ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Ανάπτυξη εφαρμογής επαυξημένης πραγματικότητας με χρήση τρισδιάστατου βίντεο

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Δημήτριος Α. Χριστόπουλος

Επιβλέπων : Στέφανος Δ. Κόλλιας

Καθηγητής Ε.Μ.Π.

Αθήνα, Σεπτέμβριος 2011

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Ανάπτυξη εφαρμογής επαυξημένης πραγματικότητας με χρήση τρισδιάστατου βίντεο

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Δημήτριος Α. Χριστόπουλος

Επιβλέπων : Στέφανος Δ. Κόλλιας,
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 5η Σεπτέμβρη 2011.

.....	.....	.....
Στέφανος Κόλλιας, καθηγητής Ε.Μ.Π.	Ανδρέας Σταφυλοπάτης, καθηγητής Ε.Μ.Π.	Γιώργος Στάμου, λέκτορας Ε.Μ.Π.

Αθήνα, Σεπτέμβριος 2011

.....

Δημήτριος Α. Χριστόπουλος

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Copyright © Δημήτριος Χριστόπουλος, 2011
Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

Περίληψη

Ο σκοπός της παρούσας διπλωματικής εργασίας είναι η δημιουργία μιας εφαρμογής επαυξημένης πραγματικότητας, βασισμένη στο ARToolKit, στην οποία ενα εικονικό αντικείμενο υπερτίθεται πάνω στην προοπτική του πραγματικού κόσμου. Το ARToolKit χρησιμοποιεί αλγόριθμους υπολογιστικής όρασης, για να υπολογίσει τη θέση της κάμερας, μεσω του εντοπισμού ενος φυσικού τετράγωνου σχεδίου, του λεγόμενου δείκτη (marker) ,και βάσει της θέσης αυτής σχεδιάζει το εικονικό αντικείμενο.
Το εν λόγω αντικείμενο αποτελείται απο μια επίπεδη επιφάνεια στην οποία προβάλλεται μια κινούμενη σκηνή (είτε εικονική είτε του πραγματικού κόσμου). Η σκηνή παρουσιάζεται απο διαφορετική προοπτική, αναλογα με τον προσανατολισμό της επιφάνειας, ενισχύοντας με αυτόν τον τρόπο τη τρισδιάστατη αίσθηση προς τον χρήστη.

.Οι διαφορετικές προοπτικές της κινούμενης σκηνής ουσιαστικά αποτελούν μια συλλογή απο αρχεία βίντεο γυρισμένα απο διαφορετική γωνία. Η εφαρμογή φροντίζει ετσι ώστε η μετάβαση μεταξυ αυτών να είναι ομαλή, χρησιμοποιώντας καταλληλες συναρτήσεις που παρέχονται απο τα API που χρησιμοποιήθηκαν για την ανάπτυξη της (στην προκειμένη περίπτωση απο το OpenCV).

Επιπροσθέτως, εφαρμόστηκε τρισδιάστατος ήχος με τη βοήθεια του OpenAL API. Τα χαρακτηριστικά του αναπαραγώμενου ήχου, όπως η ένταση μεταβάλλονται ανάλογα με τη θέση του δείκτη, δημιουργώντας με αυτόν τον τρόπο την εντύπωση οτι αυτός προέρχεται απο το εικονικό αντικείμενο και οτι κινείται μέσα στον τρισδιάστατο χώρο με ανάλογο τρόπο.

Απο άποψης λειτουργικότητας, δίνεται η επιλογή στον χρήστη να εισάγει τα δικά του αρχεία βίντεο και ήχου στο πρόγραμμα χωρίς να χρειάζεται απο πλευράς του να παρέμβει στο πρόγραμμα (αρκεί τα αρχεία αυτά να πληρούν τις προδιαγραφές που αναφέρονται παρακάτω στο κείμενο).

Η εφαρμογή αυτή μπορεί να επωφελήσει πολλούς τομείς, μερικοί απο τους οποίους είναι ο τομέας της διαφήμισης, με τη χρήση κινούμενων διαφημιστικών σπότ που μεταβάλλονται ανάλογα με την προοπτική του καταναλωτή, είτε ο τομέας των τηλε-συνδιασκέψεων, οπου ο χρήστης μπορεί να συνδιασκέπτεται με εικονικές εκδοχές των συνομιλητών του. Λαμβάνοντας υπόψιν οτι ο εξοπλισμός που απαιτείται πλέον είναι εύκολα προσβάσιμος απο το μέσο χρήστη γίνεται κατανοητό οτι η εφαρμογή μπορεί να χρησιμοποιηθεί ευρέως και με μικρό κόστος.

Λέξεις Κλειδία

Επαυξημένη πραγματικότητα, ARToolKit, OpenGL, OpenCV, OpenAL, υπολογιστική όραση, τρισδιάστατος ήχος

Abstract

The purpose of this thesis is the development of an augmented reality application, based on the ARToolKit, in which a virtual object is superimposed over the view of the real world. ARToolKit uses computer vision algorithms in order to calculate the position of the camera, by tracking a physical square shape called marker, and according to that position it draws the virtual object.

This object consists of a flat surface on which a moving scene is being projected (the scene can be either virtual or real-world). Depending on the orientation of the object, the scene is being presented from a different angle, thus creating an immersive user experience.

The different perspectives of the moving scene are actually a collection of video files captured from a different angle. The application makes sure the transition between them is smooth, with the help of the appropriate functions, supplied by the used APIs (in this case, OpenCV).

Furthermore, three-dimensional sound was implemented with the use of the OpenAL API. The characteristics of the audio file in reproduction, such as the volume vary depending on the position of the marker, thus creating the illusion that the virtual object is the sound source.

From functional perspective, the user can import his own video and audio files in the application without the need to modify the source code (as long as these files fill the requirements that are mentioned below).

The program can be used in many areas, some of which are the advertising, with the use of moving billboards, that vary depending on the angle observed from the consumers, or in teleconference where the user can confer with virtual editions of his interlocutors. Considering the fact the equipment required for the application is easily accessible to the average user, it becomes clear it can be used widely and at a small cost.

Keywords

Augmented Reality, ARToolKit, OpenGL, OpenCV, OpenAL, computer vision, 3D sound

Περιεχόμενα

1. Εισαγωγή στην «Επαυξημένη πραγματικότητα» (augmented reality)

1.1 Γενική περιγραφή

Ο όρος **Επαυξημένη πραγματικότητα** (Augmented reality ή AR) χρησιμοποιείται για να περιγράψει μια άμεση ή έμμεση άποψη του πραγματικού κόσμου του οποίου τα στοιχεία *επαυξάνονται* μέσω αισθητήριων εισόδων (sensory input) που δημιουργούνται απο υπολογιστή, όπως ήχος ή γραφικά. Σχετίζεται με ένα πιο γενικό κόνσεπτ που ονομάζεται *διαμεσολαβημένη πραγματικότητα* (mediated reality), κατα την οποία μια οπτική της πραγματικότητας τροποποιείται (πιθάνως και να ελαττώνεται αντί να επαυξάνεται) απο υπολογιστή. Εν αντιθέσει, η εικονική πραγματικότητα (virtual reality) αντικαθιστά τον πραγματικό κόσμο με έναν προσομοιωμένο.

Η επαύξηση γίνεται τυπικά σε πραγματικό χρόνο και συνδέεται απο άποψης περιεχομένου με στοιχεία του περιβάλλοντος όπως σκόρ σε έναν αγώνα ποδοσφαίρου στην τηλεόραση κατα τη διάρκεια του. Με τη βοήθεια προηγμένης τεχνολογίας επαυξημένης πραγματικότητας (π.χ. προσθέτοντας υπολογιστική όραση και αναγνώριση αντικειμένων) οι πληροφορίες σχετικά με τον περιβάλλοντα κόσμο του χρήστη γίνονται διαδραστικές (interactive) και δίνεται η δυνατότητα να χειραγωγηθούν ψηφιακά. Οι τεχνητές πληροφορίες σχετικά με το περιβάλλον και τα αντικείμενά του μπορεί να σχεδιαστεί «πανω» απο τον πραγματικό κόσμο. Ο όρος «Επαυξημένη πραγματικότητα» πιστεύεται οτι επινοήθηκε το 1990 απο τον Thomas Caudell, ο οποίος εργαζόταν στην εταιρία “Boeing”.

Οι σχετικές έρευνες που γίνονται εξερευνούν την εφαρμογή εικόνων δημιουργημένων απο υπολογιστή (computer-generated imagery) σε «ζωντανά» βίντεο σαν έναν τρόπο για να «ενισχύσουν» την προοπτική του πραγματικού κόσμου.

Η τεχνολογία επαυξημένης πραγματικότητας περιλαμβάνει τις λεγόμενες συσκευές “head-mounted displays” (βλ. Ενότητα 2.10.2) και τις “virtual retinal displays” για σκοπούς οπτικοποίησης (visualization), και την κατασκευή ελεγχόμενων περιβαλλόντων που περιέχουν αισθητήρες και ενεργοποιητές (actuators).

Εικόνα 1.1 : browser στο "iPhone 3GS" χρησιμοποιεί το GPS και την πυξίδα

Εικόνα 1.2 : παιχνίδι στο "Nokia N95" που χρησιμοποιεί δείκτες (markers)

Εικόνα 1.3 : πρόγραμμα που «επαυξάνει» την προοπτική του πραγματικού κόσμου, προβάλλοντας σχετικές πληροφορίες

Εικόνα 1.4 : πρόγραμμα εικονικού πληκτρολογίου οπου ο χρήστης το ελέγχει μέσω της κίνησης του δείκτη

1.2 Παραδείγματα

1.2.1) Σπορ

Η επαυξημένη πραγματικότητα έχει διαδοθεί αρκετά στην μετάδοση αγώνων σπορ. Για παράδειγμα, σε αγώνες ποδοσφαίρου, κατα τη διάρκεια του replay, φαίνεται η κίτρινη γραμμή που υποδηλώνει πότε ένας παίκτης είναι off-side. Τα στοιχεία του πραγματικού κόσμου είναι το γήπεδο και οι παίκτες, ενώ το εικονικό στοιχείο η κίτρινη γραμμή, που επαυξάνει την εικόνα. Αντίστοιχα, σε αγώνες ice-hockey, μία κόκκινη «ουρά» υποδηλώνει την τροχιά του δίσκου. Είναι διαδεδομένη επίσης η προβολή διαφημίσεων σε τμήματα του αγωνιστικού χώρου (συνήθως στο μπάσκετ).

Παρομοίως, στις μεταδόσεις αγώνων κολύμβησης συχνά προστίθεται μία γραμμή κάθετα στις λωρίδες των αγωνιζομένων που υποδεικνύει τη θέση του τρέχοντος αθλήτη που είναι πρώτος.

1.2.2) Άλλες εφαρμογές

Τα παιχνίδια βολών προοπτικής πρώτου προσώπου (first-person shooters) προσομοιώνουν μια προοπτική του παίκτη, χρησιμοποιώντας επαυξημένη πραγματικότητα για να τον κατευθύνει προς μία περιοχή, να προβάλλει την απόσταση απο έναν άλλον παίκτη ο οποίος δε ειναι ορατός και να δώσει πληροφορίες σχετικά με τον εξοπλισμό και τα εναπομείναντα πυρομαχικά. Αυτό επιτυγχάνεται με τη χρήση ενός εικονικού "head-up display".

Τα "head-up displays" εφαρμόζονται επίσης σε ορισμένα μοντέλα αυτοκινήτων ή σε αεροπλάνα και συνήθως είναι ενσωματωμένα στο παρμπρίζ.

Σε κάποια πολεμικά αεροπλάνα η πληροφορία απεικονίζεται στο κράνος του πιλότου, δίνοντας του τη δυνατότητα να βλέπει «μέσα» απο το αεροπλάνο (σαν αυτό να ήταν διαφανές) και να έχει έτσι καλύτερη εποπτεία του περιβάλλοντος χώρου.

1.3 Εφαρμογές μέχρι το 2011

***Διαφήμιση*:** Η χρήση επαυξημένης πραγματικότητας για την προώθηση προϊόντων δια μέσου διαδραστικών εφαρμογών γίνεται ολοένα και περισσότερο πιο κοινή τακτική.

Υποστήριξη εργασιών (task support): Πολύπλοκες εργασίες όπως η συναρμολόγηση, η συντήρηση, και η χειρουργική μπορούν να απλοποιηθούν με την εισαγωγή επιπρόσθετων πληροφοριών στο οπτικό πεδίο. Για παράδειγμα, επιγραφές μπορούν να απεικονιστούν σε τμήματα ενός συστήματος και να κάνουν πιο σαφείς τις οδηγίες χρήσης του για έναν μηχανικό που εκτελέι συντήρηση στο σύστημα αυτό. Η επαυξημένη πραγματικότητα μπορεί να περιλαμβάνει εικόνες απο κρυφά αντικείμενα, κατι που μπορεί να είναι πολύ αποτελεσματικό στη διαγνωστική ιατρική ή στη χειρουργική. Τα παραδείγματα περιλαμβάνουν μια εικονική προοπτική ακτίνων-Χ που βασίζεται σε προηγούμενες τομογραφείες ή στις εικόνες πραγματικού χρόνου απο υπέρηχο και μικροομοεστιακούς ανιχνευτές (micro-confocal probes). Η επαυξημένη πραγματικότητα μπορεί να επαυξήσει την οπτική ενός εμβρύου μέσα στην μήτρα της μητέρας του.

