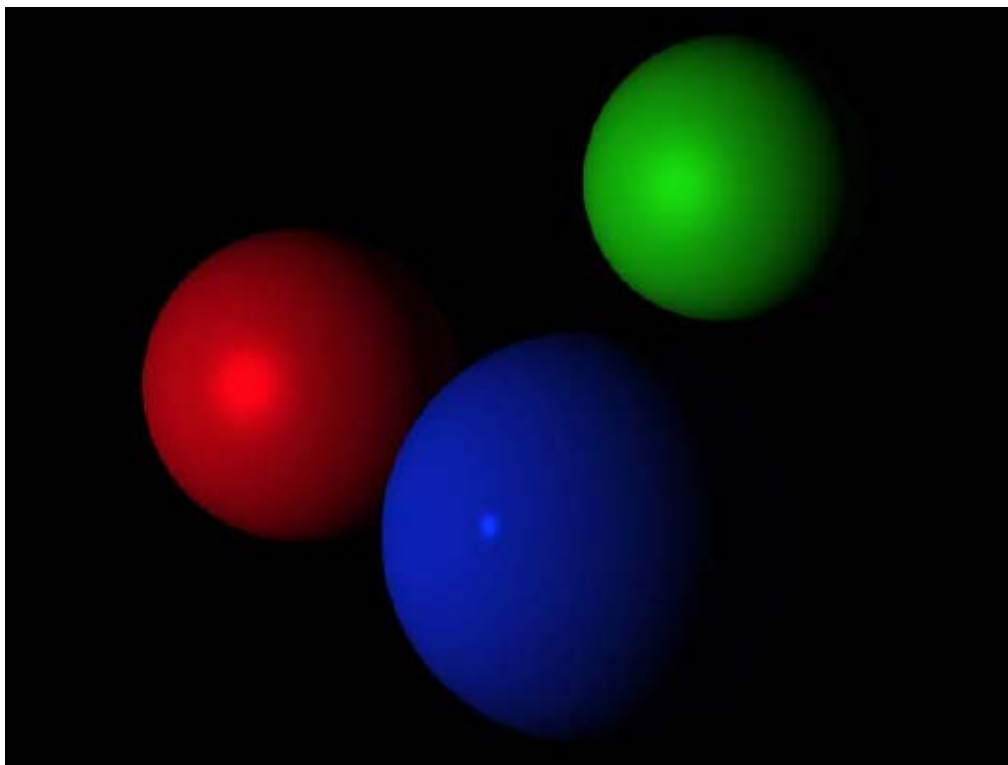
Σχήμα 2.7 – Ανάκλαση για τιμές m 5, 25, 75, 125, 225



Σχήμα 2.8 – Κανόνας Lambert



Σχήμα 2.9 – Φωτισμός σκηνής μόνο με Diffuse Reflection



Σχήμα 2.10 – Φωτισμός σκηνής Diffuse και Specular Reflection

2.4 Είδη AR

Υπάρχουν πολλά είδη AR καθώς επίσης και διάφορες συσκευές. Αν και δεν υπάρχει επίσημος τρόπος διαχωρισμού τους, μερικά από τα πιο δημοφιλή (και σε περισσότερη λεπτομέρεια από αυτά που αναφέρθηκαν πιο πάνω) είδη είναι:

- a) Projection based AR. Σε αυτό το είδος AR τα ψηφιακά αντικείμενα προβάλλονται στο χώρο και επιτρέπουν διάδραση με το χρήστη, όπως για παράδειγμα η προβολή ενός digital keyboard.
- b) Recognition based AR. Αυτό είναι από τα πιο διαδεδομένα είδη AR και έχει να κάνει με αναγνώριση εικόνων και σημαδιών που δώθηκαν εκ των προταίρων από τον προγραμματιστή ή το χρήστη. Ένα παράδειγμα τέτοιου είδους AR είναι η σάρωση QR codes ή εικόνων.
- c) Location based AR. Σε αυτή την κατηγορία η εφαρμογή AR κάνει χρήση του GPS, της πυξίδας και του accelerometer της συσκευής του χρήστη για να λάβει πληροφορίες σχετικά με τη θέση του κινητού.

- d) Outlining AR. Αυτή η κατηγορία είναι πολύ παρόμοια με τις Projection και Recognition based AR, δεν αναγνωρίζει όμως προκαθορισμένα σημάδια αλλά αναγνωρίζει patterns στον πραγματικό κόσμο. Για παράδειγμα τα καινούρια συστήματα στάθμευσης αναγνωρίζουν εμπόδια γύρω από το αυτοκίνητο και τα όρια του δρόμου και καθοδηγούν τον οδηγό για να σταθμεύσει.
- e) Superimposition based AR. Αυτό το μοντέλο, όπως και το προηγούμενο, κάνει χρήση object recognition με τη διαφορά ότι αντικαθιστά μέρος ή ολόκληρο το πραγματικό αντικείμενο με ένα ψηφιακό. Παραδείγματα τέτοιου AR μπορούμε να βρούμε σε στρατιωτικό εξοπλισμό (Infrared ή Night Vision) ή στην Ιατρική (X-rays και CT scans).

Πέρα από τα είδη AR που αφορούν το λογισμικό κομμάτι της εμπειρίας έχουμε επίσης και το υλικό που τα υποστηρίζει. Το λογισμικό του AR μπορεί να χρησιμοποιηθεί με διάφορες συσκευές και ανάλογα με το πόσο εξειδικευμένες είναι, δίνουν στο χρήστη μια πιο ζωντανή και πραγματική εμπειρία. Για παράδειγμα μια εφαρμογή AR μπορεί να τρέξει πάνω σε κινητό ή tablet, μπορεί όμως να χρησιμοποιηθεί σε συνεργασία με Head-Mounted Displays όπως Oculus Rift και Microsoft Hololens για μια ακόμη πιο immersive εμπειρία. Επίσης μπορεί να γίνει χρήση HUD (head-up display) για την προβολή πληροφοριών πάνω σε μια διαφανή επιφάνεια, όπως για παράδειγμα ο μπροστινό γυαλί του αυτοκινήτου ή πάνω στα γυαλιά ενός ατόμου, δίνοντας έτσι τη δυνατότητα επεξεργασίας αυτών των πληροφοριών παρακολουθώντας παράλληλα τί γίνεται στον πραγματικό κόσμο.

Νέες τεχνολογίες αναπτύσσονται κάθε μέρα για τη χρήση AR συστημάτων. Γίνεται ήδη έρευνα πάνω σε φακούς επαφής οι οποίοι έχουν τη δυνατότητα να προβάλλουν εικόνες μέσα στο οπτικό πεδίο του χρήστη με τη χρήση κυκλωμάτων ενσωματωμένων στο φακό. Ένα άλλο παράδειγμα είναι το Virtual Retinal Display με τη χρήση του οποίου γίνεται προβολή μιας οθόνης στον αμφιβληστροειδή χιτώνα του χρήστη η οποία θα φαίνεται να αιωρείται στο χώρο. Ένα τελευταίο παράδειγμα τέτοιας τεχνολογίας είναι η χρήση ειδικών γυαλιών τα οποία αιχμαλωτίζουν το φως, που θα έφτανε κανονικά στο μάτι του χρήστη, και το αντικαθιστούν με δικό τους φως δίνοντας έτσι μια διαφορετική εικόνα στο χρήστη.

Υπάρχουν ακόμη πάρα πολλά παραδείγματα από projects που έχουν σκοπό την ανάπτυξη και βελτίωση των τεχνολογιών AR, όμως δε θα αναφέρουμε άλλα εδώ.

2.5 AR Μηχανές

Για το κομμάτι της εφαρμογής το οποίο θα χρησιμοποιεί AR για την αναγνώριση των marker και την τοποθέτηση των αντικειμένων στη σκηνή θα χρησιμοποιήσω έτοιμες βιβλιοθήκες και προγράμματα.

Μερικές από τις μηχανές που μπορούν να μας βοηθήσουν να υλοποιήσουμε το AR κομμάτι της εφαρμογής είναι:

- DroidAR
- Metaio
- Vuforia
- Wikitude

Οι πιο πάνω μηχανές προσφέρονται δωρεάν και μερικές από αυτές είναι open-source η/και πολλά άτομα συνέβαλαν στην κατασκευή τους. Όλες τους μπορούν να χρησιμοποιηθούν για την ανάπτυξη της εφαρμογής, θα κρατήσουμε όμως αυτήν που είναι πιο εύκολο να χρησιμοποιηθεί και στην οποία παρέχεται καλύτερη υποστήριξη.

Μια σύντομη περιγραφή μεταξύ αυτών των τεσσάρων μηχανών και τα κριτήρια σύγκρισής τους παρατίθενται στο επόμενο υποκεφάλαιο.

2.6 Περιγραφή/Σύγκριση Μηχανών AR

Για κάθε μια από τις μηχανές AR που ονόμασα πιο πάνω θα δώσω μερικές πληροφορίες καθώς επίσης και το τι προσφέρει (ή δεν προσφέρει).

- **DroidAR**

Το DroidAR είναι μια δομή για ανάπτυξη Augmented Reality εφαρμογών στο Android. Προσφέρει Location και Marker based AR, δηλαδή η δομή μπορεί είτε να αναγνωρίσει

ένα marker (που μπορεί να είναι οποιαδήποτε εικόνα, QR code, κλπ) που δόθηκαν εκ των προτέρων από τον χρήστη, ή να χρησιμοποιήσει Real Time Tracking μέσω της συσκευής που χρησιμοποιεί την εφαρμογή (πχ Pokémon GO), για να χρησιμοποιήσει ως σημείο αναφοράς ή για να ενεργοποιήσει κάποιο script.

Έρχεται με άδεια τύπου GNU GPL v3 και δίνεται πρόσβαση σε αυτό μέσω του gitHub για κατέβασμα και χρήση ή για συνεισφορά στον κώδικα. Οι δημιουργοί του προσφέρουν το δικό τους SDK για πιο εύκολη χρήση και ενσωμάτωση στον κώδικα όποιου θέλει να το χρησιμοποιήσει. Υπάρχει επαρκής σχολιασμός και υποστήριξη από τους δημιουργούς, όμως η κοινότητα δεν είναι αρκετά μεγάλη και δεν υπάρχει forum, εξ όσων γνωρίζω, για αναζήτηση βοήθειας από άλλους χρήστες.

- **Metaio**

Το Metaio δημιουργήθηκε από ιδιωτική εταιρία η οποία έχει τις ρίζες της στη Volkswagen και προσέφερε AR SDK's και εφαρμογές. Αγοράστηκε το 2015 από την Apple και πρόσφατα σταμάτησε τη διανομή των προϊόντων της και παρέχει υπηρεσίες και υποστήριξη μόνο σε χρήστες που αγόρασαν και χρησιμοποιούν ήδη την τεχνολογία τους.

- **Vuforia**

Το Vuforia είναι ίσως η πιο διαδεδομένη AR SDK. Προσφέρει Marker και Markerless based AR, μπορεί δηλαδή να αναγνωρίσει εικόνες τρισδιάστατα αντικείμενα ή κάποιο είδος marker που του δόθηκε προηγουμένως από το χρήστη όπως επίσης και τη μορφή της επιφάνειας την οποία βλέπει (terrain recognition) δυναμικά.

Μπορεί να χρησιμοποιηθεί σε συνεργασία με το Android Studio για ανάπτυξη εφαρμογών για το κινητό, επιτρέπει όμως και τη χρήση του μέσω του Unity για ανάπτυξη και άλλου είδους προγραμμάτων. Έχει αρκετά μεγάλο community, υποστήριξη από τους developers, και απαντήσεις σε προβλήματα ή ερωτήσεις που μπορεί να προκύψουν μπορούν να βρεθούν εντός και εκτός των forums που βρίσκονται στη σελίδα τους.