Πλοήγηση: Η επαυξημένη πραγματικότητα μπορεί να αυξήσει την αποτελεσματικότητα των συσκευών πλοήγησης. Για παράδειγμα, η πλοήγηση κτιρίων μπορεί να βοηθήσει στην

συντήρηση βιομηχανικών εγκαταστάσεων. Η πλοήγηση εξωτερικού χώρου μπορει να επαυξηθεί για στρατιωτικούς σκοπους ή για χειρισμό καταστροφών. Τέλος, η πλοήγηση μπορεί να χρησιμοποιηθεί σε διάφορα οχήματα όπως αυτοκίνητα, αεροπλάνα κλπ.

Ψυχαγωγία και μόρφωση: Με τη βοήθεια της επαυξημένης πραγματικότητας μπορούμε να δημιουργήσουμε εικονικά αντικείμενα σε μουσεία και εκθέσεις, θεματικά πάρκα, παιχνίδια και βιβλία. Επίσης, η επαυξημένη πραγματικότητα μπορει να «διανθίσει» συναυλίες και θεατρικές παραστάσεις.

1.4 Δυνατές εφαρμογές

Πιθανές εφαρμογές περιλαμβάνουν:

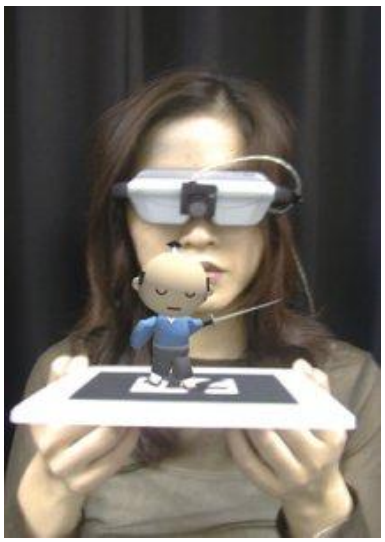
- *Συσκευές:* Δημιουργία νέων εφαρμογών που είναι φυσικώς αδύνατες σε «πραγματικό» hardware, όπως 3Δ αντικείμενα που αλλάζουν διαδραστικώς το σχήμα τους και την εμφάνισή τους ανάλογα με την τρέχουσα εργασία ή ανάγκη.
- *Προσομοίωση πολλαπλών οθονών:* Απεικόνιση πολλαπλών «παραθύρων» εφαρμογών σαν εικονικά μόνιτορ στον πραγματικό χώρο και αλλαγή μεταξύ τους με χειρονομίες ή με την ανακατεύθυνση του κεφαλιού και των ματιών
- *Αυτοκίνηση:* eye-dialing, βέλη πλοήγησης στο δρόμο
- *Φωνητική σύνθεση:* Προβολή πληροφοριών σχετικών με την τοποθεσιά/περιεχόμενο μέσα απο προφορικό λόγο.
- *Αναζήτηση:* στην υδρολογία, οικολογία και γεωλογία. Η επαυξημένη πραγματικότητα μπορεί να απεικονίσει μία διαδραστική ανάλυση των γεωγραφικών χαρακτηριστικών. Οι χρήστες μπορούν συνεργατικά να τροποποιούν και να αναλύουν διαδραστικού τρισδιάστατους χάρτες.

2. Το ARToolKit

2.1 Γενικά

Το ARToolKit είναι μια βιβλιοθήκη λογισμικού για την δημιουργία εφαρμογών επαυξημένης πραγματικότητας (Augmented Reality). Πρόκειται για εφαρμογές που αφορούν στην προβολή εικονικών αντικειμένων (virtual imagery) στον πραγματικό κόσμο. Για παράδειγμα, στην παρακάτω εικόνα ενας τρισδιάστατος εικονικός χαρακτήρας φαίνεται να στέκεται σε μια πραγματική κάρτα. Ο χρήστης μπορεί να το δει απο τα ειδικά γυαλιά. Όταν ο χρήστης μετακινεί το φύλλο, ο εικονικός χαρακτήρας κινείται με αυτό και φαίνεται συνδέδεμένος με το πραγματικό αντικείμενο.

Μία από τις βασικές δυσκολίες στην ανάπτυξη εφαρμογών επαυξημένης πραγματικότητας είναι το πρόβλημα του εντοπισμού της προοπτικής των χρηστών (δηλαδή, το πού κοιτά ο χρήστης). Για να γνωρίζουμε από ποιά άποψη πρέπει να σχεδιαστεί το εικονικό αυτό αντικείμενο, η εφαρμογή πρέπει να ξέρει προς τα πού κοιτά ο χρήστης στον πραγματικό κόσμο.



Εικόνα 2.1: Η εμπειρία της επαυξημένης πραγματικότητας

Το ARToolKit χρησιμοποιεί αλγορίθμους υπολογιστικής όρασης (computer vision algorithms) για να λύσει αυτό το πρόβλημα. Οι βιβλιοθήκες βιντεο-εντοπισμού του ARToolKit υπολογίζουν τη θέση και τον προσανατολισμό της πραγματικής κάμερας σε σχέση με τους πραγματικούς δείκτες (markers) σε πραγματικό χρόνο. Αυτό επιτρέπει την εύκολη ανάπτυξη ενός ευρέος φάσματος εφαρμογών επαυξημένης πραγματικότητας.

Κάποια απο τα χαρακτηριστικά του ARToolKit είναι τα παρακάτω:

- Εντοπισμός θέσης/προσανατολισμού μονής κάμερας.
- Κώδικας εντοπισμού που χρησιμοποιεί απλά μαύρα τετράγωνα.
- Δυνατότητα χρήσης οποιουδήποτε pattern για τον τετράγωνο δείκτη.
- Απλός κώδικας για τη βαθμονόμηση της κάμερας.
- Αρκετά γρήγορο για δημιουργία εφαρμογών επαυξημένης πραγματικότητας πραγματικού χρόνου.
- Διανομές για SGI IRIX, Linux, MacOS και Windows OS.
- Διανέμενεται με πλήρη πηγαίο κώδικα.

2.2 Λίστα Χαρακτηριστικών

- Απλό framework για τη δημιουργία εφαρμογών επαυξημένης πραγματικότητας πραγματικού χρόνου.
- Βιβλιοθήκη που υποστηρίζει πολλές πλατφόρμες (Windows, Linux, Mac OS X, SGI)
- Υποστηρίζει τα περισσότερα φορμάτ συσκευών εισόδου (USB, Firewire, ειδικές κάρτες σύλληψης)
- Πολλαπλά φορμάτ βίντεο (RGB/YUV420P, YUV)
- Εντοπισμό πολλαπλών καμερών
- Διαπροσωπία αρχικοποίησης γραφικού περιβάλλοντος
- Γρήγορος και φτηνός εντοπισμός δείκτη 6 διαστάσεων (real-time planar detection)
- Εύκολη ρουτίνα βαθμονόμησης
- Απλή βιβλιοθήκη γραφικών (βασισμένη στο GLUT)
- Γρήγορη σχεδίαση γραφικών βασισμένη στο OpenGL
- Υποστήριξη 3D VRML
- Απλό και αρθρωτό API[(1)] (σε γλώσσα C)
- Υποστηρίξη και άλλων γλωσσών (JAVA, Matlab)
- Ολοκληρωμένα σετ παραδειγμάτων και βοηθημάτων
- Μια καλή λύση για μια πιο «απτή» αλληλεπίδραση
- Ανοιχτού κώδικα
- Πολλές δυνατές εφαρμογές σε εμπορικό και ακαδημαϊκό επίπεδο
- Πολύ προσιτό στον μέσο προγραμματιστή καθώς υπάρχουν λεπτομερείς οδηγίες για την εγκατάσταση του , καθως και σωρεία έτοιμων εφαρμογων που μπορούν να τον βοηθήσουν στην εύκολη ανάπτυξη των δικών του

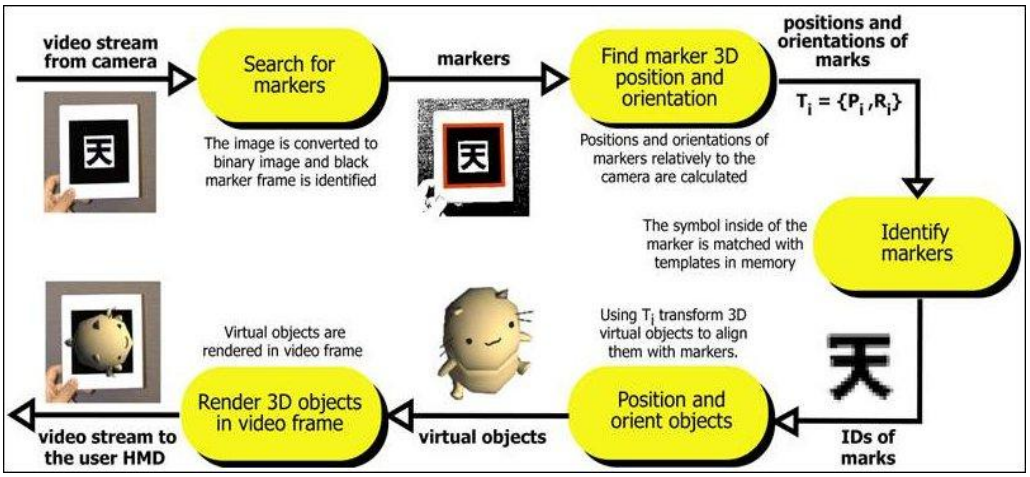
2.3 Βασικές Αρχές

Οι εφαρμογές που είναι βασισμένες στο ARToolKit επιτρέπουν την υπέρθεση εικονικών αντικειμένων πάνω απο ζωντανό βίντεο του πραγματικού κόσμου. Παρότι αυτό μπορεί να φαντάζει «μαγικό» δεν είναι. Το μυστικό κρύβεται στα μαύρα τετράγωνα τα οποία χρησιμοποιούνται σαν δείκτες εντοπισμού. Ο εντοπισμός στο ARToolKit λειτουργέι ως εξής:

1. Η κάμερα συλλαμβάνει βίντεο απο τον πραγματικό κόσμο και το στέλνει στον υπολογιστή.
2. Λογισμικό στον υπολογιστή αναζητά σε κάθε καρέ του εν λόγω βίντεο για τετράγωνα σχήματα.
3. Αν βρεθεί κάποιο τετράγωνο, το λογισμικό χρησιμοποιεί μαθηματικά για να υπολογίσει τη θέση της κάμερας σε σχέση με το μαύρο τετράγωνο.
4. Οταν η θέση της κάμερας γίνει γνωστή ένα μοντέλο γραφικών σχεδιάζεται απο την ίδια θέση
5. Το μοντέλο σχεδιάζεται πάνω απο το βίντεο και με αυτόν τον τρόπο φαίνεται κολλημένο στον τετράγωνο δείκτη.

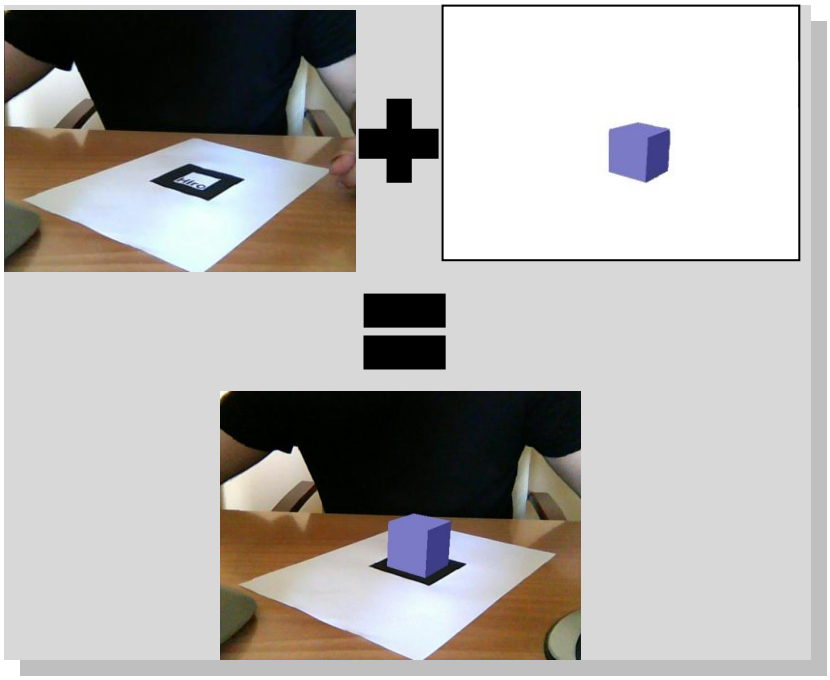
6. Η τελική έξοδος εμφανίζεται στην οθόνη, οπότε όταν ο χρήστης την κοιτά βλέπει τα γραφικά σαν να αποτελούν μέρος του πραγματικού κόσμου.

Η παρακάτω εικόνα συνοψίζει αυτά τα βήματα. Το ARToolKit έχει τη δυνατότητα να εκτελεί τον εντοπισμό της θέσης της κάμερας σε πραγματικό χρόνο, εξασφαλίζοντας έτσι οτι τα εικονικά αντικείμενα εμφανίζονται πάντα πάνω απο τους δείκτες εντοπισμού.



Εικόνα 2.2: Βηματα εντοπισμού του δείκτη

Παρακάτω βλέπουμε μια εικόνα που δείχνει οτι το τελικό αποτέλεσμα αποτελεί υπέρθεση της εικόνας του πραγματικού κόσμου και της εικονικής σκηνής (χρησιμοποιήθηκε σαν παραδειγμα η εφαρμογή simple.exe, ο κώδικας της οποίας μπορεί να βρεθει στο παράρτημα 2).