- **Wikitude**

Το Wikitude είναι μια AR μηχανή που δημιουργήθηκε αρχικά για να προσφέρει Location Based AR αργότερα όμως επεκτάθηκε για να υποστηρίζει και user defined εικόνες. Είναι αρκετά διαδεδομένο όπως και το Vuforia είναι όμως πιο εξειδικευμένο για ανάπτυξη Location Based AR.

Υπάρχει αρκετή υποστήριξη από τους Developers του και ένα καλά στημένο forum για την υποστήριξη όσων επιθυμούν να το χρησιμοποιήσουν. Προσφέρει με free trial το Wikitude Studio για ανάπτυξη εφαρμογών, μπορεί όμως να χρησιμοποιηθεί και με το Android Studio και προσφέρει extensions για χρήση των Cordova, Xamarin και Unity για την ανάπτυξη εφαρμογών.

2.7 Επιλογή Λογισμικού

Για τους σκοπούς της Διπλωματικής Εργασίας μου επέλεξα να χρησιμοποιήσω το Vuforia. Ο κύριος λόγος που το επέλεξα είναι η υποστήριξη που μπορείς να βρεις online μέσα από τα forums, την ιστοσελίδα και διάφορες άλλες πηγές. Επίσης προσφέρει όλα όσα ζητώ για την εφαρμογή μου και μοιάζει να είναι πιο εύρωστο σε σχέση με τα υπόλοιπα.

Σε συνεργασία με το Vuforia θα χρησιμοποιήσω το Unity, επειδή μου προσφέρει έναν πιο εύκολο τρόπο για να υλοποιήσω μεγάλο μέρος της εργασίας μου μέσα από το γραφικό περιβάλλον του, ενώ παράλληλα μου επιτρέπει να έχω πρόσβαση στον κώδικά μου. Επίσης οι αρχικές δοκιμές μπορούν να γίνουν στον υπολογιστή, μέσω του Unity για τη βοήθεια που θα μου προσφέρει το feedback που θα έχω από το console του για το debugging. Για τα scripts του Unity μπορώ να χρησιμοποιήσω είτε C# ή JavaScript.

Κεφάλαιο 3

Θεωρητική Ανάλυση Λύσης

3.1 Θεωρία Λύσης	20
3.2 Κομμάτια του Συστήματος	21

3.1 Θεωρία Λύσης

Η δουλειά του συστήματος θα είναι καταρχάς να παίρνει με τη χρήση κάμερας εικόνα από τον πραγματικό κόσμο. Θα γίνεται ανάλυση αυτής της εικόνας για ανεύρεση ενός marker και θα τοποθετείται σε μια AR σκηνή η οποία θα δημιουργηθεί επίσης από στο σύστημα και θα περιέχει τις θέσεις των αντικειμένων μας, του marker, των καμερών μας και του background το οποίο θα παίζει live feed από την κάμερα. Όλα τα πιο πάνω γίνονται μέσα στον editor και με τη βοήθεια ενός AR SDK.

Ταυτόχρονα μια δεύτερη κάμερα, με θέση ίδια με την πρώτη αλλά βλέποντας προς την αντίθετη κατεύθυνση, θα πάρει στιγμιότυπο και θα το χρησιμοποιήσει για να βρεί το πιο φωτεινό του σημείο το οποίο θα αντιπροσωπεύει και τη φωτεινή πηγή στο δωμάτιο. Η θέση αυτού του σημείου θα μεταφραστεί σε παγκόσμιες συντεταγμένες για την AR σκηνή μας και σε εκείνη τη θέση θα τοποθετήσουμε ένα Light Source το οποίο θα φωτίζει τα αντικείμενα που παρεμβάλλουμε στο feed της κάμεράς μας.

3.2 Κομμάτια του Συστήματος

Ολόκληρη η λύση μου μπορεί να σπάσει σε δύο κατηγορίες, τα μέρη που ανήκουν στον πραγματικό κόσμο και τα μέρη που ανήκουν στο σύστημα, και κάθε μία απ' αυτές αποτελείται από τα δικά τους components (Βλέπε Σχήμα 3.1):

- a) Κομμάτια στον πραγματικό κόσμο
 - a. Συσκευή που χρησιμοποιούμε (κινητό/tablet)
 - i. Μπροστά κάμερα, την οποία χρησιμοποιούμε για να βρούμε τη φωτεινή πηγή.
 - ii. Πίσω κάμερα, την οποία χρησιμοποιούμε για να βρούμε τον marker και θα αποτελέσει το feed για το background της σκηνής μας.
 - iii. Οθόνη, πάνω στην οποία θα εμφανίζουμε τα αποτελέσματά μας, δηλαδή το feed της πίσω κάμερας με το εικονικό αντικείμενό μας τοποθετημένο στη θέση που υποδεικνύει ο marker.
 - b. Φωτεινή πηγή, βάσει της οποίας θα φωτιστεί το αντικείμενό μας.
 - c. Marker
- b) Κομμάτια Συστήματος
 - a. Editor, μέσω του οποίου θα δημιουργήσουμε την AR σκηνή και θα χρησιμοποιήσουμε για την τοποθέτηση των κομματιών της. Επίσης θα ενώσει το λογισμικό AR με τους αλγόριθμούς μας για να παράξει μια ολοκληρωμένη λύση.
 - i. Θέση marker.
 - ii. Θέση αντικειμένων σε σχέση με τον marker.
 - iii. Θέση κάμερας.
 - iv. Θέση Φωτεινής πηγής.
 - b. Λογισμικό AR, χρησιμοποιείται για την αναγνώριση και τον ορισμό των παραμέτρων του marker (απόσταση και orientation) και τα επιστρέφει στον editor για να μπουν στην AR σκηνή.

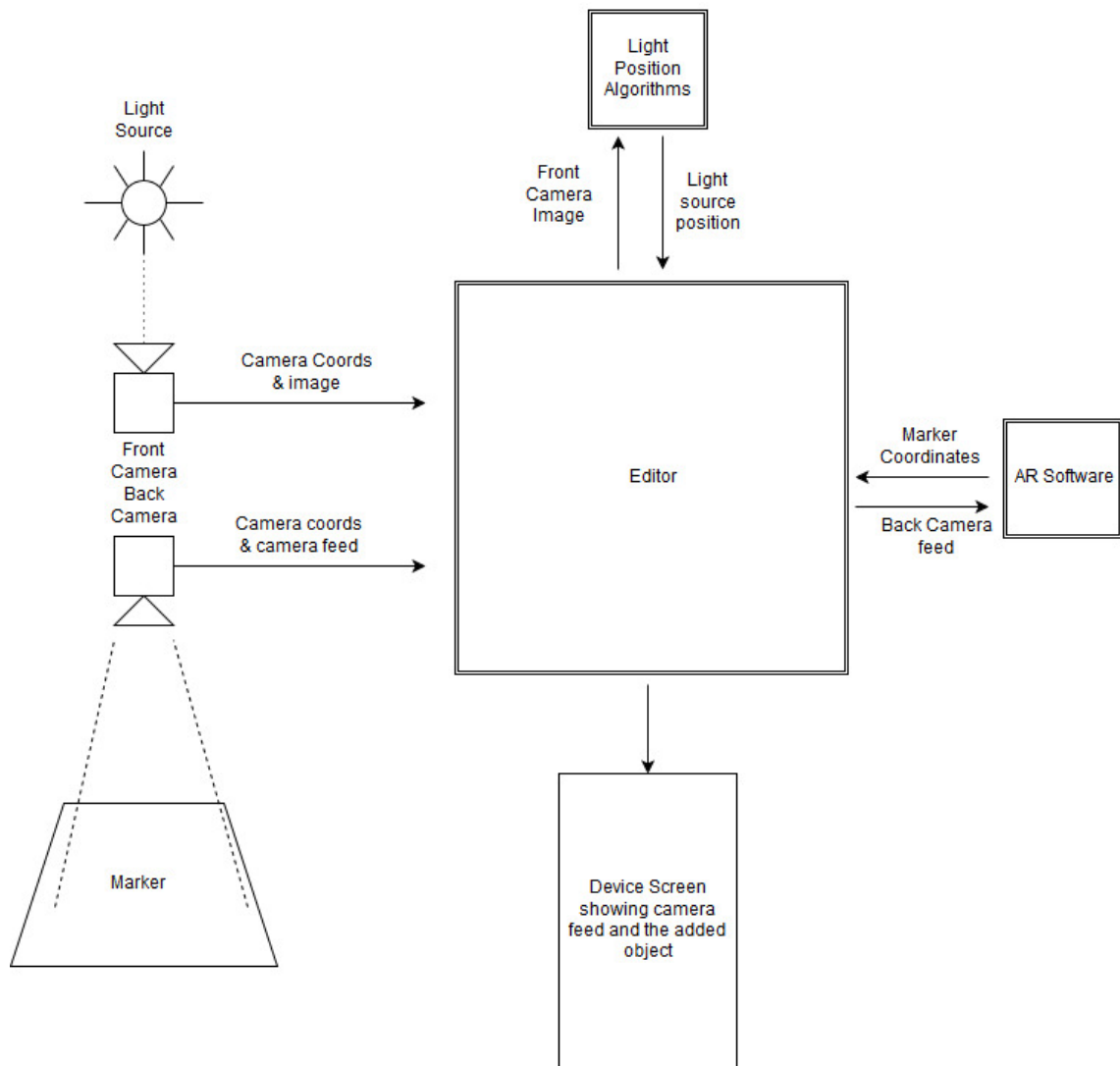
c) Αλγόριθμοι

a. Αλγόριθμος για την εύρεση της θέσης της φωτεινής πηγής.

i. Αλγόριθμος που εξαλείφει το πρόβλημα του noise από την εικόνα.

ii. Αλγόριθμος που επιλύει το πρόβλημα του glare στην εικόνα.

b. Αλγόριθμος που μετατρέπει τη θέση της φωτεινής πηγής από την εικόνα που δόθηκε από την κάμερα σε παγκόσμιες συντεταγμένες για την AR σκηνή.



Σχήμα 3.1 – Σχηματική περιγραφή συστήματος

Κεφάλαιο 4

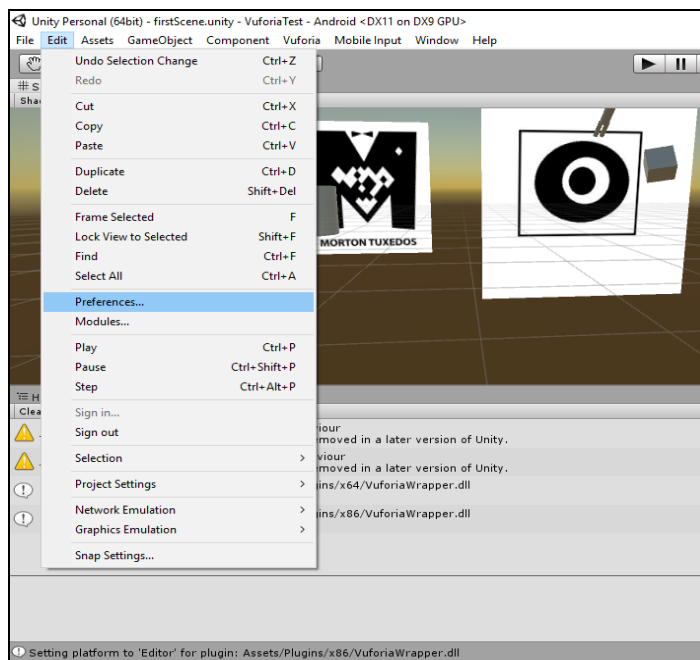
Set Up

4.1 Στήσιμο Unity	23
4.2 Στήσιμο Vuuforia	25
4.3 VuMarks, ImageTargets, Object Recognition	26