Εικόνα 2.3: Η τελική εικόνα προκύπτει ως αθροισμα της εικόνας του πραγματικού κόσμου και των γραφικών που εναποτίθενται πάνω σε αυτήν

2.4 Περιορισμοί

Υπάρχουν κάποιοι περιορισμοί στα συστήματα επαυξημένης πραγματικότητας που βασίζονται στο ARToolKit. Είναι φυσικό επακόλουθο οτι τα εικονικά αντικείμενα εμφανίζονται μόνο όταν οι δείκτες εντοπισμού είναι ορατοί. Αυτό μπορεί να περιορίσει το μέγεθος ή την κίνηση των εικονικών αυτών αντικειμένων. Σημαίνει επίσης οτι αν ο χρήστης καλύψει μέρος του δείκτη με τα χέρια του ή με άλλο αντικείμενο τότε το εικονικό αντικείμενο θα εξαφανιστεί.

Υπάρχουν επίσης θέματα εμβέλειας (range issues). Όσο μεγαλύτερος είναι ο δείκτης σε μέγεθος τόσο αυξάνεται η εμβέλεια στην οποία μπορεί να ανιχνευτεί και άρα και ο χώρος μεσα στον οποίον ο χρήστης μπορεί να ανιχνευτεί. Ο Πίνακας 1 συνοψίζει κάποιες τυπικές μέγιστες εμβέλειες για τετράγωνους δείκτες διάφορων μεγεθών. Τα αποτελέσματα αυτά συλλέχθηκαν, χρησιμοποιώντας δείκτες διαφορετικών μεγεθών (διαφορετικά μήκη πλευρών), και τοποθετώντας τα κάθετα στην κάμερα και μετακινώντας την κάμερα προς τα πίσω μέχρι τα εικονικά αντικείμενα πάνω στα τετράγωνα να εξαφανιστούν.

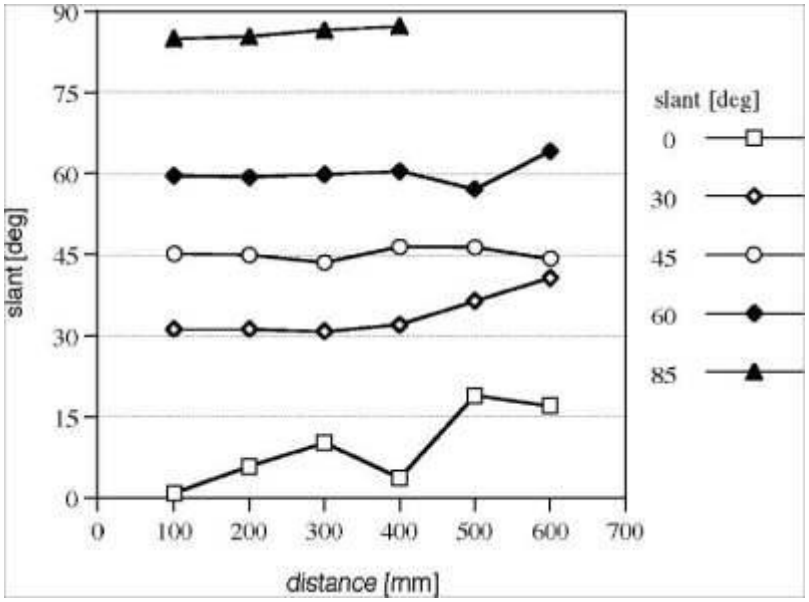
Μέγεθος Δείκτη (ίντσες)	**Ωφέλιμη εμβέλεια** (ίντσες)
2.75	16
3.50	25
4.25	34
7.37	50

Πίνακας 2.1: Εμβέλεια εντοπισμού για δείκτες διαφορετικού μεγέθους.

Η εμβέλεια επηρεάζεται επίσης σε μικρό βαθμό και απο πολυπλοκότητα του εσωτερικού σχεδίου του δείκτη (pattern complexity). Οσο πιο απλό είναι το σχέδιο τόσο το καλύτερο. Σχέδια με μεγάλες μαύρες και άσπρες περιοχές (π.χ. σχέδια χαμηλής συχνότητας) είναι τα πιο αποτελεσματικά. Αντικατάσταση του τετράγωνου σχεδίου που χρησιμοποιήθηκε παραπάνω, με ένα σχέδιο ίδιου μεγέθους αλλα πιο πολύπλοκο, μειώνει την εμβέλεια εντοπισμού απο τις 34 στις 15 ίντσες.

Εικόνα 2.4: Διάφορα δημοφιλή σχέδια για τον τετράγωνο δείκτη

Ο εντοπισμός επηρεάζεται επίσης και απο τον προσανατολισμό του δείκτη σε σχέση με την κάμερα. Καθώς οι δείκτες γίνονται πιό οριζόντιοι και αυξάνεται η κλίση τους, το σχέδιο στο κέντρο γίνεται όλο και λιγότερο ορατό, με αποτέλσμα η αναγνώριση να γίνεται λιγοτερο αξιόπιστη.



Εικόνα 2.5: Σφάλμα εντοπισμού συναρτήσει της γωνίας προσανατολισμού

Τέλος, ο εντοπισμός επηρεάζεται και απο τις συνθήκες φωτισμού. Συνήθεις πήγες φωτισμού όπως τα φωτιστικά μπορούν να δημιουργήσουν αντανακλάσεις και φωτεινά σημεία σε έναν δείκτη που είναι τυπωμένος σε χαρτί και ως εκ τούτου η αναγνώριση του σχεδίου να γίνεται πιο δυσχερής. Για να περιοριστούν τέτοια φαινόμενα, συστήνεται η χρήση υλικών που δεν είναι επιρρεπή στις αντανακλάσεις για την κατασκευή του δείκτη. Για παράδειγμα μπορει να χρησιμοποιηθεί βελούδινο ύφασμα, που κολλά σε μία άσπρη βάση, και το οποίο είναι ευρέως διαδεδομένο και μπορεί να βρεθεί σε πολλά καταστήματα.

2.5 Αρχές ανάπτυξης εφαρμογών βασισμένες στο ARToolKit

Η ανάπτυξη εφαρμογών πο βασίζονται στο ARToolKit μπορεί να χωριστεί σε δύο μέρη: στη συγγραφή της εφαρμογής, και στην «εκπαίδευση» ρουτινών επεξεργασίας εικόνας πάνω σε δείκτες του πραγματικού κόσμου οι οποίες θα χρησιμοποιηθούν στην εφαρμογή.

Η δημιουργία μιας εφαρμογής με το ARToolKit είναι μια πολύ απλή υπόθεση· ένα απλό σχεδιάγραμμα χρησιμοποιείται για την ανάπτυξη της εφαρμογής επαυξημένης πραγματικότητας. Ο προγραμματιστής βασίζεται σε αυτό για να συγγράψει μια νέα εφαρμογή. Παρομοίως, η φάση της εκπαίδευσης που αναφέρθηκε προηγουμένως είναι αρκετά απλοποιημένη με τη βοήθεια ενός πολύ απλού εργαλείου.

Ο προγραμματιστής πρέπει να ακολουθήσει τα παρακάτω βήματ.α για την ανάπτυξη της εφαρμογής:

Αρχικοποίηση	1. Αρχικοποίηση της σύλληψης βίντεο και ανάγνωση των αρχείων σχεδίων δείκτη και των παραμέτρων της κάμερας
Main  **Loop**	2. Σύλληψη ενός καρέ απο το βίντεο.
	3. Εντοπισμός του δείκτη και των αναγνωρίσιμων σχεδίων απο το το καρέ.
	4. Υπολογισμός του μετασχηματισμού της κάμερας σε σχέση με τους εντοπισμένους δείκτες.
	5. Σχεδιασμός των εικονικών αντικειμένων πάνω στους εντοπισμένους δείκτες
Τερματισμός	6.Τερματισμός της σύλληψης βίντεο (video capture)

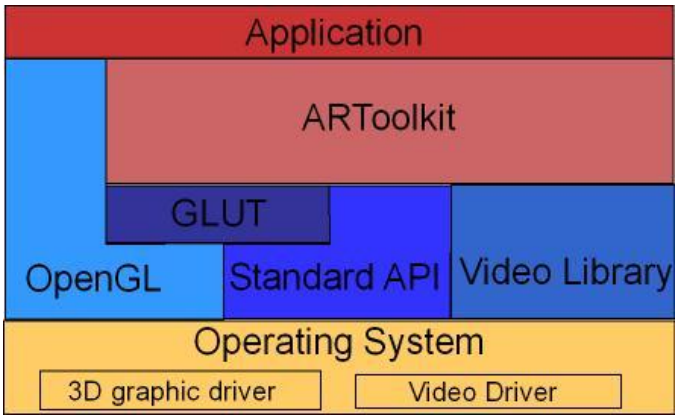
Πίνακας 2.2: Βήματα που ακολουθούν οι εφαρμογές ARToolKit

Τα βήματα 2 έως 5 επαναλαμβάνονται συνεχώς μέχρι η εφαρμογή να τερματιστεί, ενώ τα βήματα 1 και 6 απλά εκτελούνται στην αρχικοποίηση και στον τερματισμό της εφαρμογής αντίστοιχα. Επιπροσθέτως, η εφαρμογή ενδεχομένως να χρειαστεί να ανταποκρίνεται σε εισόδους που δίνει ο χρήστης μέσω του ποντικιού, του πληκτρολογίου ή σε γεγονότα (events) συγκεκριμένα στην εφαρμογή.

2.6 Το framework του ARToolKit (μια πιο αναλυτική ματιά)

Το ARToolKit αποτελεί μια εργαλειοθήκη λογισμικού (software toolkit). Περιλαμβάνει προκαθορισμένες συναρτήσεις που θα χρειαστεί να καλέσει ο προγραμματιστής με συγκεκριμένη σείρα για να αναπτύξει μια εφαρμογή επαυξημένης πραγματικότητας. Μπορεί όμως να χρησιμοποιήσει διαφορετικά μέρη του toolkit ξεχωριστά. Το ARToolKit υποστηρίζει πολλαπλές πλατφόρμες, ενω είναι ανεπτυγμένο έτσι ώστε να ελαχιστοποιούνται οι εξαρτήσεις απο βιβλιοθήκες χωρίς να γίνονται θυσίες στην αποδοτικότητα. Το ARToolKit χρησιμοποιεί το OpenGL για τον σχεδιασμό των εικονικών αντικειμένων, το GLUT για το κομμάτι του χειρισμού γεγονότων (event handling), καθώς και βιβλιοθήκες βίντεο που εξαρτώνται απο το υλικό (hardware). Τέλος, σε κάθε πλατφόρμα χρησιμοποιείται το προκαθορισμένο API[(1)] (π.χ. win32 στα Windows).

Η παρακάτω εικόνα συνοψίζει τη σχέση μεταξύ της εφαρμογής (Application), του ARToolKit και των εξαρτώμενων βιβλιοθηκών.



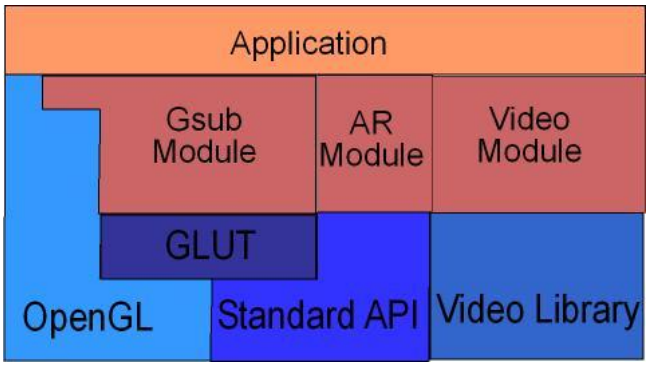
Εικόνα 2.6: Η αρχιτεκτονική του ARToolKit.

2.6.1) Δομή

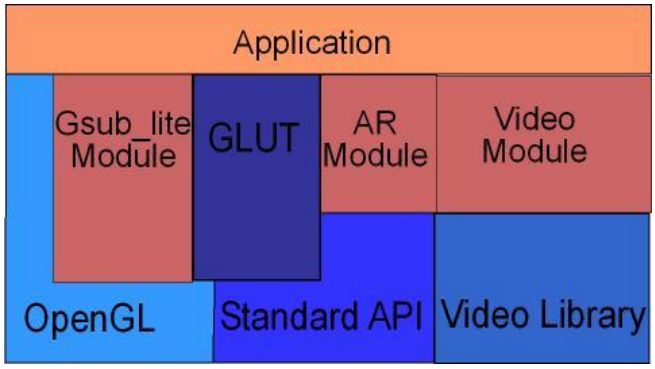
Η βιβλιοθήκη του ARToolKit αποτελείται απο 4 modules:

- AR module: το βασικό module με ρουτίνες εντοπισμού δείκτη, βαθμονόμηση και συλλογή παραμέτρων
- Video module: μια συλλογή απο ρουτίνες βίντεο για τη σύλληψη των καρέ του βίντεο εισόδου.
- Gsub module: μια συλλογή απο ρουτίνες γραφικών βασισμένες στις βιβλιοθήκες OpenGL και GLUT.
- Gsub Lite: αντικαθιστά το Gsub με μία πιο αποδοτική συλλογή ρουτινών για γραφικά

Οι επόμενες εικόνες δείχνουν την ιεραρχική δομή του ARToolKit και τη σχέση με τις εξαρτώμενες βιβλιοθήκες:

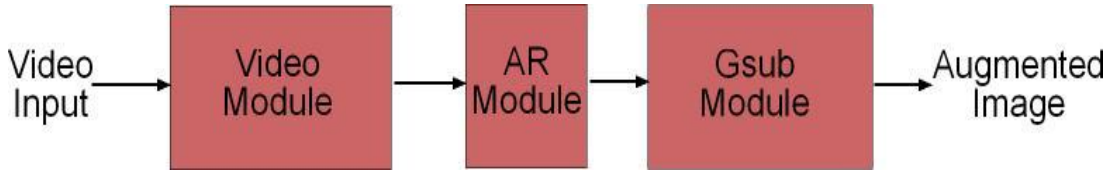


Εικόνα 2.6.α: Ιεραρχική δομή του ARToolKit με τη χρήση του Gsub module



Εικόνα 2.6.β: Ιεραρχική δομή του ARToolKit με τη χρήση του Gsub_Lite module

Τα module ακολουθούν το λεγόμενο *pipeline metaphor*, του οποίου η σημασία γίνεται σαφής στο παρακάτω σχήμα, έτσι ώστε αν ο χρήστης επιθυμεί να αντικαταστήσει κάποιο απο αυτά, να το κάνει με ευκολία και χωρίς να δημιουργουνται προβλήματα (π.χ. η αντικατάσταση του gsub απο τον Open Inventor renderer).

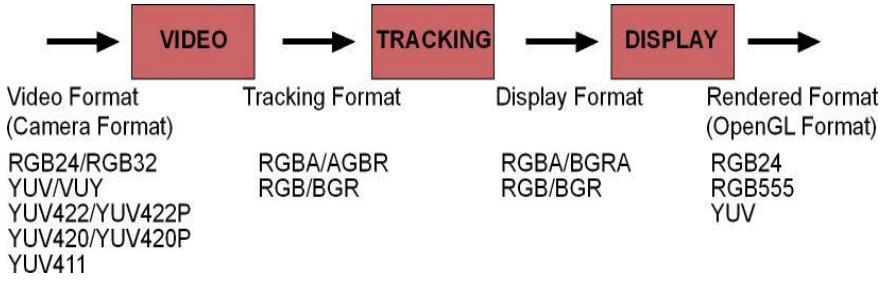


Εικόνα 2.7: Βασικό pipeline του ARToolKit

2.6.2) Τύποι Δεδομένων

Το ARToolKit χειρίζεται πολλά διαφορετικά είδη μεταβλητών. Εσωτερικά, χρησιμοποιεί καθολικές (global) μεταβλητές, οι οποίες περιορίζουν την εκ νέου είσοδο σε μέρος του κώδικα. Διαφορετικά, χρησιμοποιείται η καθιερωμένη διαπροσωπία πολλαπλών ορισμάτων (multi-argument interface) με βάση μια προσέγγιση ροής δεδομένων (data-flow approach).

Το ARToolKit χρησιμοποιεί διαφορετικούς τύπους αρχείου εικόνας μεταξύ διαφορετικών modules. Η εικόνα 6 συνοψίζει όλους τους διαφορετικούς τύπους αρχείων που υποστηρίζονται. Κάποιοι τύποι είναι διαθέσιμοι σε συγκεκριμένες πλατφόρμες ή σε συγκεκριμένο hardware.



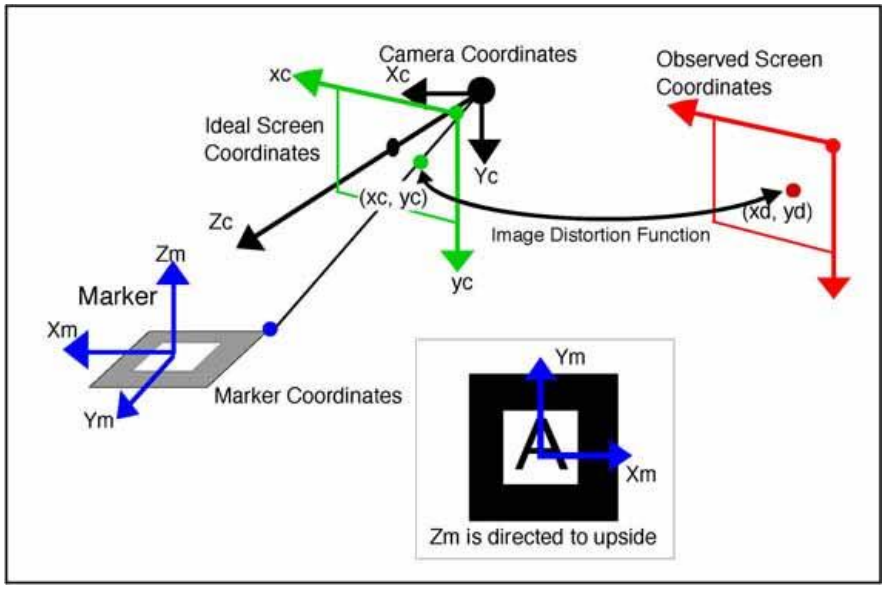
Εικόνα 2.8: Ροή δεδομένω του ARToolKit.

2.7 Σύστηματα συντενταγμένων στο ARToolKit

Το ARToolKit υπολογίζει τη θέση του δείκτη στο σύστημα συντεταγμένων της κάμερας, και χρησιμοποιεί το σύστημα πινάκων του OpenGL για τη θέση του εικονικού αντικειμένου. Ετσι, η θέση του δείκτη επιστρέφει σε έναν πίνακα 3X4, με τις σαφείς συντεταγμένες του (περισσότερα επι αυτού στα επόμενα κεφάλαια).

Το ARToolKit ορίζει διάφορα συστήματα συντεταγμένων, τα οποία χρησιμοποιούνται κυρίως απο τον αλγόριθμο υπολογιστικής όρασης (computer vision algorithm) και τη σχεδίαση της εικονικής σκηνής.

Το παρακάτω σχηματικό συνοψίζει τα βασικά συστήματα συντεταγμένων που χρησιμοποιούνται απο το ARToolKit:

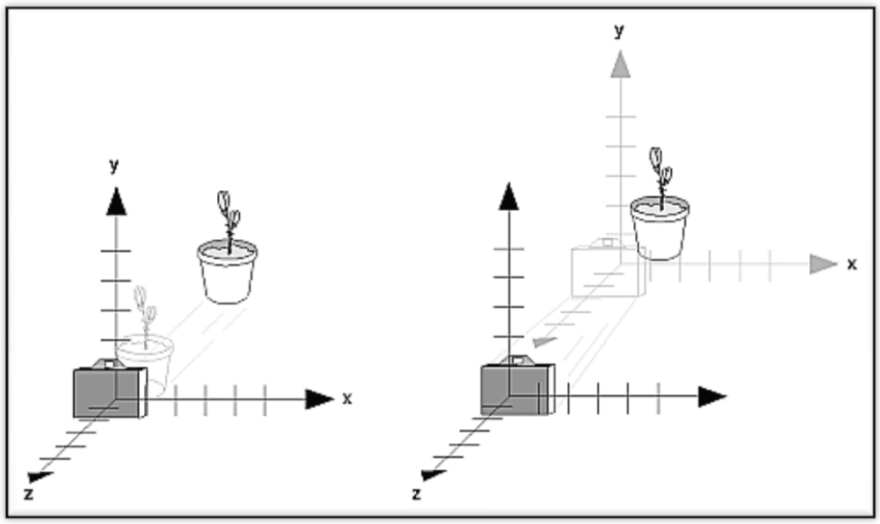


Εικόνα 2.9 : Συστήματα συντεταγμένων του ARToolKit

Πρέπει να καταστεί σαφές οτι το πρόγραμμα θα επιστρέψει **τη θέση του δείκτη στο σύστημα συντεταγμένων της κάμερας**(και όχι το αντίστροφο). Εάν ο χρήστης επιθυμεί να του επιστραφεί η θέση της κάμερας στο σύστημα συντεταγμένων του δείκτη θα πρέπει να αντιστρέψει τον πίνακα.

2.7.1) Σύστημα συντεταγμένων που γίνεται η σχεδίαση της σκηνής

Όταν χρησιμοποιείται το ARToolKit με το OpenGL, πρέπει να επισημανθεί οτι το σύστημα συντεταγμένων είναι δεξιόστροφο, με την κάμερα να κοιτά προς την κατεύθυνση -Z



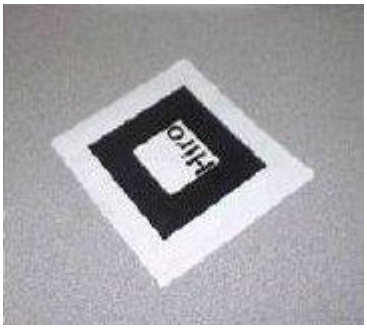
Εικόνα 2.10: Ο μετασχηματισμός του συστήματος συντεταγμένων

Το ARToolKit χρησιμοποιεί μια βαθμονομημένη προοπτική για την κάμερα που συνήθως δίνει ως αποτέλεσμα εναν πίνακα προβολής εκτός άξονα (off-axis projection matrix) για το OpenGL. Κάτι τέτοιο δε μπορεί να γίνει μόνο απο το OpenGL και τις συναρτήσεις που αυτό παρέχει.

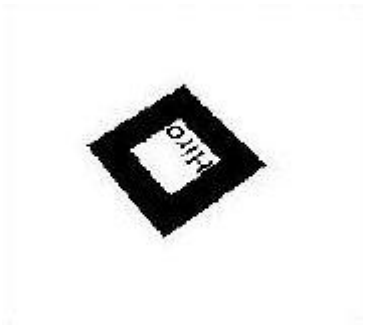
2.8 Αλγόριθμος υπολογιστικής όρασης (computer vision algorithm)

Ο αλγόριθμος υπολογιστικής όρασης χρησιμοποιείται για τον εντοπισμό του δείκτη απο την εικόνα που έχουμε ως είσοδο απο την κάμερα. Αυτό που κάνει ειναι να λαμβάνει τα καρέ του βιντεο, έπειτα να απομονώνει τα μαύρα τμήματα και με μαθηματικές μεθόδους να αναγνωρίζει αν τα σχήματα που εμφανίζονται είναι δείκτες ή όχι. Αν βρεθεί δείκτης, τοτε ακολουθεί ο μετασχηματισμός συντεταγμένων που περιγράφθηκε παραπάνω.

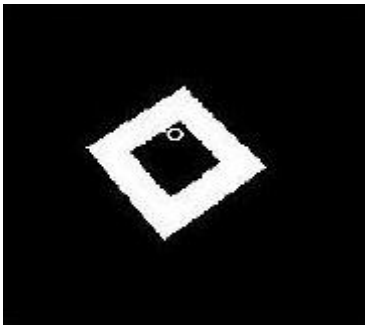
Οι επόμενες εικόνες περιγράφουν συνοπτικά τις αρχές του αλγορίθμου.



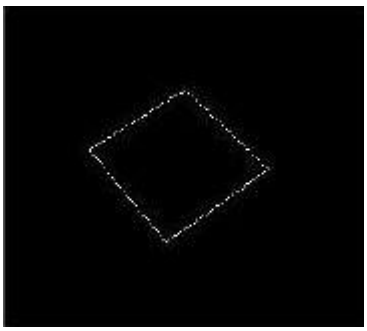
2.11.a.Αρχική εικόνα



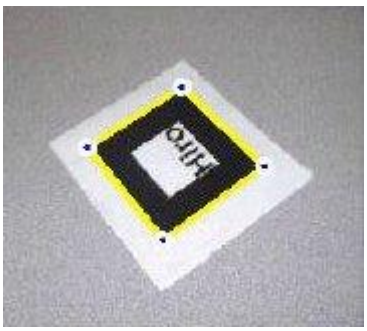
2.11.b. Απομόνωση μαύρων τμημάτων



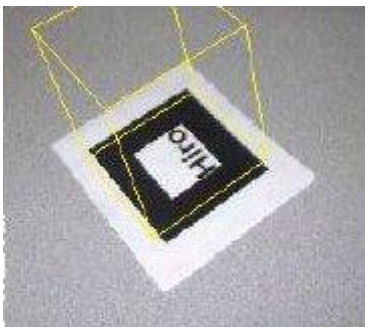
2.11.c. Συνδεδεμένα τμήματα



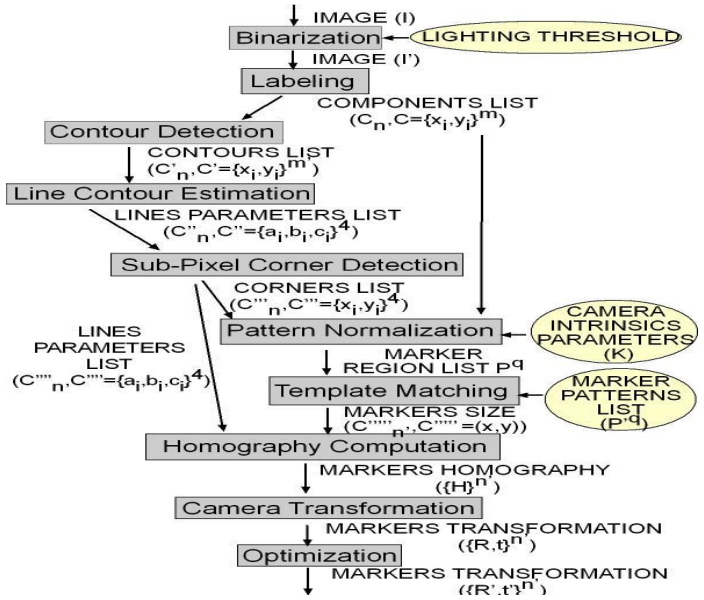
2.11.d. Περίγραμμα



2.11.e. Εντοπισμός των ακμών και των γωνιών του δείκτη



2.11.f.Εφαρμοσμένο τετράγωνο



Εικόνα 2.12 :Θεωρητικά βήματα για την εύρεση του δείκτη στην εικόνα

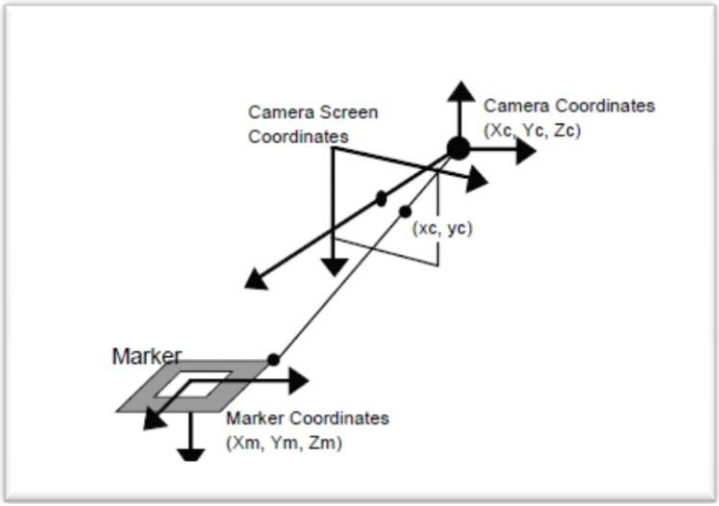
2.9 Εκτίμηση θέσης και προσανατολισμού των δεικτών