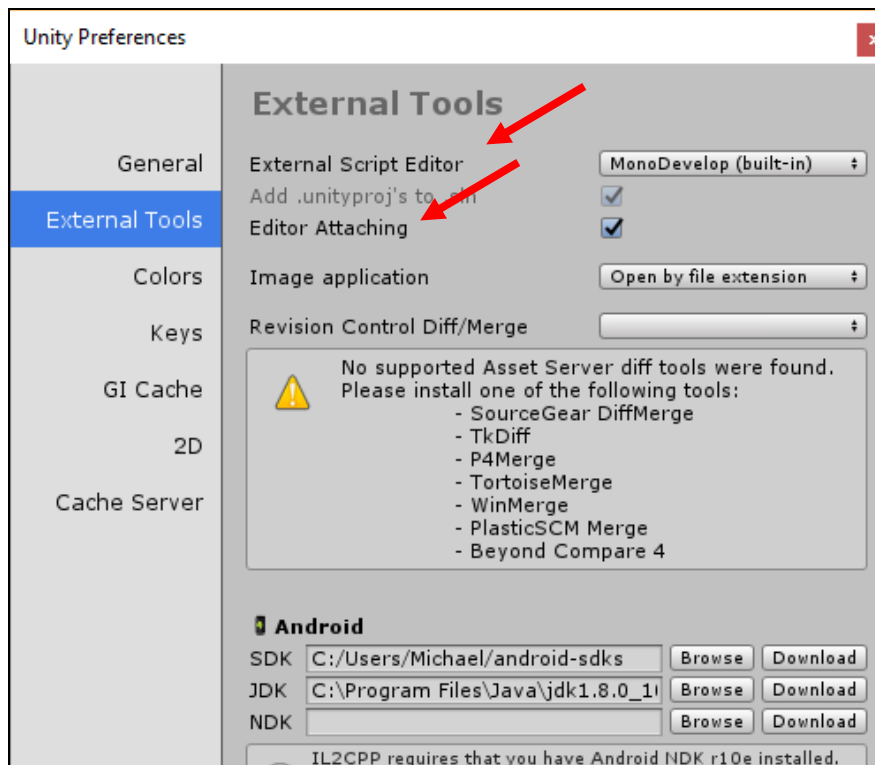
4.1 Στήσιμο Unity

- i. Από την ιστοσελίδα του Unity κατέβασα και εγκατέστησα το Personal Package. Η εγκατάστασή του είναι αρκετά απλή και την αναλαμβάνει σχεδόν εξολοκλήρου το installer τους.
- ii. Από κει και πέρα το τι χρειάζεται να κάνει ο χρήστης είναι να κάνει log in ή sign up πριν του δοθεί πρόσβαση στο περιβάλλον ανάπτυξης.
- iii. Δημιουργεί ένα καινούριο Project.
- iv. Για το Script Editor που θα χρησιμοποιηθεί δίνονται δύο επιλογές από το Unity. Είτε θα γίνει χρήση του Visual Studio ή του MonoDevelop. Εγώ επέλεξα το MonoDevelop όμως και για τις δύο περιπτώσεις ορίζουμε τον script editor με τα εξής βήματα (δεδομένου ότι είναι ήδη εγκατεστημένοι στον υπολογιστή).
Edit → Preferences... (βλέπε Σχήμα 4.1) → Στο External Tools tab επιλέγω τον editor που θέλω από το External Script Editor (βλέπε Σχήμα 4.2) → Ενεργοποιώ το Editor Attaching
- v. Τέλος πρέπει να κάνουμε αλλαγές έτσι ώστε ο compiler να βγάζει σαν executable .apk αρχείο για να μπορεί να εγκατασταθεί σε κινητό. Αυτό γίνεται με τα ακόλουθα βήματα.

File → Build Settings...→ Στο παράθυρο Platform επιλέγω Android → Κάτω από το παράθυρο Platform πατώ το κουμπί Player Settings → Μέσα στο Inspector window που θα ανοίξει μπορώ να δω και να αλλάξω όλα τα settings που έχουν να κάνουν με την εφαρμογή που θα μπει στο κινητό (π.χ. thumbnail, όνομα, orientation κ.λπ.)



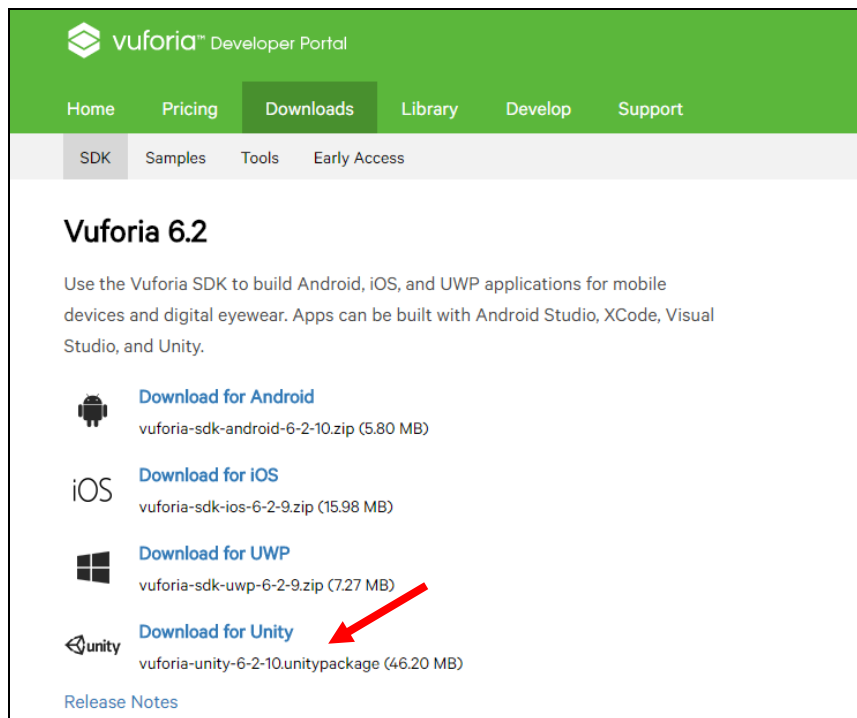
Σχήμα 4.1



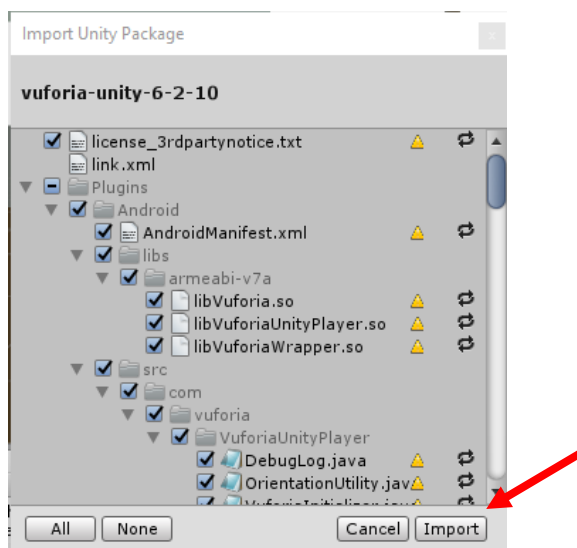
Σχήμα 4.2 – Unity Preferences window

4.2 Στήσιμο Vuforia

- i. Από την ιστοσελίδα του Vuforia πατώ το tab Dev Portal και μετά στο Downloads. Από εκεί κατεβάζω το πακέτο για το Unity (βλέπε Σχήμα 4.3).
- ii. Δημιουργώ ή ανοίγω ένα καινούριο project στο Unity.
- iii. Βρίσκω το πακέτο που κατέβασα και κάνω διπλό κλικ πάνω σ' αυτό.
- iv. Στο παράθυρο που θα ανοίξει μέσα στο Unity πατώ Import (βλέπε Σχήμα 4.4).



Σχήμα 4.3 – Link για κατέβασμα των assets για το Vuforia



Σχήμα 4.4 – Import package window

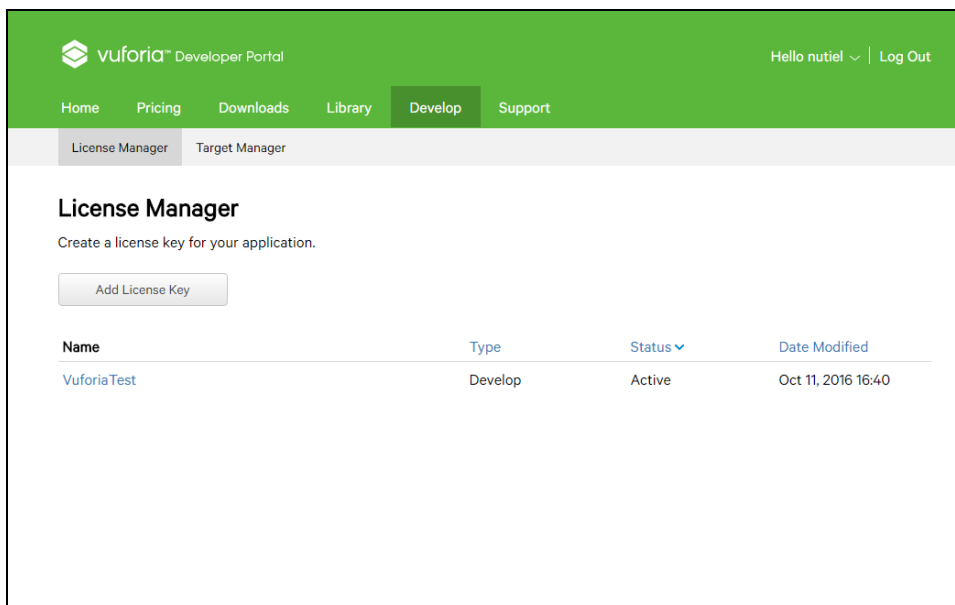
4.3 VuMarks, ImageTargets, Object Recognition

Όπως αναφέρθηκε και πιο πάνω το Vuforia μας δίνει τη δυνατότητα να αναπτύξουμε είτε marker based ή markerless applications. Για τους σκοπούς της Διπλωματικής μου εργασίας επέλεξα να χρησιμοποιήσω markers για τη δική μου εφαρμογή.

Για το σκοπό αυτό στην ιστοσελίδα του Vuforia μέσω του Dev Portal δίνεται στον χρήστη η επιλογή να κάνει Log in στο λογαριασμό του. Από εκεί στο tab Develop υπάρχουν δύο υποσελίδες. Η μία είναι η License Manager και η άλλη η Target Manager μέσω των οποίων μπορούμε να διαχειριστούμε τα License keys και διάφορα assets που θα χρησιμοποιήσουμε στο project μας.

Από το License Manager μπορούμε να προσθέσουμε και να διαχειριστούμε ήδη υπάρχοντα κλειδιά (βλέπε Σχήμα 4.5). Κάθε χρήστης μπορεί να έχει μέχρι 100 κλειδιά και αυτό του δίνει τη δυνατότητα να δημιουργήσει πολλαπλές εφαρμογές ανεξάρτητα τη μια από την άλλη. Κάθε καινούριο project έχει το δικό του License Key (βλέπε Σχήμα 4.6) και επικολλάται πάνω στο ARCamera (βλέπε Σχήμα 4.7) το οποίο είναι ένα από τα assets που εισάγαμε από το πακέτο του Vuforia και μας δίνει τη δυνατότητα να το χρησιμοποιήσουμε.

Από το Target Manager μπορούμε να προσθέσουμε και να διαχειριστούμε διάφορων ειδών marks (βλέπε Σχήμα 4.8), όπως VuMarks, ImageTargets, 3D objects κλπ, τα οποία μπορούμε να κατεβάσουμε σαν packages και να τα προσθέσουμε στο project μας. Αυτό μας επιτρέπει να τα αναγνωρίζουμε μέσω της ARCamera και να τους δοθούν συντεταγμένες μέσα στη σκηνή μας σε σχέση με τη συσκευή μας.