2.9.1) Εκτίμηση του πίνακα μετασχηματισμού

Οι τετράγωνοι δείκτες γνωστού μεγέθους χρησιμοποιούνται σαν βάση για το συστημα συντεταγμένων. Για να εξάγουμε τη θέση τους στο σύστημα συντεταγμένων, απαιτείται να γίνει μετασχηματισμός συντεταγμένων. Οι μήτρες μετασχηματισμού απο το σύστημα συντεταγμένων με βάση τον δείκτη σε αυτό με βάση την κάμερα παρουσιάζονται στις παρακάτω εξισώσεις:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} V_{11} & V_{12} & V_{13} & W_x \\ V_{21} & V_{22} & V_{23} & W_y \\ V_{31} & V_{32} & V_{33} & W_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_m \\ Y_m \\ Z_m \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} \mathbf{V}_{3\times3} & \mathbf{W}_{3\times1} \\ 0 \quad 0 \quad 0 & 1 \end{bmatrix} \begin{bmatrix} X_m \\ Y_m \\ Z_m \\ 1 \end{bmatrix} = \mathbf{T}_{cm} \begin{bmatrix} X_m \\ Y_m \\ Z_m \\ 1 \end{bmatrix} \quad \text{(εξ. 1)}$$



Εικόνα 2.13: Η σχέση μεταξύ των συντεταγμένων του δείκτη και των συντεταγμένων της κάμερας προκύπτει απο την ανάλυση της εικόνας

Όπως αναφέρθηκε και σε προηγούμενες ενότητες, μεσω της επεξεργασίας των καρέ του βίντεο, αναγνωρίζεται ο τετράγωνος δείκτης. Οι παράμετροι των τεσσάρων πλευρών του καθώς και τα διανύσματα που αντιστοιχούν στις τέσσερις κορυφές του αποθηκεύονται για περαιτέρω χρήσεις.

Οι περιοχές αυτές κανονικοποιούνται και η εικόνα που εμπεριέχεται στο ανιχνευθέν τετράγωνο συγκρίνεται με το σχέδιο που έχει οριστεί απο τον χρήστη. Η παρακάτω εξίσωση πινάκων αντιπροσωπεύει τη διαδικασία μετασχηματισμού για την κανονικοποίηση. Όλες οι μεταβλητές στη μήτρα μετασχηματισμού προσδιορίζονται απο την αντικατάσταση των συντεταγμένων με βαση την οθόνη και τον δείκτη των τεσσάρων κορυφών του τετραγωνου για (x_c , y_c) και (X_m, Y_m) αντίστοιχα. Μετά απο αυτό το βήμα η διαδικασία κανονικοποίησης μπορεί να ακολουθήσει βάσει του παρακάτω τύπου:

$$\begin{bmatrix} hx_c \\ hy_c \\ h \end{bmatrix} = \begin{bmatrix} N_{11} & N_{12} & N_{13} \\ N_{21} & N_{22} & N_{23} \\ N_{31} & N_{32} & 1 \end{bmatrix} \begin{bmatrix} X_m \\ Y_m \\ 1 \end{bmatrix} \quad \text{(εξ.2)}$$

Μετασχηματισμός κανονικοποίησης

Όταν δύο παράλληλες πλευρές ενός τετράγωνου δείκτη προβάλλονται στην εικόνα, οι εξισώσεις αυτών των γραμμικών τμημάτων σε συντεταγμένες κάμερας είναι:

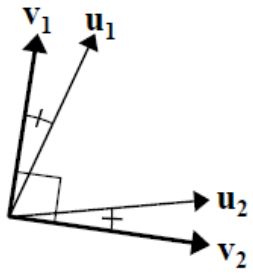
$$a_1x + b_1y + c_1 = 0, \qquad a_2x + b_2y + c_2 = 0 \quad \text{(εξ. 3)}$$

Για καθένα απο τους δείκτες, η τιμή αυτών των παραμέτρων έχει ληφθεί κατα τη διαδικασία ταιριάσματος ευθειών (line-fitting process). Δεδομένου του πίνακα προοπτικής προβολής **P** που έχει προκύψει κατα τη βαθμονόμηση της κάμερας (εξ. 4), οι εξισώσεις των επιπέδων που περιλαμβάνουν αυτές τις δύο πλευρές αντίστοιχα μπορούν να αναπαρασταθούν απο τις παρακάτω εξισώσεις στο σύστημα συντεταγμένων της κάμερας (εξ. 5), αντικαθιστώντας τα x_c, y_c στην (εξ. 4) με x και y στην (εξ. 3).

$$\mathbf{P} = \begin{bmatrix} P_{11} & P_{12} & P_{13} & 0 \\ 0 & P_{22} & P_{23} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad \begin{bmatrix} hx_c \\ hy_c \\ h \\ 1 \end{bmatrix} = \mathbf{P} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \quad \text{(εξ. 4)}$$

$$a_1P_{11}X_c + (a_1P_{12} + b_1P_{22})Y_c + (a_1P_{13} + b_1P_{23} + c_1)Z_c = 0$$
$$a_2P_{11}X_c + (a_2P_{12} + b_2P_{22})Y_c + (a_2P_{13} + b_2P_{23} + c_2)Z_c = 0 \quad \text{(εξ. 5)}$$

Δεδομένου ότι τα κανονικά διανύσματα για αυτά τα επίπεδα είναι $\mathbf{n_1}$ και $\mathbf{n_2}$ αντίστοιχα, το διάνυσμα κατεύθυνσης δύο παραλλήλων πλευρών του τετραγώνου δίνεται απο το εξωτερικό γινόμενο $\mathbf{n_1}$ x $\mathbf{n_2}$.



Εικόνα 2.14: Δύο κάθετα μοναδιαία διανύσματα: v_1, v_2 υπολογίζονται απο τα u_1 και u_2

Δύο μοναδιαία διανύσματα κατεύθυνσης που προκύπτουν απο δύο σετ δύο παράλληλων πλευρών του του τετραγωνου ($\mathbf{u_1}$ και $\mathbf{u_2}$), θα έπρεπε θεωρητικά να είναι κάθετα μεταξύ τους. Παρ΄όλα αυτά, σφάλματα που παρουσιάζονται κατα την επεξεργασία της εικόνας συνεπεται και οτι κάτι τέτοιο ενδεχομένως να μη συμβεί. Για να αντισταθμιστεί κάτι τέτοιο, δύο νεα κάθετα μοναδιαία διανύσματα κατεύθυνσης ορίζονται ($\mathbf{v_1}$, $\mathbf{v_2}$) επίπεδο που περιλαμβάνει τα $\mathbf{u_1}$ και $\mathbf{u_2}$ όπως φαίνεται και στην παραπάνω εικόνα. Δεδομένου ότι το μοναδιαίο διάνυσμα κατεύθυνσης που είναι κάθετο στα $\mathbf{v_1}$ και $\mathbf{v_2}$ είναι το $\mathbf{v_3}$, τό στοιχείο περιστροφής $\mathbf{V_{3x3}}$ στόν πίνακα μετασχηματισμού $\mathbf{T_{cm}}$ απο τις συντεταγμένες του δείκτη στις συντεταγμένες της κάμερας που ορίζονται στην (εξ. 1) είναι $[\mathbf{V_1}^t\ \mathbf{V_2}^t\ \mathbf{V_3}^t]$.

Απο τη στιγμή που το στοιχείο περιστροφής $\mathbf{V_{3x3}}$ στον πίνακα μετασχηματισμού δίνεται, χρησιμοποιώντας τις (εξ. 1) και (εξ. 4) καθώς και τις συντεταγμένες των τεσσάρων κορυφών του δείκτη στο σύστημα συντεταγμένων του δείκτη, και εκείνες τις συντεταγμένες στο σύστημα συντεταγμένων της οθόνης κάμερας, οκτώ εξισώσεις συμπεριλαμβανομένου και του στοιχείου μετάφρασης W_x W_y W_z δημιουργούνται και η τιμή αυτού του στοιχείου μετάφρασης μπορεί να ανακτηθεί απο τις εξισώσεις αυτές.

Ο πίνακας μετασχηματισμού που προκύπτει απο την παραπάνω μέθοδο μπορεί να παρουσιάζει κάποιο σφάλμα. Παρ΄όλα αυτά, μπορεί να μειωθεί με την ακόλουθη διαδικασία. Οι συντεταγμένες των κορυφών των δεικτών στο σύστημα συντεταγμένων του

δείκτη μπορούν να μετασχηματιστούν στο σύστημα συντεταγμένων της κάμερας χρησιμοποιώντας τον πινακα αυτό. Κατόπιν, αυτός ο πίνακας (μετασχηματισμου) βελτιστοποιείται καθώς το άθροισμα των διαφορών των μετασχηματισμένων συντεταγμένων και αυτων που μετρήθηκαν απο την εικόνα τείνει στο μηδέν. Παρότι υπάρχουν μόνο έξι ανεξάρτητες μεταβλητές στον πίνακα μετασχηματισμού, μόνο τα στοιχεία περιστροφής βελτιστοποιούνται και κατόπιν τα στοιχεία μετάφρασης επανεκτιμούνται χρησιμοποιώντας την παραπάνω μέθοδο. Επανάληψη αυτής της διαδικασίας για κάποιο αριθμό φορών εγγυάται πιο ακριβές αποτέλεσμα. Θα ήταν πιθανό να χρησιμοποιήσουμε και τις εξι ανεξάρτητες μεταβλητές κατα τη διαδικασία βελτιστοποίησης. Παρ' όλα αυτά, το υπολογιστικό κόστος πρέπει να ληφθεί υπόψιν.

2.10 Απαιτούμενο Hardware

Η ανάπτυξη εφαρμογών με το ARToolKit είναι χαμηλού κόστους καθώς οι απαιτήσεις σε υλικό είναι χαμηλές, και πλέον με τη διαδοση της τεχνολογίας που παρατηρείται τα τελευταία χρόνια οποιοσδήποτε σχετικά σύγχρονος υπολογιστής μπορεί να ανταπεξέλθει παραπάνω απο ικανοποιητικά.

Παρακάτω αναλύουμε κάποιο απο το υλικό που χρησιμοποιείται απο το ARToolKit.

2.10.1) Κάμερα

Η επιλογή της κάμερας είναι η πιό σημαντική. Ανάλυση, ρυθμός ανανέωσης, οπτική παραμόρφωση, είναι κάποιες απο τις παραμέτρους που πρέπει να ληφθούν υπόψιν κατά την επιλογή αυτή. Επίσης, οι υποστηριζόμενες πλατφόρμες, παράμετροι του προγράμματος – οδηγού (driver) όπως π.χ. η λειτουργία auto-contrast, είναι εξίσου σημαντικοί παράγοντες. Κάποιες κάμερες έχουν ενεργοποιημένη τη λειτουργία auto-contrast, η οποία μειώνει την απόδοση, ή παρέχει ρυθμό ανανέωσης 25Hz με αντάλλαγμα όμως την ποιότητα της εικόνας (παραμόρφωση, καθυστέρηση).

Παρακάτω παρουσιάζονται κάποιοι διαδεδομένοι τυποι αρχείων βίντεο, καθώς και οι απαιτήσεις τους σε bandwidth:

SIF RGB 15 fps	27 MBit/s (3.37 MByte/s)
SIF RGB 30 fps	55 MBit/s (6.87 MByte/s)
SIF YUV 4:1:1 15 fps	13 MBit/s (1.62 MByte/s)

SIF YUV 4:1:1 30 fps	27 MBit/s (3.37 MByte/s)
VGA RGB 15 fps	106 MBit/s (13.25 MByte/s)
VGA RGB 30 fps	221 MBit/s (26.37 MByte/s)
VGA YUV 4:1:1 15 fps	53 MBit/s (6.63 MByte/s)
VGA YUV 4:1:1 30 fps	106 MBit/s (13.25 MByte/s)

2.10.2) Head Mounted Display (HMD)

Το ARToolKit χρησιμοποιεί τεχνικές υπολογιστικής όρασης για αναγνώριση και εντοπισμό βασισμένα σε εικόνα (image-based). Οταν χρησιμοποιείται μόνο μία κάμερα, ένα αυτόνομο σύστημα που εντοπίζει τη θέση του κεφαλιού του χρήστη μπορεί να αναπτυχθεί αν η κάμερα αυτή είναι τοποθετημένη σε μία συσκευή head mounted display (HMD). Με αυτόν τον τρόπο, μπορούν να αναπτυχθούν εφαρμογές επαυξημένης πραγματικότητας, οι οποίες θα κάνουν χρήση τέτοιων συσκευών. Οι παρακάτω εικόνες απεικονίζουν δύο συσκευές τέτοιου τύπου, τα γυαλιά Virtual i-O και την Olympus EyeTrek.