The screenshot shows the Vuforia Developer Portal interface. The top navigation bar is green with the Vuforia logo and the text 'Developer Portal'. On the right, it says 'Hello nutiel' and 'Log Out'. Below the navigation bar, there are tabs for 'Home', 'Pricing', 'Downloads', 'Library', 'Develop', and 'Support'. The 'Develop' tab is selected, and within it, the 'License Manager' sub-tab is active. The 'License Manager' section has a heading 'License Manager' and a sub-heading 'Create a license key for your application.' Below this is a button labeled 'Add License Key'. A table lists the existing license keys:

Name	Type	Status	Date Modified
VuforiaTest	Develop	Active	Oct 11, 2016 16:40

Σχήμα 4.5 – License Manager tab

VuforiaTest
[Edit Name](#)
[Delete License Key](#)

License Key

Usage

Please copy the license key below into your app

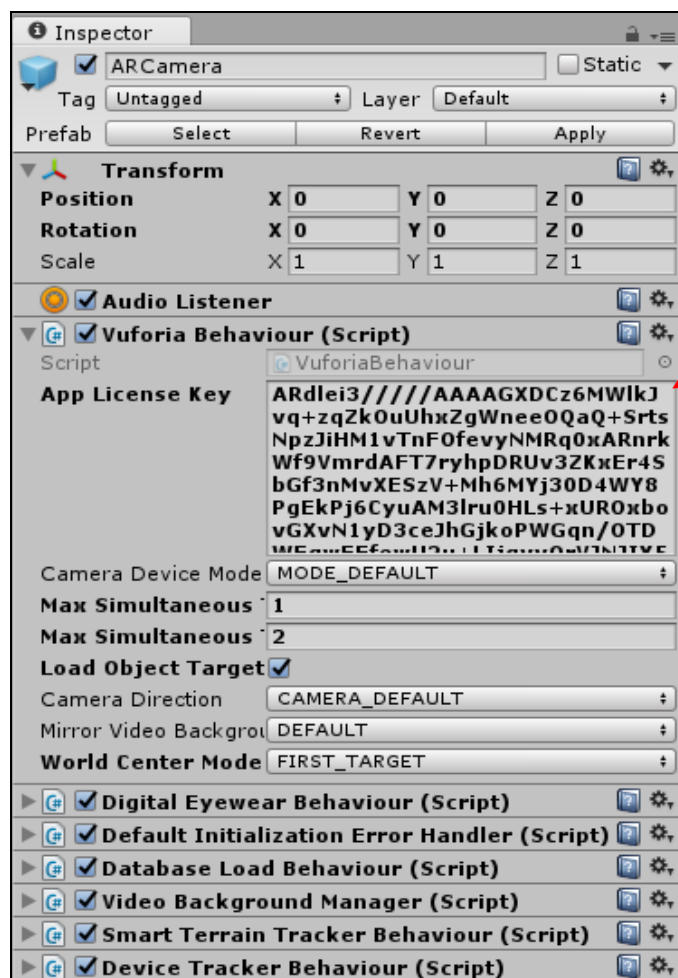
```

ARDlei3/////AAAAGXDCz6MWlkJvq+zqZkOuUhxZgWneeOQaQ+Sr
tsNpzJiHM1vTnFOfevyNMRq0xARnrkWf9VmrdaFT7ryhpDRUv3ZK
xEr4SbGf3nMvXESzV+Mh6MYj30D4WY8PgEkPj6CyuAM3lru0HLs+
xUROxbvGXvN1yD3ceJhGjkoPWGqn/OTDWFgwFEfewU2u+LIjgvv
QrVJNJIY5ID6qcbDFyNj2zIQu7Ft3Enw2pEJ9WEmlu+KCbTjJ4sM
8B4yDvmCCXy1eMrZKzcqrQae3tv2p6MBJ7V3UDJ0CIAC3ECddjTm
PQsNfFbMasSRDOywmQ817ND18nZgm5Ew7sd8t6Wp8Q1zqe0+GH57
URc+iN1Nb0byWRK3

```

Device: Mobile
Type: Develop
Status: Active
Created: Oct 11, 2016 16:40

Σχήμα 4.6 – License key για την ARCamera



Σχήμα 4.7 – Inspector window για ARCamera

 vuforia™ Developer Portal

Hello nutiel  | [Log Out](#)

[Home](#) [Pricing](#) [Downloads](#) [Library](#) [Develop](#) [Support](#)

[License Manager](#) [Target Manager](#)

Target Manager

Use the Target Manager to create and manage databases and targets.

Add Database

Database	Type	Targets	Date Modified
VTest	VuMark	1	Nov 01, 2016 19:37
VTest2	Device	4	Nov 15, 2016 21:11

Σχήμα 4.8 – Target Manager tab

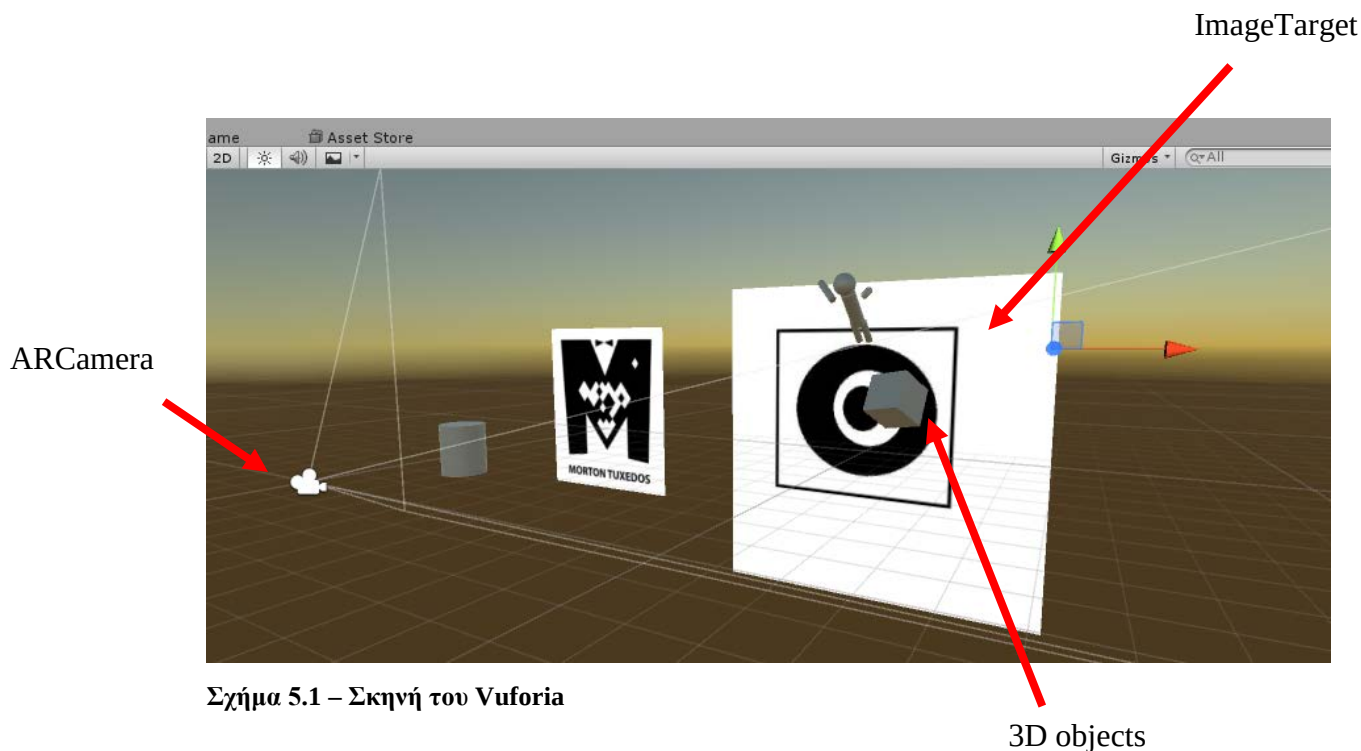
Κεφάλαιο 5

Ανίχνευση & Τοποθέτηση Φωτεινής Πηγής

5.1 Εισαγωγή	30
5.2 Ανίχνευση Φωτεινής Πηγής	31
5.3 Αλγόριθμος Gaussian Blur	32
5.4 Αλγόριθμος Pixelate	35
5.5 Τοποθέτηση Φωτεινής πηγής	37

5.1 Εισαγωγή

Το AR κομμάτι της εφαρμογής, όπως αναφέρθηκε και πιο πάνω, θα υλοποιηθεί εξολοκλήρου από το Vuforia. Το μόνο που έχουμε να κάνουμε από την πλευρά μας είναι να εισάγουμε την ARCamera και τα ImageTargets στη σκηνή και να σχεδιάσουμε είτε ένα δισδιάστατο ή τρισδιάστατο αντικείμενο το οποίο θα εμφανίζεται σε περίπτωση που ανιχνεύσουμε ένα από αυτά τα marks (βλέπε Σχήμα 5.1).



Σχήμα 5.1 – Σκηνή του Vuforia

5.2 Ανίχνευση Φωτεινής Πηγής

Για την ανίχνευση της φωτεινής πηγής παίρνω πρώτα απ' όλα το feed της μπροστινής κάμερας του κινητού και το αποθηκεύω μέσα σε ένα μονοδιάστατο πίνακα. Πριν τον αποθηκεύσω όμως μετατρέπω την RGB τιμή του σε Luminance με τον τύπο

$$\Rightarrow luminance = 0.2126*red + 0.7152*green + 0.0722*blue^{[3]}$$

Αυτός ο τύπος δε μας δίνει ακριβές αποτέλεσμα, για το σκοπό που το θέλουμε όμως είναι αρκετό και πολύ πιο γρήγορο από άλλους τύπου που κάνουν το ίδιο με περισσότερη ακρίβεια.

Αφού μετατρέψουμε με αυτόν τον τύπο όλα τα pixel της εικόνας που λάβαμε σε Luminance μπορούμε με μια γραμμική σάρωση του πίνακα να βρούμε το πιο φωτεινό pixel και με βάση τη θέση του να υπολογίσουμε προσεγγιστικά τη θέση της φωτεινής πηγής στο χώρο, τοποθετώντας την στην ανάλογη θέση στη σκηνή μας.

Το πρόβλημα που προκύπτει όμως με αυτή την τεχνική είναι το ότι πολλές φορές η ποιότητα της μπροστινής κάμερας δεν είναι πολύ καλή και οι εικόνες καταλήγουν να έχουν noise (βλέπε Σχήμα 5.2). Αυτά τα pixel λοιπόν μας δημιουργούν πρόβλημα επειδή αν είναι λευκά θα μας επιστρέψουν την ψηλότερη τιμή σε Luminance και μετά

τη σάρωση το pixel με την ψηλότερη τιμή θα βρίσκεται πάντα προς το τέλος της εικόνας. Αν βασιστούμε σε αυτή την τεχνική μόνο η θέση της φωτεινής πηγής στη σκηνή μας θα διαφέρει με την ανάλογη θέση της στον πραγματικό κόσμο.



Σχήμα 5.2 – Φωτογραφία με θόρυβο

5.3 Αλγόριθμος Gaussian Blur

Λύση στο προαναφερθέν πρόβλημα προσφέρει ο Αλγόριθμος Gaussian Blur. Αυτός ο αλγόριθμος “εξομαλύνει” τις εικόνες λαμβάνοντας υπόψη τα διπλανά pixel. Όσο περισσότερα pixel λάβουμε υπόψη για τον επαναυπολογισμό του κεντρικού, τόσο περισσότερο εξαλείφεται ο θόρυβος από την εικόνα μας, και από την άλλη μεριά τόσο πιο θολή γίνεται (βλέπε Σχήμα 5.3).

Η υλοποίηση αυτού του Αλγορίθμου δεν είναι δύσκολη. Γενικά το τι κάνει σε βήματα είναι:

- Παίρνει κάθε pixel της εικόνας ξεχωριστά.
- Για μια ακτίνα R (μονός αριθμός) παίρνει το κουτί $R \times R$ με κέντρο το pixel στο οποίο βρίσκεται και αναθέτει βάρη στα pixel που βρίσκονται μέσα στο κουτί. (βλέπε Σχήμα 5.5)

- c. Το χρώμα του κάθε pixel πολλαπλασιάζεται με το βάρος του και οι τιμές τους προστίθενται.
- d. Ο αριθμός που πήραμε διαιρείται με το άθροισμα όλων των βαρών που τοποθετήσαμε και το αποτέλεσμα που παίρνουμε είναι η νέα τιμή του κεντρικού pixel.
- e. Αν έχουμε να κάνουμε με εικόνα που χρησιμοποιεί το σύστημα RGB αυτή η διαδικασία πρέπει να γίνει για κάθε κανάλι ξεχωριστά.