Εικόνα 2.15.a : γυαλιά Virtual i-O με κάμερα

Εικόνα 2.15.b: γυαλιά Olympus EyeTrek με κάμερα

2.10.2.1) Optical vs Video see-through επαυξημένη πραγματικότητα

Τα προγράμματα που έχουν παρουσιαστεί μέχρι στιγμής χρησιμοποιούν τη λεγόμενη *video see-through* επαυξημένη πραγματικότητα, όπου τα γραφικά σχεδιάζονται πάνω απο το βίντεο του πραγματικού κόσμου. Και οι δύο παραπάνω συσκευές υποστηρίζουν είσοδο βίντεο, οπότε ένας εύκολος τρόπος να επιτευχθεί το παραπάνω είναι μέσω της σύνδεσης της συσκευής αυτης με έναν υπολογιστή που θα τρέχει την εφαρμογή ARToolKit,

απεικονίζοντας την στην οθόνη αυτού και έπειτα, μέσω της κατάλληλης μετατροπής να απεικονίζει το σήμα απο την οθόνη του υπολογιστή στην συσκευη HMD.

Το ARToolkit υποστηρίζει επίσης την *optical see-through* επαυξημένη πραγματικότητα. Όπως μαρτυρά και η ονομασία της, μόνο τα εικονικά αντικείμενα απεικονίζονται στη συσκευή HMD, και όχι και βίντεο του πραγματικού κόσμου, όπως προηγουμένως. Αντ' αυτού, τα εικονικά αντικείμενα φαίνεται να σχεδιάζονται απευθείας στον πραγματικό κόσμο, καθώς ο χρήστης έχει απευθείας οπτική επαφή με αυτόν. Κατι τέτοιο βέβαια προϋποθέτει οτι η συσκευή HMD θα είναι τουλάχιστον ημιδιαφανής, κάτι που δέ συμβαίνει με όλες τις συσκευές, αφου κάποιες περιορίζουν τελέιως την όραση του χρήστη, που βλέπει αποκλειστικά και μόνο την εσωτερική οθόνη.

Η *optical see-through* επαυξημένη πραγματικότητα παρουσιάζει πλεονεκτήματα και μειονεκτήματα σε σχέση με τη *video see-through.* Η *optical see-though* επιτρέπει στον χρήστη να δεί τα εικονικά αντικείμενα στεροσκοπικά, και όχι διοπτρικά, όπως συμβαίνει με το *video see-through* όταν χρησιμοποιείται μόνο μία κάμερα. Επίσης, ο πραγματικός κόσμος γίνεται αισθητός με την ανάλυση των ματιών αντί με την ανάλυση της οθόνης. Το βασικό μειονέκτημα με το *optical see-through* είναι η καθυστέρηση στο σύστημα. Στο *video see-through* το καρέ του βίντεο που απεικονίζεται στη συσκευή HMD είναι το ίδο καρέ που χρησιμοποιήθηκε για να υπολογιστεί η θέση και ο προσααντολισμός του κεφαλιού του χρήστη, οπότε το εικονικό αντικείμενο θα εμφανιστεί ακριβώς εκεί που πρέπει. Αυτό επιτυγχάνεται με το μη δείχνοντας το καρέ του βίντεο εως ώτου η απαιτούμενη επεξεργασία ολοκληρωθεί. Παρ' όλα αυτά, σε μία *optical see-through* εφαρμογή η άποψη του πραγματικού κόσμου δε μπορεί να καθυστερήσει, οπότε η καθυστέρηση που εισάγεται στο σύστημα απο την επεξεργασία γραφικών και εικόνας γίνεται αντιληπτή απο τον χρήστη. Αυτό έχει ως αποτέλεσμα τα εικονικά αντικείμενα να μην φαίνονται «κολλημένα» στα πραγματικά αντικείμενα με τα είναι αντιστοιχισμένα ή να φαίνεται οτι «κολυμπούν». Αυτά τα φαινόμενα καθυστέρησης γίνονται ιδιαίτερα έντονα όταν ο χρήστης κουνά απότομα το κεφάλι του ή όταν μετακινείται το πραγματικό αντικείμενο στο οποίο αντιστοιχεί το εικονικό.

3. Το OpenGL API

Η **OpenGL** (**Open G**raphics **L**ibrary) είναι ένα σύνολο προδιαγραφών που ορίζουν ενα διαπλατφορμικό API(1) (σε πολλές γλώσσες) για τη συγγραφή εφαρμογών που παράγουν δισδιάστατα και τρισδιάστατα γραφικά υπολογιστή. Η διαπροσωπία αποτελείται απο πάνω απο 250 διαφορετικές κλήσεις συναρτήσεων που μπορούν να χρησιμοποιηθούν για το σχεδιασμό πολύπλοκων τρισδιάστατων σκηνών απο απλά αρχέτυπα (primitives). Η OpenGL χρησιμοποιείται ευρέως σε εφαρμογές CAD (computer-aided design), εικονικής πραγματικότητας, επιστημονικής οπτικοποίησης, οπτικοποίησης πληροφοριών, προσομοίωσης πτήσεων, και βιντεοπαιχνίδια.

Στην περίπτωσή μας, το OpenGL χρησιμοποιείται για τη σχεδίαση όλων των εικονικών αντικειμένων, και για αυτό αξίζει να δούμε κάποια χαρακτηριστικά του, που χρησιμοποιούνται στην εφαρμογή μας και των οποίων η ανάλυση κρίνεται απαραίτητη ωστε ο αναγνώστης να κατανοήσει εις βάθος το πως ο κωδικας δίνει το αποτέλεσμα που τελικά βλεπουμε στην οθόνη μας.

Ιδιαίτερη αναφορά γίνεται στους μεταχηματισμούς που πραγματοποιεί το OpenGL, που σε συνδυασμό με τη γνώση που αποκομισε ο αναγνώστης απο το προηγούμενο κεφάλαιο μπορεί να κατανοήσei την αλληλεπίδραση του με το ARToolKit.

3.1 Σχεδίαση του API(1)

Η OpenGL εξυπηρετεί δύο βασικούς σκοπούς:

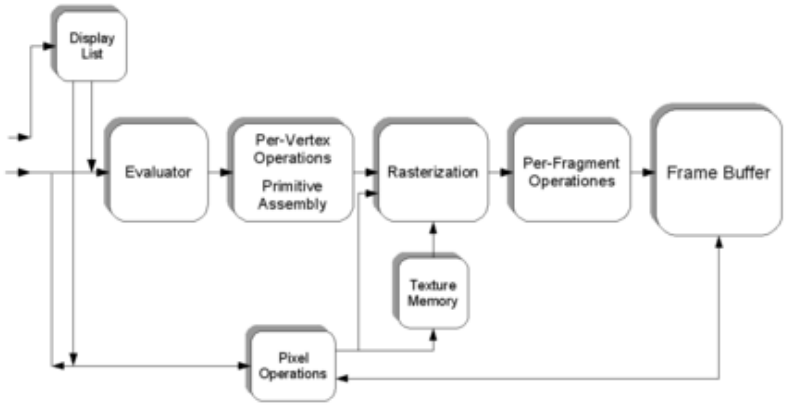
1. Να κρύβει πολυπλοκότητες της διαπροσωπίας με διαφορετικούς επιταγχυντές 3Δ παρουσιάζοντας μια απλη, ομοιόμορφη διαπροσωπία
2. Να κρύβει τις διαφέρουσες δυνατότητες των πλατφορμών hardware απαιτώντας υποστήριξη ολόκληρου του σετ δυνατοτήτων του για όλες τις εφαρμογές (implementations), χρησιμοποιώντας και εξομοίωση λογισμικού (software emulation) εάν είναι απαραίτητο.

Η βασική λειτουργία του OpenGL είναι να δέχεται αρχέτυπα όπως σημεία, γραμμές και πολύγωνα και να τα μετατρέπει σε πίξελ. Αυτό πραγματοποιείται απο τη σωλήνωση γραφικών (graphics pipeline) γνωστή και ως «μηχανή κατάστασης OpenGL» (OpenGL state machine). Η πλειονότητα των εντολών OpenGL είτε εκδίδει αρχέτυπα στην σωλήνωση γραφικών, είτε ρυθμίζει το πώς η σωλήνωση επεξεργάζεται αυτά τα αρχέτυπα. Πριν την εμφάνιση της έκδοσης 2.0 του OpenGL, κάθε στάδιο της σωλήνωσης εκτελούσε μια καθορισμένη λειτουργία και ήταν παραμετροποιήσιμο μέσα σε στενά όρια. Η έκδοση 2.0 προσφέρει διάφορα στάδια που είναι πλήρως παραμετροποιήσιμα.

Η OpenGL είναι ενα χαμηλού επιπέδου, διαδικάστικό API[(1)], που απαιτεί απο τον προγραμματιστή να υπαγορεύει τα ακριβή βήματα που απαιτούνται για να σχεδιαστεί μία σκηνή. Πρόκειται για το ακριβώς αντίθετο απο τα περιγραφικά API[(1)] (γνωστά και ως "scene graph" ή "retained graph"), όπου ο προγραμματιστής το μόνο που χρειάζεται να κάνει είναι να περιγράψει τη σκηνή αφήνοντας τη βιβλιοθήκη να χειριστεί τις λεπτομέρειες της σχεδίασής της. Ο σχεδιασμός χαμηλού επιπέδου της OpenGL απαιτεί απο τους προγραμματιστές να έχουν καλή γνώση της σωλήνωσης γραφικών, τους δίνει επίσης την ελευθερία να εφαρμόσουν καινοτόμους αλγορίθμους σχεδίασης.

Η OpenGL ιστορικά έχει ασκήσει μεγάλη επιρροή στην ανάπτυξη των επιταγχυντών 3Δ, προωθώντας ένα βασικό επίπεδο λειτουργικότητας που τώρα είναι κοινό σε υλικό που διατίθεται στην αγορά:

- Σημεία, γραμμές και πολύγωνα αποτελούν τα βασικά αρχέτυπα (primitives)
- Μια σωλήνωση μετασχηματισμού και φωτισμού (transform and lightning pipeline)
- Z-buffering
- Χαρτογράφηση υφών (texture mapping)
- Alpha blending



Εικόνα 3.1 : Απλοποιημένη εκδοχή της διεργασίας της σωλήνωσης γραφικών

Μια σύντομη περιγραφή της διεργασίας που λαμβάνει χώρα στη σωλήνωση γραφικών θα μπορούσε να είναι:

1. Αξιολόγηση, εάν είναι αναγκαίο, των πολυωνυμικών συναρτήσεων που ορίζουν συγκεκριμένες εισόδους όπως επιφάνειες NURBS, κατα προσέγγιση καμπύλες και την επιφανειακή γεωμετρία.

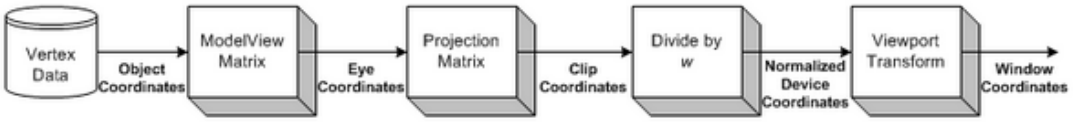
2. Χειρισμός κορυφών (vertex operations), μετασχηματισμός και φωτισμός τους που εξαρτάται απο το υλικό τους. Επίσης, «ψαλλίδισμα» των μη ορατών τμημάτων της σκηνής ώστε να παραχθεί ο όγκος θέασης (viewing volume)
3. Μετατροπή της προηγούμενης πληροφορίας σε πίξελ. Τα πολύγωνα αντιπροσωπεύονται απο το κατάλληλο χρώμα μέσω αλγορίθων παρεμβολής (interpolation algorithms).
4. Ανα τμήμα λειτουργίες, όπως η ανανέωση τιμών ανάλογα με τις εισερχόμενες και τις προηγουμένως αποθηκευμένες τιμές βάθους (depth values), ή συνδυασμούς χρωμάτων μεταξύ άλλων.
5. Τέλος, τα τμήματα εισέρχονται στο «frame buffer[4]».

Πολλοί σύγχρονοι επιταγχυντές 3Δ παρέχουν λειτουργικότητα πολύ ανώτερη απο αυτή τη βάση, αλλα αυτά τα νέα χαρακτηριστικά είναι γενικώς βελτιώσεις αυτής της βασικής σωλήνωσης παρά ριζική αναθεώρηση της.

3.2 Μετασχηματισμοί OpenGL

Οι μετασχηματισμοί στην OpenGL αποτελούν θεμελιώδεις διαδικασίες για την παραγωγή γραφικών και ρεαλιστικών σκηνών. Παρακάτω θα παρουσιάσουμε τους πιο βασικούς απο αυτούς, καθώς αποτελούν σημαντικό μέρος της εργασίας και η κατανόηση τους είναι υψιστης σημασίας και για την κατανόηση της λειτουργίας του προγράμματος που κατασκευάστηκε στα πλαίσια αυτής.