Τα βάρη που δίνονται στα pixels είναι ανάλογα με την απόστασή τους από το κεντρικό pixel (βλέπε Σχήμα 5.4), με άλλα λόγια τα αμέσως γειτονικά pixel έχουν μεγαλύτερο αντίκτυπο στον υπολογισμό του νέου χρώματος του κεντρικού απ' ό,τι άλλα που βρίσκονται πιο μακριά.

Το κόστος αυτού του αλγορίθμου είναι αρκετά μεγάλο, $O(N \cdot R^2)$ για την ακρίβεια όπου N είναι το ύψος επί το πλάτος της εικόνας μας και R η ακτίνα που παίρνουμε για κάθε pixel.

Δε θα ήταν καθόλου πρακτικό να συμπεριλάβουμε ένα τέτοιο αλγόριθμο στην εφαρμογή μας αφού ο χρόνος που χρειάζεται για να τελειώσει είναι υπερβολικά μεγάλος, μπορούμε όμως να χρησιμοποιήσουμε μια παραλλαγή αυτού του αλγορίθμου. Η παραλλαγή αυτή δεν έχει επίσημα όνομα και δεν ξέρουμε ποιος είναι ο εμπνευστής της, δημοσιεύτηκε όμως κάτω από το όνομα “Fastest Gaussian Blur (in linear Time)” από τον Ivan Kuckir^[1].

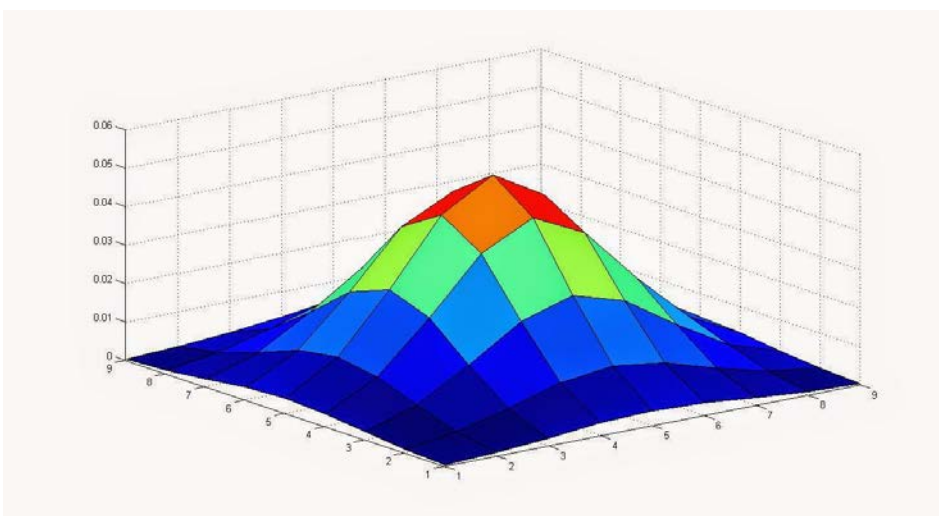
Αυτή η παραλλαγή του αλγορίθμου μας δίνει το ίδιο σχεδόν αποτέλεσμα με τον αρχικό αλλά σε γραμμικό χρόνο, $O(N)$ για την ακρίβεια. Με την καινούρια εκδοχή του αλγορίθμου απλοποιούνται πολλές διαδικασίες και εν τέλει καταφέρνουμε να μειώσουμε την πολυπλοκότητα του αλγορίθμου αρκετά έτσι ώστε να μας είναι δυνατό να τον χρησιμοποιήσουμε για την εφαρμογή μας.

Μερικές από τις αλλαγές που έγιναν είναι η αλλαγή της μεθόδου που καθορίζονται τα βάρη. Με το νέο αλγόριθμο το βάρος παραμένει ίδιο για όλα τα pixel έχουμε όμως πολλά περάσματα και μικρότερες περιοχές να καλύψουμε έτσι καταλήγουμε με μία

αρκετά καλή προσέγγιση στον αρχικό αλγόριθμο. Μετά εισάγοντας τον όρο Horizontal και Total Blur, τα οποία εξηγούνται καλύτερα στο blog του ανθρώπου που το δημοσίευσε ρίχνουμε ακόμα περισσότερο την πολυπλοκότητα μέχρι αυτή να γίνει γραμμική. Τέλος αν παρατηρήσουμε γειτονικές τιμές στο Horizontal Blur βλέπουμε ότι είναι σχεδόν ίδιες και αυτό μας δίνει τη δυνατότητα να επαναχρησιμοποιήσουμε τιμές από προηγούμενο πέρασμα και να μην χρειαστεί να κάνουμε ξανά τους υπολογισμούς. Η διαδικασία με την οποία καταλήξαμε σε αυτή την παραλλαγή του αλγορίθμου φαίνεται στο Παράρτημα Α.



Σχήμα 5.3 – Δεξιά, κανονική εικόνα φωτεινής πηγής, Αριστερά, η ίδια εικόνα με φίλτρο Gaussian Blur ακτίνας 11



Σχήμα 5.4 – Γενική ιδέα για τον υπολογισμό τιμής ενός pixel με Gaussian Blur (η κλίμακα δεν έχει να κάνει με τον αλγόριθμο)

$$\frac{1}{159}$$

2	4	5	4	2
4	9	12	9	4
5	12	15	12	5
4	9	12	9	4
2	4	5	4	2

Σχήμα 5.5 – Βάρη που τοποθετούνται τυπικά στα pixel γύρω από το κεντρικό για τον υπολογισμό της τιμής του

5.4 Αλγόριθμος Pixelate

Με τον πιο πάνω αλγόριθμο λύσαμε το πρόβλημα του θορύβου στην εικόνα, παρόλα αυτά παραμένει ένα ακόμα πρόβλημα να λύσουμε, το πρόβλημα του glare από φωτεινές πηγές (βλέπε Σχήμα 5.6).

Όταν η φωτεινή πηγή μας βρίσκεται εκτός του οπτικού πεδίου της κάμεράς μας είμαστε εντάξει. Όταν όμως περάσει εντός του οπτικού πεδίου η αντανάκλαση πάνω στο γυαλί δημιουργεί αυτό το glare effect το οποίο μας δημιουργεί προβλήματα στον εντοπισμό της θέσης της πηγής του φωτός. Πιο συγκεκριμένα, ακόμα κι αν χρησιμοποιήσουμε τον Gaussian Blur για να εξαλείψουμε το θόρυβο, οι ακτίνες που δημιουργούνται από το glare δεν είναι θόρυβος αφού δε μιλάμε πλέον για μεμονωμένα pixel με αλλοιωμένο χρώμα αλλά για συνεχόμενα pixel των οποίων το χρώμα λήφθηκε έτσι ακριβώς από τη κάμερα.

Από τη στιγμή λοιπόν που ο Gaussian Blur δε μπορεί να μας βοηθήσει στρεφόμεστε σε έναν πιο απλό, όμως πιο ταιριαστό στη συγκεκριμένη περίπτωση, αλγόριθμο τον Pixelate.

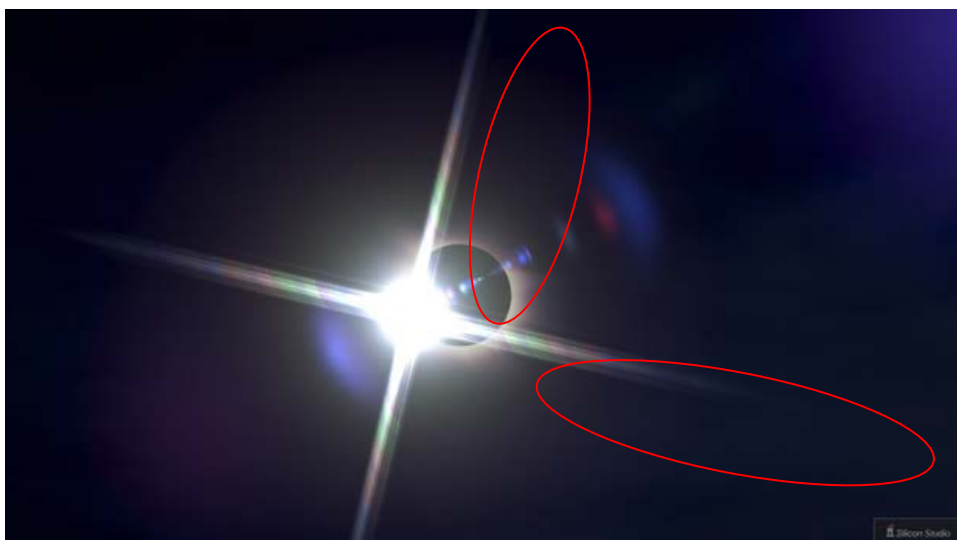
Ο Pixelate κάνει κάτι πολύ πιο απλό από τον Gaussian Blur. Αντί να ασχολείται με ένα-ένα τα pixel της εικόνας, τα επεξεργάζεται σε ομάδες και τα αντικαθιστά με ένα μεγαλύτερο για κάθε ομάδα και δίνοντάς του τον μέσο όρο χρώματος που είχαν τα pixel που αντικατέστησε (βλέπε Σχήμα 5.7).

Η βηματική περιγραφή αυτού του αλγορίθμου έχει ως εξής:

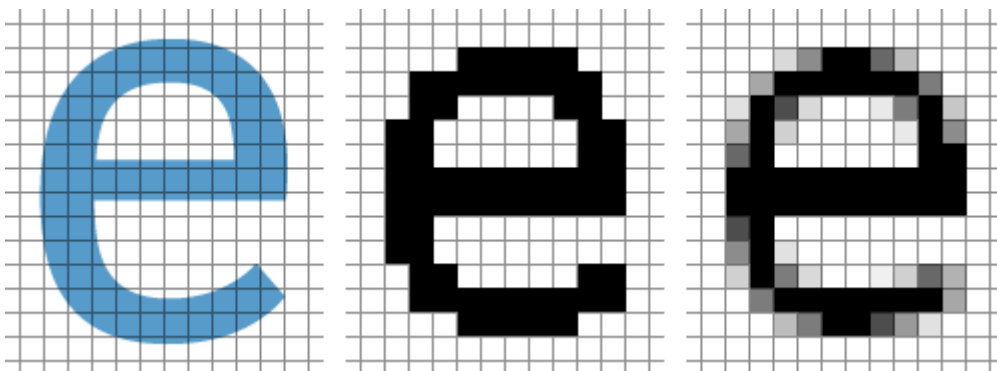
- a. Αρχίζω από το πρώτο pixel μιας εικόνας $N \times M$ και παίρνω ένα κουτί $Y \times Y$ του οποίου το πρώτο pixel είναι επίσης το πρώτο της εικόνας.
- b. Για κάθε pixel μέσα στο κουτί που πήρα αθροίζω την τιμή του χρώματός τους και διαιρώ δια $Y \times Y$. Αυτό θα μας δώσει το μέσο όρο όλων των χρωμάτων μέσα στο κουτί.
- c. Αναθέτω την τιμή που πήρα σαν το χρώμα του μεγάλου pixel που πήρε τώρα τη θέση των υπολοίπων. Σε περίπτωση που δε μπορώ να μεγαλώσω το μέγεθος των pixel απλώς αναθέτω την τιμή αυτή σε όλα τα pixel μέσα στο κουτί.
- d. Παίρνω το επόμενο pixel Y θέσεις πιο μετά και έχοντας εκείνο ως το πρώτο pixel επαναλαμβάνω τα βήματα από το a. αλλάζοντας γραμμή όποτε χρειάζεται μέχρι να τελειώσουν τα pixel.
- e. Αν έχουμε να κάνουμε με εικόνα που χρησιμοποιεί το σύστημα RGB αυτή η διαδικασία πρέπει να γίνει για κάθε κανάλι ξεχωριστά.

Ο αλγόριθμος αυτός μπορεί να είναι πιο απλός, ταιριάζει όμως καλύτερα στις ανάγκες μας αφού με αρκετά μεγάλο βαθμό pixelation μπορούμε να βρούμε το σημείο στην εικόνα με τα περισσότερα φωτεινά pixel (βλέπε Σχήμα 5.8) και να πάρουμε μια αρκετά καλή ιδέα για το πού περίπου βρίσκεται η φωτεινή πηγή.

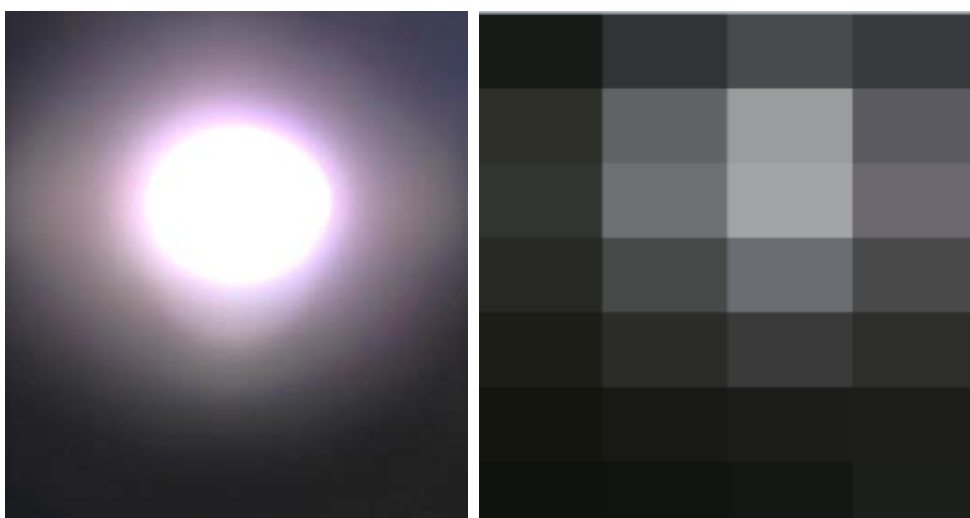
Ο Pixelate έχει χρονική πολυπλοκότητα $O(N)$ όπου N είναι το μήκος επί το πλάτος της εικόνας που μας δίνεται, μπορεί λοιπόν να εφαρμοστεί εύκολα στα πλαίσια της εκτέλεσης της εφαρμογής μας χωρίς να επιβαρύνει υπερβολικά το χρόνο εκτέλεσης.



Σχήμα 5.6 – Παράδειγμα glare φωτεινής πηγής σε φωτογραφία



Σχήμα 5.7 – Παράδειγμα αλγορίθμου Pixelate (αριστερά είναι η original εικόνα, στη μέση εμφανίζονται μόνο τα pixel που είναι γεμάτα και δεξιά το αποτέλεσμα του Pixelate)



Σχήμα 5.8 – Αριστερά, η κανονική εικόνα της φωτεινής πηγής. Δεξιά, η ίδια εικόνα με 50 φορές μεγαλύτερα pixels

5.5 Τοποθέτηση Φωτεινής Πηγής

Με το τέλος του αλγορίθμου Pixelate καταλήγουμε με μια καινούρια εικόνα η οποία αποτελείται από ένα αριθμό μεγαλύτερων pixel το χρώμα των οποίων αντιπροσωπεύει τον μέσο όρο των χρωμάτων που υπήρχαν σε εκείνη τη περιοχή στην αρχική εικόνα.

Ο αλγόριθμος για την εύρεση της θέσης της φωτεινής πηγής αποτελείται από τα ακόλουθα βήματα:

- a. Για τη δική μου εφαρμογή έσπασα την εικόνα σε 49 pixel (7x7).
- b. Με ένα γραμμικό έλεγχο βρίσκω τη θέση του πιο φωτεινού pixel
- c. Η θέση της φωτεινής πηγής σε παγκόσμιες συντεταγμένες ορίζεται ως (c_i+2*i, c_j+2*j) όπου (c_i, c_j) οι παγκόσμιες συντεταγμένες της κάμερας. (Σημείωση: Οι συντεταγμένες, i, j του κάθε pixel ορίζονται ως 0x0 για το κεντρικό pixel, το i αυξάνεται προς τα πάνω και μειώνεται προς τα κάτω, το j αυξάνεται προς τα δεξιά και μειώνεται προς τα αριστερά)

Από τη στιγμή που χρησιμοποιούμε παγκόσμιες συντεταγμένες, δε χρειάζεται να ανησυχούμε για τη μετατροπή της θέσης των αντικειμένων για το συσχετισμό τους με άλλα αντικείμενα και με τη χρήση του mark που ανιχνεύσαμε ως το κέντρο του κόσμου μας ξέρουμε ότι η φωτεινή πηγή μας θα μένει σταθερή στη σκηνή μας ακόμα και αν μετακινηθεί η κάμερα.

Κεφάλαιο 6

Αποτελέσματα

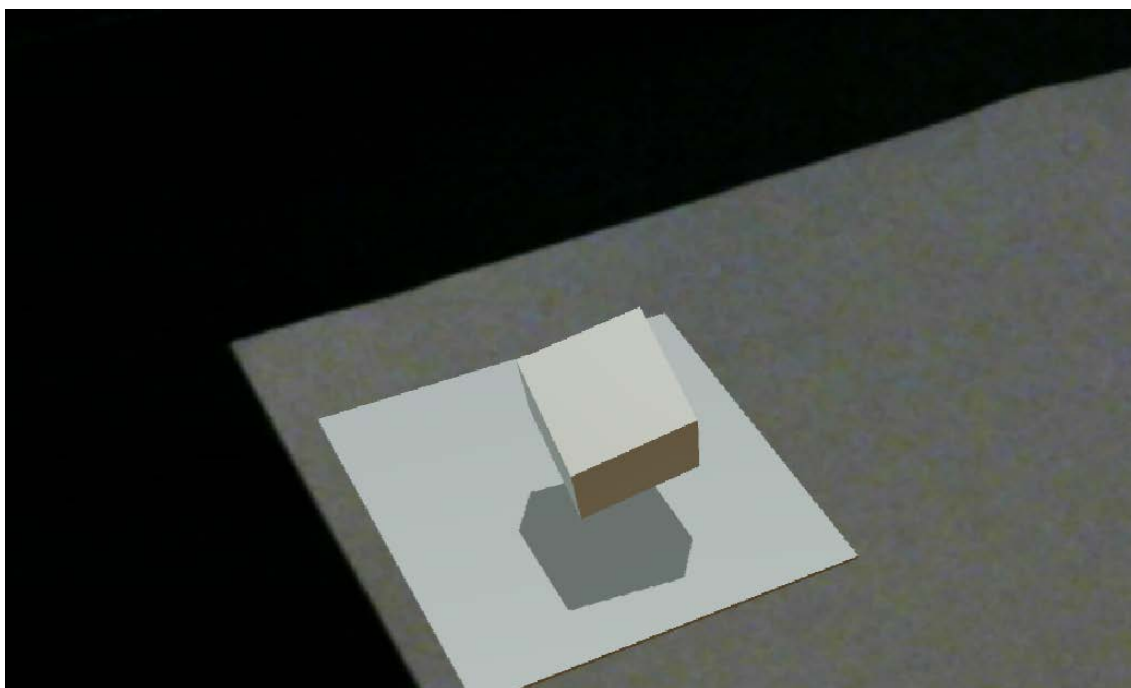
6.1 Προσθήκη Φωτισμού	39
6.2 Ανάλυση Gaussian Blur	41
6.3 Ανάλυση Pixelate	42
6.4 Σύγκριση Gaussian Blur & Pixelate	42

6.1 Προσθήκη Φωτισμού

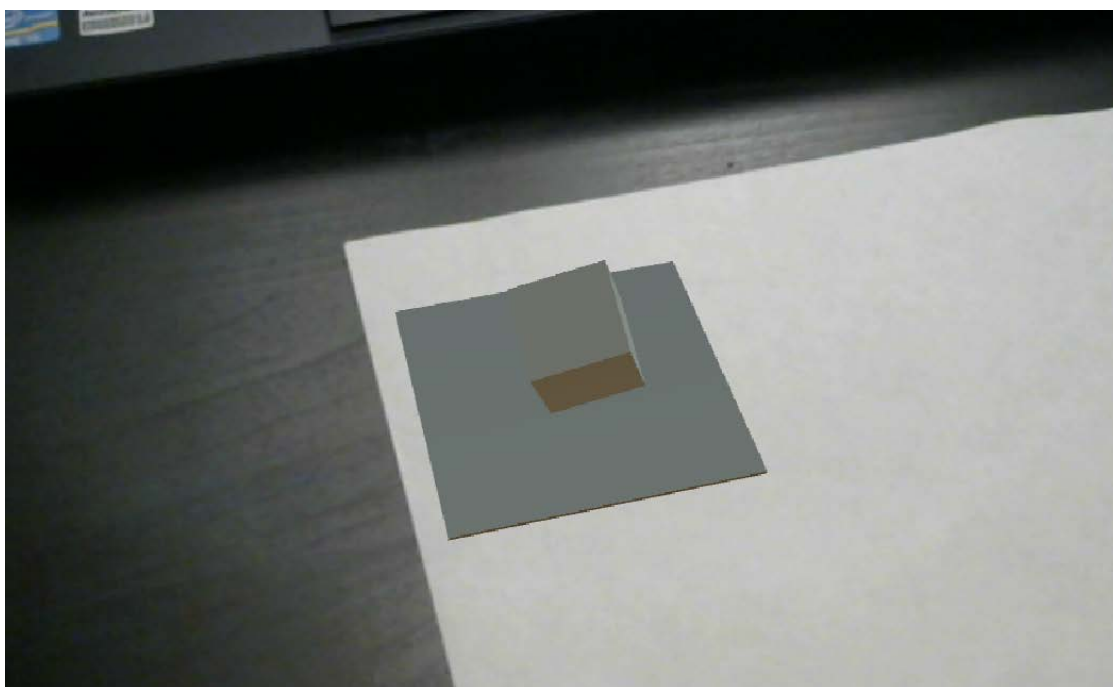
Η προσθήκη φωτισμού σε ένα τέτοιο σύστημα είναι πολύ σημαντικός παράγοντας ως προς την αίσθηση που θέλουμε να δώσουμε στο χρήστη. Ο λόγος που είναι τόσο σημαντικός ο σωστός φωτισμός είναι το ότι το φώς και οι σκιές είναι κάτι που βιώνουμε στην καθημερινότητά μας, πολλές φορές χωρίς να το καταλάβουμε. Μας βοηθούν στην κατανόηση του σχήματος και πιο μικρών λεπτομεριών στην τοπολογία της επιφάνειάς τους. Επίσης οι σκιές και ο τρόπος που φωτίζονται κάποια αντικείμενα σε σχέση με άλλα μας βοηθούν να κατανοήσουμε τη θέση τους στο χώρο και μερικές φορές την απόστασή τους από εμάς.

Η διαφορά μεταξύ μιας σκηνής που φωτίζεται από κάποια πηγή φωτός (Βλέπε Σχήμα 6.1) και μιας σκηνής χωρίς να δοθεί ιδιαίτερη σημασία στο φωτισμό και χωρίς Shadow Casting (Βλέπε Σχήμα 6.2) είναι πολύ εμφανής.