Γεωμετρικά δεδομένα όπως η θέση των κορυφών (vertex positions) και τα κανονικά διανύσματα* (normal vectors) μετασχηματίζονται μέσω της **Λειτουργίας κορυφών (vertex operation)** και **Συναρμολόγησης Αρχέτυπων (Primitive Assembly)** στην σωλήνωση της OpenGL πριν τη διαδικασία μετατροπής σε πίξελ (rasterization process):



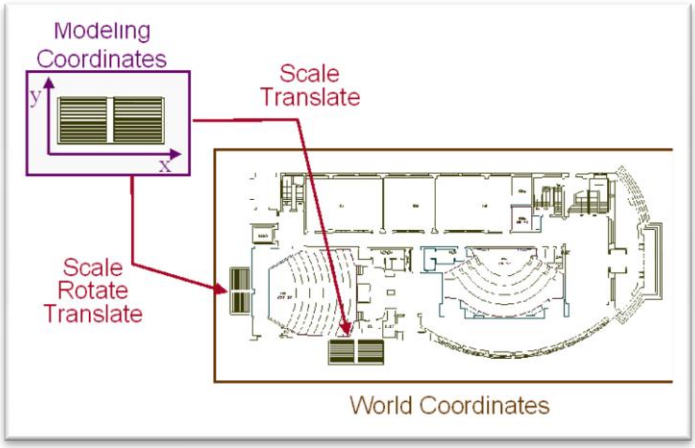
Εικόνα 3.2: μετασχηματισμός κορυφών στο OpenGL

3.2.1) Συντεταγμένες Αντικειμένου (Object Coordinates)

Είναι το τοπικό σύστημα συντεταγμένων των αντικειμένων και η αρχική θέση και προσανατολισμός προτού εφαρμοστεί οποιοσδήποτε μετασχηματισμός. Οι μετασχηματισμοί γίνονται με τις συναρτήσεις *glRotatef* (περιστροφή), *glTranslatef* (μετατόπιση της θέσης), *glScalef* (εφαρμογή κλίμακας).

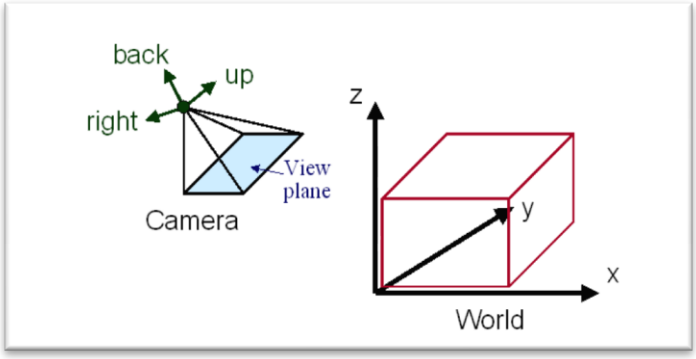
3.2.2) Συντεταγμένες Προοπτικής (Eye Coordinates)

Προκύπτει απο τον πολλαπλασιασμό του πίνακα GL_MODELVIEW με τις συντεταγμένες του αντικειμένου. Τα αντικείμενα μετασχηματίζονται απο τον χώρο του αντικειμένου στο χώρο της προοπτικής χρησιμοποιώντας τον πινακα GL_MODELVIEW. Ο πίνακας αυτός είναι συνδυασμός των πινάκων Model και View (M_{model}*M_{view}). Ο μετασχηματισμός μοντέλου (model) χρησιμοποιείται για την μετατροπή απο τον χώρο του αντικειμένου (model space) στο χώρο του κόσμου (world space).



Εικόνα 3.3: παράδειγμα μετασχηματισμού απο συντεταγμένες μοντέλου σε συντεταγμένες κόσμου

Ο μετασχηματισμός οπτικής (view) βοηθα στη μετατροπή απο τον χώρο του κόσμου στο χώρο της προοπτικής/κάμερας (eye space)



Εικόνα 3.4: συστήματα συντεταγμένων κόσμου (world) και οπτικής/κάμερας (camera)

Παρακάτω παρουσιάζεται ο μαθηματικός τυπος για το μετασχηματισμό:

$$\begin{pmatrix} x_{eye} \\ y_{eye} \\ z_{eye} \\ w_{eye} \end{pmatrix} = M_{modelView} \cdot \begin{pmatrix} x_{obj} \\ y_{obj} \\ z_{obj} \\ w_{obj} \end{pmatrix} = M_{view} \cdot M_{model} \cdot \begin{pmatrix} x_{obj} \\ y_{obj} \\ z_{obj} \\ w_{obj} \end{pmatrix}$$

Να τονιστεί οτι στο OpenGL δεν υπάρχει ξεχωριστός πίνακας για την κάμερα/προοπτική (view). Το OpenGL θεωρεί οτι η κάμερα βρίσκεται πάντα στη θέση (0,0,0) και κοιτά προς τον άξονα –Z στον χώρο προοπτικής και άρα δε μπορει να μετασχηματιστεί. (περισσότερες πληροφορίες παρακάτω.

Τα κανονικά διανύσματα* μετασχηματίζονται επίσης απο τις συντεταγμένες αντικειμένου σε συντεταγμένες προοπτικής για τον υπολογισμό φωτισμού. Υπόψιν ότι τα κανονικά διανύσματα μετασχηματίζονται με διαφορετικό τρόπο απ' 'οτι οι κορυφές. Γίνεται με τον πολλαπλασιασμό της αντιμετάθεσης (γραμμές->στήλες) του αντίστροφου πίνακα του GL_MODELVIEW με ένα κανονικό διάνυσμα[2].

$$\begin{pmatrix} nx_{eye} \\ ny_{eye} \\ nz_{eye} \\ nw_{eye} \end{pmatrix} = \left(M_{modelView}{}^{-1}\right)^{T} \cdot \begin{pmatrix} nx_{obj} \\ ny_{obj} \\ nz_{obj} \\ nw_{obj} \end{pmatrix}$$

3.2.3) Συντεταγμένες «ψαλλιδίσματος» (clip coordinates)

Είναι οι συντεταγμένες που προκύπτουν μετά την εφαρμογή των συντεταγμένων προοπτικής στον πίνακα GL_PROJECTION. Τα αντικείμενα ψαλλιδίζονται εκτός του όγκου προοπτικής (viewing frustum). Οι συντεταγμένες αυτές χρησιμοποιούνται για να καθοριστεί το πως τα αντικείμενα προβάλλονται στην οθόνη (προοπτικά ή ορθογώνια) και ποιά αντικείμενα ή μέρη αυτών αφαιρούνται απο την τελική εικόνα

$$\begin{pmatrix} x_{clip} \\ y_{clip} \\ z_{clip} \\ w_{clip} \end{pmatrix} = M_{projection} \cdot \begin{pmatrix} x_{eye} \\ y_{eye} \\ z_{eye} \\ w_{eye} \end{pmatrix}$$

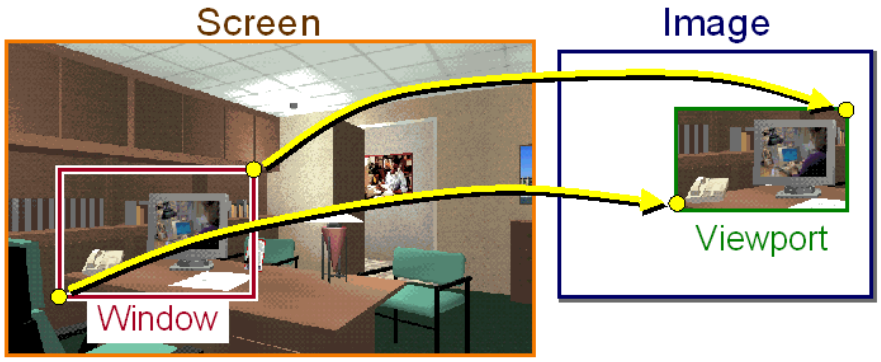
3.2.4) Κανονικοποιημένες συντεταγμένες συσκευής (Normalized Device Coordinates)

Προκύπτουν διαιρώντας τις συντεταγμένες «ψαλλιδίσματος» με *w*. Αποκαλείται προοπτική διαίρεση (perspective division). Πρόκειται περισσότερο για τις συντεταγμένες του παραθύρου της εφαρμογής, χωρίς ωστόσο ακόμα να έχουν μετατραπεί σε πίξελ οθόνης. Το εύρος τιμών είναι κανονικοποιημένο και κυμαίνεται απο -1 έως 1 και στους 3 άξονες

$$\begin{pmatrix} x_{ndc} \\ y_{ndc} \\ z_{ndc} \end{pmatrix} = \begin{pmatrix} x_{clip}/w_{clip} \\ y_{clip}/w_{clip} \\ z_{clip}/w_{clip} \end{pmatrix}$$

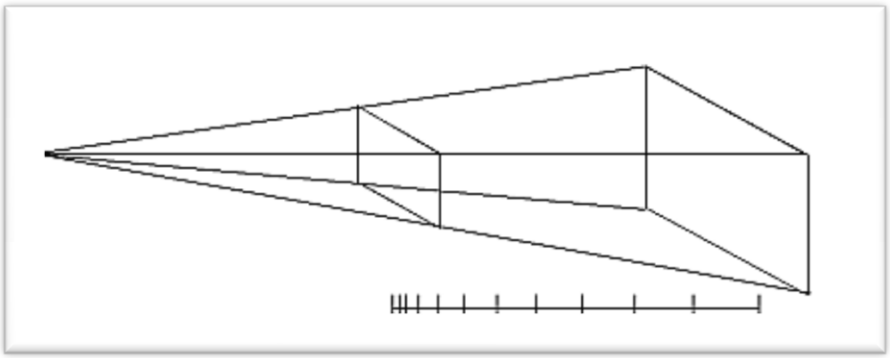
3.2.5) Συντεταγμένες παραθύρου/οθόνης (screen/window coordinates)

Προκύπτουν εφαρμόζοντας τις κανονικοποιημένες συντεταγμένες συσκευής στον μετασχηματισμο *Viewport*



Εικόνα 3.5: μετασχηματισμός *vieport*

Οι κανονικοποιημένες συντεταγμένες συσκευής μετασχηματίζονται έτσι ωστε να χωράνε στην οθόνη σχεδίασης (rendering screen). Τελικώς, οι συντεταγμένες παραθύρου περνούν απο τη διαδικασία μετατροπής σε πίξελ (rasterization) της σωλήνωσης OpenGL. Η συναρτηση *glViewport()* χρησιμοποιείται για να καθορίσει το ορθογώνιο παραλληλόγραμμο οπου γίνεται η σχεδίαση της τελικής εικόνας. Η συνάρτηση *glDepthRange()* χρησιμοποιείται για να καθορίσει την τιμή z των συντεταγμένων παραθύρου.



Εικόνα 3.6 : απόσταση συντεταγμένων βάθους

Οι συντεταγμένες παραθύρου υπολογίζονται με τις δοσμένες παραμέτρους των δύο παρακάτω συναρτήσεων:

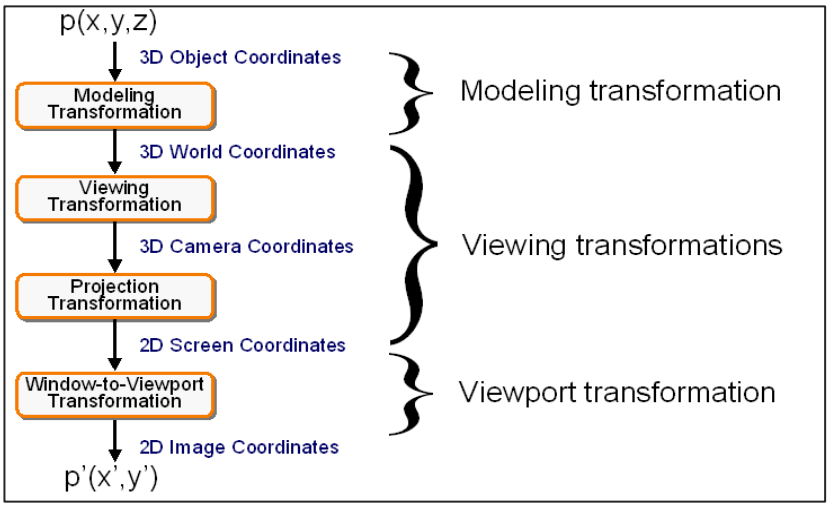
glViewport(x,y,w,h);
glDepthRange(n, f);

$$\begin{pmatrix} x_w \\ y_w \\ z_w \end{pmatrix} = \begin{pmatrix} \frac{w}{2} x_{ndc} + (x + \frac{w}{2}) \\ \frac{h}{2} y_{ndc} + (y + \frac{h}{2}) \\ \frac{f-n}{2} z_{ndc} + \frac{f+n}{2} \end{pmatrix}$$

Ο τυπος μετασχηματισμού viewport προκύπτει απλά απο τη γραμμική σχέση μεταξύ των κανονικοποιημένων συντεταγμένων συσκευής και των συντεταγμένων παραθύρου:

$$\begin{cases} -1 & \to x \\ 1 & \to x + w \end{cases} \quad \begin{cases} -1 & \to y \\ 1 & \to y + h \end{cases} \quad \begin{cases} -1 & \to n \\ 1 & \to f \end{cases}$$

Παρακάτω παρουσιάζονται διαγραμματικά όλοι οι μετασχηματισμοί:



Εικόνα 3.7 : διάγραμμα που συνοψίζει όλους τους μετασχηματισμούς που λαμβάνουν χώρα

3.2.6) Πίνακας μετασχηματισμού OpenGL

$$\begin{pmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{pmatrix}$$

Πίν. Μετασχηματισμου OpenGL

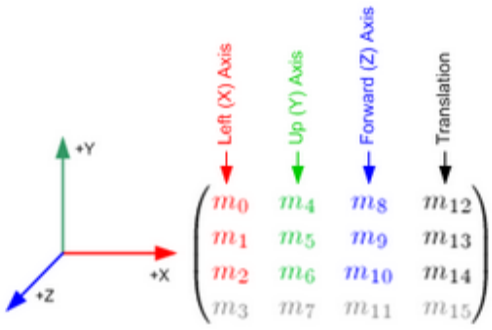
Το OpenGL χρησιμοποιεί πίνακα 4 X 4 για τους μετασχηματισμούς. Είναι σημαντικό να αναφέρουμε οτι τα 16 στοιχεία στον πίνακα αποθηκεύονται ως μονοδιάστατος πίνακας με έμφαση στις στήλες (column-major).