Σημείωση: Στο σχήμα 6.2 χρησιμοποιήθηκε ένα πολύ απλό μοντέλο φωτισμού το οποίο δεν δίνει σημασία στις σκιές για να φαίνονται τα αντικείμενα, διαφορετικά το μόνο που θα βλέπαμε είναι ένα μαύρο κουτί.



Σχήμα 6.1 – Σκηνή με φωτισμό και Shadow Casting



Σχήμα 6.2 – Σκηνή χωρίς φωτισμό και Shadow Casting

6.2 Ανάλυση Gaussian Blur

Όπως αναφέραμε και πιο πάνω ο Αλγόριθμος Gaussian Blur εκτελείται κανονικά σε χρόνο $O(N \cdot R^2)$, εξαρτάται δηλαδή από N ο αριθμός των pixel της εικόνας πάνω στην οποία εφαρμόζουμε το αλγόριθμο, και R η απόσταση σε pixel από το κεντρικό pixel που θα λάβουμε υπόψη για τον υπολογισμό του χρώματός του.

Με τη χρήση ενός πιο βέλτιστου αλγορίθμου καταφέραμε να ρίξουμε την πολυπλοκότητα σε $O(N)$, εξαλείψαμε δηλαδή την εξάρτηση του αλγορίθμου από το R και μπορούμε να πάρουμε ότι μέγεθος ακτίνας θέλουμε.

Ακολουθεί πίνακας (Βλέπε Πίνακα 6.1) με το χρόνο που χρειάστηκε ο Gaussian Blur για να τερματίσει. Οι πάρθηκαν πολλές μετρήσεις για κάθε περίπτωση και οι περιπτώσεις μας είναι η ακτίνα R που χρησιμοποιούμε.

#\rad	3	5	7	9	11	13	15	17	19
1	128	128	126	129	143	132	126	133	128
2	130	131	127	133	144	130	128	129	125
3	128	129	124	132	143	132	126	127	126
4	129	127	125	129	147	131	128	126	127
5	127	129	129	133	145	133	132	126	130
6	127	130	125	129	143	132	131	125	130
7	129	127	125	129	143	130	133	125	126
8	130	132	124	128	145	129	134	125	127
9	127	130	124	126	145	131	128	125	127
10	127	127	125	128	144	128	133	125	125
Avg	128.2	129	125.4	129.6	144.2	130.8	129.9	126.6	127.1

Πίνακας 6.1 – Πίνακας με πειραματικές μετρήσεις του αλγορίθμου Gaussian Blur

6.3 Ανάλυση Pixelate

Ο Αλγόριθμος Pixelate έχει και αυτός πολυπλοκότητα $O(N)$ αρα δεν έχει σημασία το μέγεθος των pixel που θα καταλήξει να έχει η εικόνα αφού θα περάσουμε από το κάθε pixel της αρχικής εικόνας μόνο μια φορά σε όλες τις περιπτώσεις.

Ακολουθεί πίνακας (Βλέπε Πίνακα 6.2) με το χρόνο που χρειάστηκε ο Pixelate για να τερματίσει. Η μεθοδολογία μας είναι ίδια με πιο πάνω με τη διαφορά ότι οι περιπτώσεις μας είναι ο αριθμός των pixel τον οποίο καταλήγει να έχει η τελική μας εικόνα.

#\rad	9	16	25	36	49	64	81	100	121
1	19	20	20	20	21	21	19	25	19
2	20	20	20	22	20	20	21	21	20
3	19	20	20	21	20	21	20	20	19
4	19	20	20	20	19	22	20	21	20
5	20	21	19	20	20	20	19	26	20
6	21	20	20	19	20	20	20	21	20
7	20	20	21	21	21	21	20	26	20
8	20	20	20	20	20	20	20	30	19
9	20	19	20	20	20	21	21	23	19
10	20	20	21	20	20	21	19	20	19
Avg	19.8	20	20.1	20.3	20.1	20.7	19.9	23.3	19.5

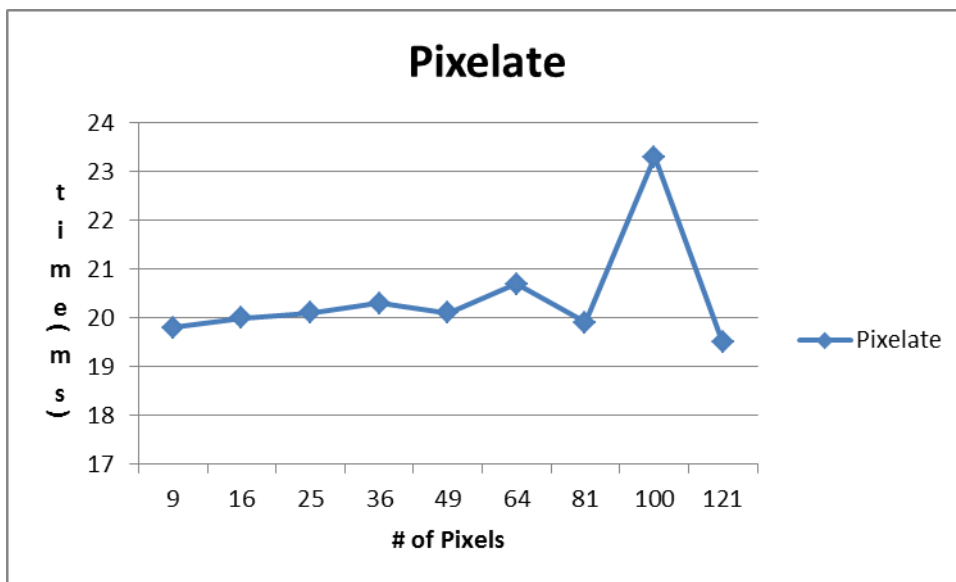
Πίνακας 6.2 – Πίνακας με πειραματικές μετρήσεις του αλγορίθμου Pixelate

6.4 Σύγκριση Gaussian Blur & Pixelate

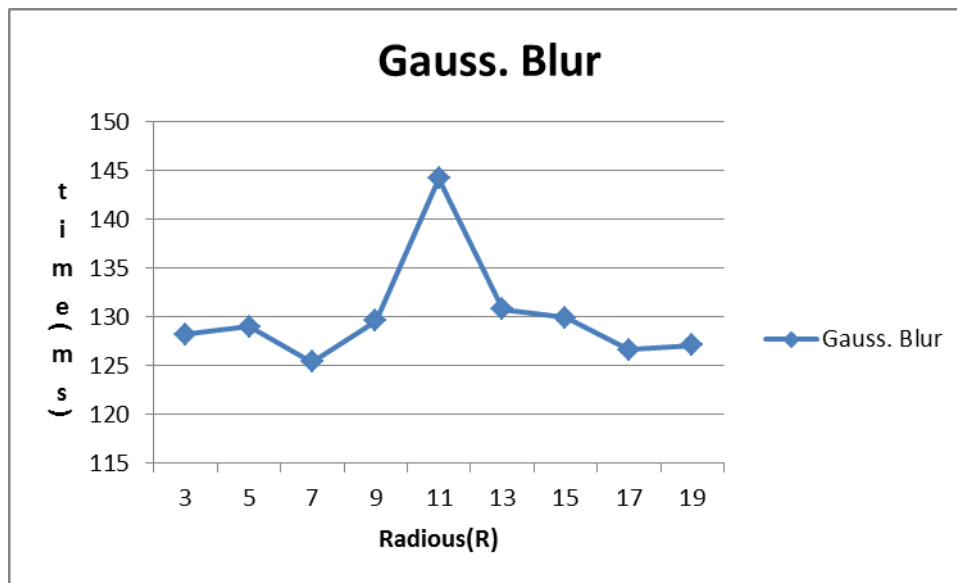
Στο τέλος της ημέρας ο αλγόριθμος Pixelate, αν και πιο απλός στην υλοποίησή του, αποδείχτηκε να εξυπηρετεί καλύτερα τις ανάγκες μας από τον Gaussian Blur. Το ερώτημα που μένει να απαντήσουμε είναι κατά πόσο οι δύο αυτοί αλγόριθμοι θα μας παράξουν καλύτερο αποτέλεσμα αν τους χρησιμοποιήσουμε σε συνδυασμό παρά τον

καθένα μόνο του. Με άλλα λόγια να τρέξουμε πρώτα τον Gaussian Blur για να καθαρίσει η εικόνα από το θόρυβο και μετά να τρέξουμε τον Pixelate για να λύσουμε το πρόβλημα του glare και εν τέλει να βρούμε τη θέση της φωτεινής πηγής.

Μετά από αρκετές δοκιμές φάνηκε ότι ο αλγόριθμος Pixelate είναι αρκετός για να κάνει τη δουλειά που θέλουμε. Τα αποτελέσματα είναι αρκετά σταθερά, αν και όχι τέλεια, οπότε δεν υπάρχει λόγος να επιβαρύνουμε τη εφαρμογή με περιττούς υπολογισμούς. Ο Gaussian Blur αν και έχει σταθερό χρόνο εκτέλεσης χρειάζεται μερικά milliseconds περισσότερα από τον Pixelate (Βλέπε Σχήμα 6.3 και Σχήμα 6.4), και αν και δεν είναι πολύ μεγάλη διαφορά, σε μια εφαρμογή που αλληλεπιδρά με τον χρήστη σε πραγματικό χρόνο είναι αρκετό για να φανεί η διαφορά.



Σχήμα 6.3 – Γράφος στοιχείων Πίνακα 6.2



Σχήμα 6.4 – Γράφος στοιχείων Πίνακα 6.1

Κεφάλαιο 7

Συμπεράσματα

Εν τέλει δημιουργήσαμε μια εφαρμογή για κινητό η οποία αναγνωρίζει ένα marker μέσω της κάμερας και τοποθετεί, με τη βοήθεια του marker και λογισμικού AR, ένα αντικείμενο στο χώρο. Στη συνέχεια βρίσκει τη φωτεινή πηγή στο δωμάτιο και φωτίζει το αντικείμενο ανάλογα.

Για καλύτερο υπολογισμό της θέσης της φωτεινής πηγής χρησιμοποιήσαμε τον αλγόριθμο Pixelate και μελετήσαμε την εναλλακτική του Gaussian Blur την οποία και απορρίψαμε αφού δεν ταίριαζει σε αυτή την περίπτωση.

Το πρόγραμμα δουλεύει όπως σχεδιάστηκε στην αρχή αν και η ακριβής θέση της φωτεινής πηγής μερικές φορές είναι δύσκολο να εντοπιστεί, ειδικά μέσα σε δωμάτια με ανοιχτόχρωμο χρώμα τοίχων, αφού ο τύπος του Luminance που χρησιμοποιούμε βασίζεται στο χρώμα των pixel. Αυτό θα μπορούσε να διορθωθεί με το να ζητήσουμε από τον χρήστη να γυρίσει τη συσκευή προς τη φωτεινή πηγή ή με κάποιον ποιο εξειδικευμένο αλγόριθμο, σε μια τέτοια περίπτωση όμως πρέπει να είμαστε προετοιμασμένοι να θυσιάσουμε απόδοση.

Στο μέλλον αυτή η εφαρμογή θα μπορούσε να αναπτυχθεί περαιτέρω με το να αναγνωρίζει περισσότερες από μια φωτεινές πηγές, αφού έτσι φωτίζονται πολλοί χώροι στην πραγματικότητα.

Βιβλιογραφία

- [1] Ivan Kuckir (Иван Куцкир - 'Cootskeer'), “Fastest Gaussian Blur (in linear time)”, blog.ivank.net/fastest-gaussian-blur.html
- [2] L. Frank Baum, 1901, Συγγραφέας που αναφέρει για πρώτη φορά την προβολή πληροφοριών σαν ολογράμματα στον πραγματικό κόσμο.
- [3] Maciej Laskowski, “Detection of light sources in digital photographs”
- [4] M.Slater, A. Steed and Y. Chrysanthou, (2001), Computer Graphics and Virtual Environments: From Realism to Real-Time, Addison Wesley publishers, ISBN 0-201-62420-6.

Παράρτημα Α

Ακολουθεί απόσπασμα από τη δημοσίευση της λύσης του αλγορίθμου για το Gaussian Blur. Ολόκληρη η ανάλυση μπορεί να βρεθεί στην ιστοσελίδα [https://en.wikipedia.org/wiki/Luma_\(video\)](https://en.wikipedia.org/wiki/Luma_(video))

Definition

The convolution of two 2D functions f and g is defined as the volume of product of f and "shifted" g . The second function g is sometimes called "weight", since it determines, how much of f will get into the result

The Gaussian blur of a 2D function can be defined as a convolution of that function with 2D Gaussian function. Our gaussian function has an integral 1 (volume under surface) and is uniquely defined by one parameter σ called standard deviation. We will also call it "radius" in the text below.

In our discrete finite case, we represent our 2D functions as matrices of values. We compute the volume (integral) as a sum. Gaussian function has near to zero values behind some radius, so we will use only the values $-r \leq x \leq r$, $-r \leq y \leq r$. This "useful" part of weight is also called the **kernel**. The value of convolution at $[i, j]$ is the weighted average, i. e. sum of function values around $[i, j]$ multiplied by weight.

Algorithm 1

For a general discrete convolution of f and weight function w , we can compute the result b as:

$$b[i,j] = \sum_{y=i-r}^{i+r} \sum_{x=j-r}^{j+r} f[y,x] * w[y,x] \quad b[i,j] = \sum_{y=i-r}^{i+r} \sum_{x=j-r}^{j+r} f[y,x] * w[y,x]$$

For gaussian weight, we can compute only weights around $[i, j]$ (area of $4 \cdot r^2$). When our matrix has n values, the time complexity is $O(n \cdot r^2)$. For large radii, e. g. $r=10$, we have to do $n \cdot 400$ operations, which correspond to 400 loops over the whole matrix and that is ugly.

```
// source channel, target channel, width, height, radius
function gaussBlur_1 (scl, tcl, w, h, r) {
    var rs = Math.ceil(r * 2.57); // significant radius
    for(var i=0; i<h; i++)
        for(var j=0; j<w; j++) {
```

```

        var val = 0, wsum = 0;
        for(var iy = i-rs; iy<i+rs+1; iy++)
            for(var ix = j-rs; ix<j+rs+1; ix++) {
                var x = Math.min(w-1, Math.max(0, ix));
                var y = Math.min(h-1, Math.max(0, iy));
                var dsq = (ix-j)*(ix-j)+(iy-i)*(iy-i);
                var wght = Math.exp( -
dsq / (2*r*r) ) / (Math.PI*2*r*r);
                val += scl[y*w+x] * wght; wsum += wght;
            }
        tcl[i*w+j] = Math.round(val/wsum);
    }
}

```

Algorithm 2

Let's introduce the **box blur**. It is the convolution of function f and weight w , but weight is constant and lies within a square (box). The nice feature of box blur is, that when you have some weight function having the same variance, it converges to gaussian blur after several passes.

In this algorithm, we will simulate the gaussian blur with 3 passes of box blur. Let's denote the half of size of square as br ("box radius"). The constant value of weight is $1/(2 \cdot br)^2$ (so the sum over the whole weight is 1). We can define box blur as:

$$bb[i,j] = \sum_{y=i-br}^{i+br} \sum_{x=j-br}^{j+br} f[y,x] / (2 \cdot br)^2$$

We have to convert the standard deviation of gaussian blur r into dimensions of boxes for box blur. I am not very good at calculus, but fortunately I have found [this website](#) and used their implementation.

```

function boxesForGauss(sigma, n) // standard deviation, number of
boxes
{
    var wIdeal = Math.sqrt((12*sigma*sigma/n)+1); // Ideal averaging
filter width
    var wl = Math.floor(wIdeal); if(wl%2==0) wl--;
    var wu = wl+2;

    var mIdeal = (12*sigma*sigma - n*wl*wl - 4*n*wl - 3*n)/(-4*wl -
4);
    var m = Math.round(mIdeal);
    // var sigmaActual = Math.sqrt( (m*wl*wl + (n-m)*wu*wu - n)/12 );

    var sizes = []; for(var i=0; i<n; i++) sizes.push(i<m?wl:wu);
    return sizes;
}

```

Our algorithm has still the same complexity $O(n \cdot r^2)$, but it has two advantages: first, the area is much smaller (br is almost equal to σ , while significant radius for gaussian is much larger). The second advantage is, that the weight is constant. Even though we have to do it 3 times, it performs faster.

```
function gaussBlur_2 (scl, tcl, w, h, r) {
    var bxs = boxesForGauss(r, 3);
    boxBlur_2 (scl, tcl, w, h, (bxs[0]-1)/2);
    boxBlur_2 (tcl, scl, w, h, (bxs[1]-1)/2);
    boxBlur_2 (scl, tcl, w, h, (bxs[2]-1)/2);
}
function boxBlur_2 (scl, tcl, w, h, r) {
    for(var i=0; i<h; i++)
        for(var j=0; j<w; j++) {
            var val = 0;
            for(var iy=i-r; iy<i+r+1; iy++)
                for(var ix=j-r; ix<j+r+1; ix++) {
                    var x = Math.min(w-1, Math.max(0, ix));
                    var y = Math.min(h-1, Math.max(0, iy));
                    val += scl[y*w+x];
                }
            tcl[i*w+j] = val/((r+r+1)*(r+r+1));
        }
}
```

Algorithm 3

We have already simplified gaussian blur into 3 passes of box blur. Let's do a little experiment. Let's define a **horizontal blur** and **total blur**:

$$bh[i,j] = \sum_{x=j-brj+br}^{j+br} f[i,x] / (2 \cdot br) \quad bt[i,j] = \sum_{y=j-brj+br}^{j+br} bh[y,j] / (2 \cdot br) \\ bh[i,j] = \sum_{x=j-brj+br}^{j+br} f[i,x] / (2 \cdot br) \quad bt[i,j] = \sum_{y=j-brj+br}^{j+br} bh[y,j] / (2 \cdot br)$$

Those two functions are "looping" in a line, producing "one-dimensional blur". Let's see, what total blur corresponds to:

$$bt[i,j] = \sum_{y=i-br}^{i+br} bh[y,j] / (2 \cdot br) = \sum_{y=j-br}^{j+br} \left(\sum_{x=j-br}^{j+br} f[y,x] / (2 \cdot br) \right) / (2 \cdot br) = \sum_y \\ = \sum_{y=i-br}^{i+br} \sum_{x=j-br}^{j+br} f[y,x] / (2 \cdot br)^2 \quad bt[i,j] = \sum_{y=i-br}^{i+br} bh[y,j] / (2 \cdot br) = \sum_{y=j-br}^{j+br} \left(\sum_{x=j-br}^{j+br} f[y,x] / (2 \cdot br) \right) / (2 \cdot br) = \sum_{y=i-br}^{i+br} \sum_{x=j-br}^{j+br} f[y,x] / (2 \cdot br)^2$$

We just discovered, that our total blur is box blur! Both total blur and horizontal blur have a complexity $O(n \cdot r)O(n \cdot r)$, so the whole box blur has $O(n \cdot r)O(n \cdot r)$.

```
function gaussBlur_3 (scl, tcl, w, h, r) {
    var bxs = boxesForGauss(r, 3);
    boxBlur_3 (scl, tcl, w, h, (bxs[0]-1)/2);
```

```

        boxBlur_3 (tcl, scl, w, h, (bxs[1]-1)/2);
        boxBlur_3 (scl, tcl, w, h, (bxs[2]-1)/2);
    }
    function boxBlur_3 (scl, tcl, w, h, r) {
        for(var i=0; i<scl.length; i++) tcl[i] = scl[i];
        boxBlurH_3(tcl, scl, w, h, r);
        boxBlurT_3(scl, tcl, w, h, r);
    }
    function boxBlurH_3 (scl, tcl, w, h, r) {
        for(var i=0; i<h; i++)
            for(var j=0; j<w; j++) {
                var val = 0;
                for(var ix=j-r; ix<j+r+1; ix++) {
                    var x = Math.min(w-1, Math.max(0, ix));
                    val += scl[i*w+x];
                }
                tcl[i*w+j] = val/(r+r+1);
            }
    }
    function boxBlurT_3 (scl, tcl, w, h, r) {
        for(var i=0; i<h; i++)
            for(var j=0; j<w; j++) {
                var val = 0;
                for(var iy=i-r; iy<i+r+1; iy++) {
                    var y = Math.min(h-1, Math.max(0, iy));
                    val += scl[y*w+j];
                }
                tcl[i*w+j] = val/(r+r+1);
            }
    }
}

```

Algorithm 4

One-dimensional blur can be computed even faster. E. g. we want to compute horizontal blur. We compute $b_h[i,j], b_h[i,j+1], b_h[i,j+2], \dots, b_h[i,j], b_h[i,j+1], b_h[i,j+2], \dots$. But the neighboring values $b_h[i,j]b_h[i,j]$ and $b_h[i,j+1]b_h[i,j+1]$ are almost the same. The only difference is in one left-most value and one right-most value.

$$\text{Sob}_{b_h}[i,j+1] = b_h[i,j] + f[i,j+r+1] - f[i,j-r]b_h[i,j+1] = b_h[i,j] + f[i,j+r+1] - f[i,j-r].$$

In our new algorithm, we will compute the one-dimensional blur by creating the **accumulator**. First, we put the value of left-most cell into it. Then we will compute next values just by editing the previous value in constant time. This 1D blur has the complexity $O(n)$ (independent on r). But it is performed twice to get box blur, which is performed 3 times to get gaussian blur. So the complexity of this gaussian blur is $6 * O(n)$.

```

function gaussBlur_4 (scl, tcl, w, h, r) {
    var bxs = boxesForGauss(r, 3);
    boxBlur_4 (scl, tcl, w, h, (bxs[0]-1)/2);

```

```

    boxBlur_4 (tcl, scl, w, h, (bxs[1]-1)/2);
    boxBlur_4 (scl, tcl, w, h, (bxs[2]-1)/2);
}
function boxBlur_4 (scl, tcl, w, h, r) {
    for(var i=0; i<scl.length; i++) tcl[i] = scl[i];
    boxBlurH_4(tcl, scl, w, h, r);
    boxBlurT_4(scl, tcl, w, h, r);
}
function boxBlurH_4 (scl, tcl, w, h, r) {
    var iarr = 1 / (r+r+1);
    for(var i=0; i<h; i++) {
        var ti = i*w, li = ti, ri = ti+r;
        var fv = scl[ti], lv = scl[ti+w-1], val = (r+1)*fv;
        for(var j=0; j<r; j++) val += scl[ti+j];
        for(var j=0 ; j<=r ; j++) { val += scl[ri++] - fv
;    tcl[ti++] = Math.round(val*iarr); }
        for(var j=r+1; j<w-r; j++) { val += scl[ri++] -
scl[li++];    tcl[ti++] = Math.round(val*iarr); }
        for(var j=w-r; j<w ; j++) { val += lv -
scl[li++];    tcl[ti++] = Math.round(val*iarr); }
    }
}
function boxBlurT_4 (scl, tcl, w, h, r) {
    var iarr = 1 / (r+r+1);
    for(var i=0; i<w; i++) {
        var ti = i, li = ti, ri = ti+r*w;
        var fv = scl[ti], lv = scl[ti+w*(h-1)], val = (r+1)*fv;
        for(var j=0; j<r; j++) val += scl[ti+j*w];
        for(var j=0 ; j<=r ; j++) { val += scl[ri] - fv
;    tcl[ti] = Math.round(val*iarr);    ri+=w; ti+=w; }
        for(var j=r+1; j<h-r; j++) { val += scl[ri] -
scl[li];    tcl[ti] = Math.round(val*iarr);    li+=w; ri+=w; ti+=w; }
        for(var j=h-r; j<h ; j++) { val += lv -
scl[li];    tcl[ti] = Math.round(val*iarr);    li+=w; ti+=w; }
    }
}
}

```