Το OpenGL χρησιμοποιεί 4 διαφορετικούς τύπους πινάκων: **GL_MODELVIEW,GL_PROJECTION, GL_TEXTURE** και **GL_COLOR.** Η αλλαγή μεταξύ των πινάκων αυτών γίνεται με την κλήση της συνάρτησης **glMatrixMode()** μέσα στον κώδικα. Για παράδειγμα, για να χρησιμοποιήσουμε τον πίνακα GL_MODELVIEW χρησιμοποιούμε το εξής: **glMatrixMode(GL_MODELVIEW)**

3.2.7) Πινακας μοντελου-προοπτικής ή Model-View Matrix (GL_MODELVIEW)

Ο πίνακας GL_MODELVIEW συνδυάζει τον πίνακα προοπτικής και τον πίνακα μοντέλου σε έναν πίνακα. Προκειμένου να μετασχηματίσουμε την προοπτική (κάμερα), χρειάζεται να μετακινήσουμε όλη τη σκηνή με τον αντίστροφο μετασχηματισμό (αντι να μετακινήσουμε δηλαδή την κάμερα μετακινουμε τη σχετική θέση της σκηνής ως προς αυτήν).

Τα 3 στοιχεία στην πιο δεξιά στήλη (m_{12}, m_{13}, m_{14}) χρησιμοποιούνται για το μετασχηματισμό μεταφρασης (δείχνουν την απόλυτη θέση του αντικειμένου) μέσω της συνάρτησης **glTranslatef().** Το στοιχείο m_{15} είναι η ομογενής συντεταγμένη (homogenous coordinate). Χρησιμοποιείται ειδικά για προβολικό μετασχηματισμό (projective transformation).



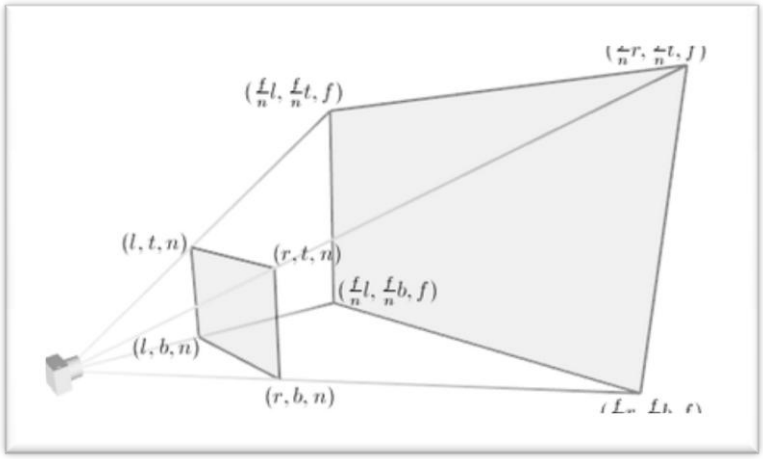
4 στήλες του πίνακα GL_MODELVIEW

3 σετ στοιχείων, (m_0, m_1, m_2), (m_4, m_5, m_6) και (m_8, m_9, m_{10}) χρησιμοποιούνται για Ευκλείδιους μετασχηματισμούς όπως περιστροφή και εφαρμογή κλιμάκας (scaling). Σημειώστε οτι αυτά τα 3 σετ αναπαριστούν ουσιαστικά τους τρεις ορθογώνιους άξονες:

- (m_0, m_1, m_2) : +X άξονας, αριστερά διάνυσμα, (1,0,0) προκαθορισμένη τιμή
- (m_4, m_5, m_6) : +Y άξονας, πανω διάνυσμα, (0,1,0) προκαθορισμένη τιμή
- (m_8, m_9, m_{10}) : +Z άξονας, μπροστά διάνυσμα, (0,0,1) προκαθορισμένη τιμή

3.2.8) Πίνακας Προβολής ή Projection Matrix (GL_PROJECTION)

Ο πίνακας GL_PROJECTION καθορίζει το οπτικό πεδίο. Το οπτικό πεδίο (frustum) προσδιορίζει ποιά αντικείμενα ή ποιά τμήματα των αντικειμένων ψαλλιδίζονται. Επίσης, καθορίζει πώς η τρισδιάστατη σκηνή προβάλλεται στην οθόνη. Η **glFrustum()** χρησιμοποιείται για προοπτική προβολή, ενώ η **glOrtho()** για ορθογραφική (παράλληλη) προβολή. Και οι δύο συναρτήσεις απαιτούν 6 παραμέτρους για να πρσδιορίσουν 6 επίπεδα «ψαλλιδίσματος»: *αριστερό, δεξιό, κάτω, πάνω, κοντινό και μακρινό* επίπεδα. 8 κορυφές του οπτικού πεδίου φαίνονται στην εικόνα που ακολουθεί:

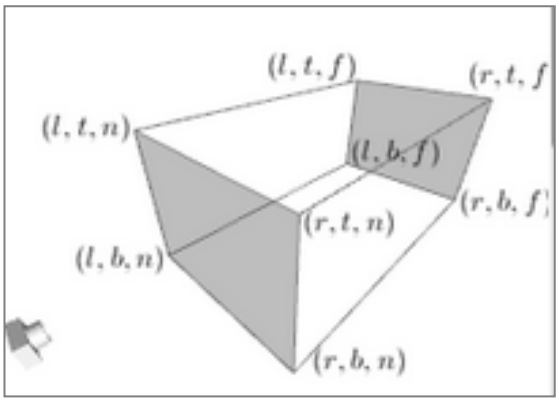


Εικόνα 3.8 : οπτικό πεδίο OpenGL

Οι κορυφές στο μακρινό επίπεδο μπορούν να υπολογιστούν απλά απο το λόγο των ομοίων τριγώνων, για παράδειγμα, το αριστερό απο το μακρινό επίπεδο είναι:

$$\frac{far}{near} = \frac{left_{far}}{left}, \quad left_{far} = \frac{far}{near} \cdot left$$

Για ορθογραφική προβολή, ο λόγος θα είναι 1, ώστε οι τιμές στο αριστερό, δεξί, πάνω και κάτω σημείο του μακρινού επιπέδου να είναι ίδιες με του κοντινού.



Εικόνα 3.9: Ορθογραφικό οπτικό πεδίο

3.2.8.1) Πλεονεκτήματα και μειονεκτήματα των δύο τύπων προοπτικών

- **Προοπτική προβολή**

+ Το μέγεθος μεταβάλλεται αντίστροφα σε σχέση με την απόσταση –δείχνει ρεαλιστικό
- Η απόσταση και οι γωνίες δεν παραμένουν (γενικώς) αμετάβλητες
- Οι παράλληλες γραμμές δεν παραμένουν (γενικώς) παράλληλες

- **Παράλληλη προβολή**

+ Καλη για ακριβείς μετρήσεις
+ Οι παράλληλες γραμμές παραμένουν παράλληλες
- Οι γωνίες (γενικώς) δε διατηρούνται
- Λιγότερο ρεαλιστική

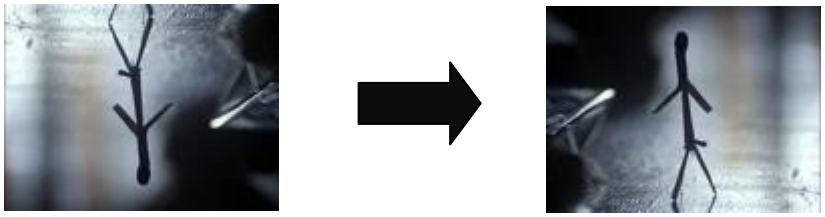
3.2.9) Πίνακας Υφών ή Texture Matrix (GL_TEXTURE)

Οι συντεταγμένες των υφών (s, t, r, q) πολλαπλασιάζονται με τον πίνακα GL_TEXTURE πρωτού γινει το *mapping υφών*. Ως προκαθορισμένη τιμή είναι η ταυτότητα (πολλ/σμος με μοναδιαίο πίνακα), έτσι η υφή θα τοποθετηθεί στα αντικείμενα ακριβώς οπως ορίζει ο

χρήστης μεσω των συντεταγμένων της υφής. Τροποιώντας τον πινακα αυτόν όμως, ο χρήστης μπορεί να περιστρέψει, «τεντώσει» και να συρρικνώσει την υφή. Ακολουθεί παράδειγμα απο τον κώδικα μας:

```
glMatrixMode(GL_TEXTURE);//flip texture from bottop-up to top-down
glScalef(1.0, -1.0, 1.0);
```

Επειδή στο προγραμμά μας τυχαίνει οι υφές που εισάγουμε απο το αρχείο βίντεο να προκύπτουν ανεστραμμένες, το παραπάνω κομμάτι κώδικα πολλαπλασιαζει τις συντεταγμένες Y της υφής με -1 ουσιαστικά αντιμεταθέτοντας την πάνω με την κάτω πλευρά:



Εικόνα 3.10: Μετασχηματισμός υφής με τη χρήση του πίνακα GL_TEXTURE

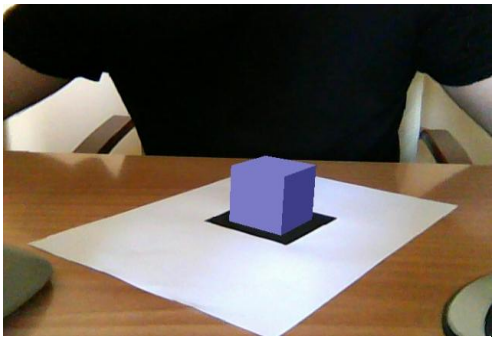
3.2.10) Πίνακας χρώματος ή Color Matrix (GL_COLOR)

Τα χρώματα (red, green, blue, alpha) πολλαπλασιάζονται με τον πίνακα GL_COLOR. Χρησιμοποιείται για χωρική χρωματική μετατροπή ή αντιμετάθεση χρωμάτων. Σπανια χρησιμοποιείται στην πράξη καθώς απαιτεί επεκτάσεις.

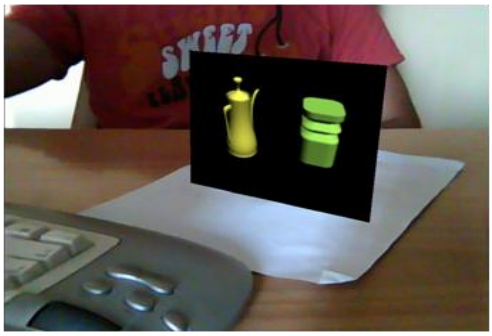
4. Η εφαρμογή

4.1 Σκοπός του προγράμματος

Στα πλαίσια της παρούσας εργασίας υλοποιήθηκε μια εφαρμογή της οποίας η λειτουργία είναι η εξής: Το πρόγραμμα αναγνωρίζει τη θέση του δείκτη (με τη βοήθεια του ARToolKit) και έπειτα σχεδιάζει ένα εικονικό αντικείμενο στην οθόνη. Η διαφορά σε σχέση με άλλες παρόμοιες εφαρμογές επαυξημένης πραγματικότητας είναι οτι το αντικείμενο που σχεδιάζουμε είναι μια απλή επιφάνεια, που όμως σαν υφή δέχεται ένα βίντεο.



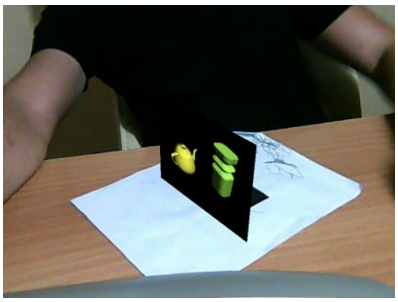
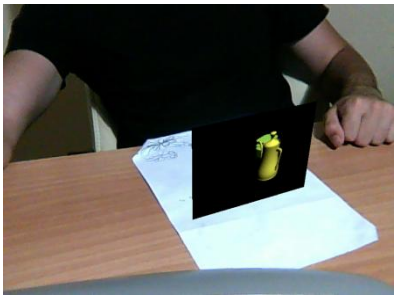
Εικόνα 4.1: η απλή εφαρμογή-παράδειγμα



Εικόνα 4.2: η εφαρμογή μας

Επίσης σημαντικό είναι οτι το βίντεο, του οποίου τα καρέ θα χρησιμοποιούνται σαν υφή θα αλλάζει ανάλογα με την κλίση του δείκτη, έτσι ώστε να δίνεται στον παρατηρητή μια τρισδιάστατη αίσθηση των εικονικών αντικειμένων που σχεδιάζονται. Προφανώς, για να επιτευχθεί κάτι τέτοιο χρειάζεται να έχουμε στη διάθεση μας βίντεο της ίδιας σκηνής απο διαφορετικές οπτικές γωνίες. Πρόκειται για παρόμοια φιλοσοφία με την κάλυψη κάποιων αγώνων football που είχε γίνει στο παρελθόν, οπου ο αγώνας καλυπτόταν απο πολλές κάμερες ταυτόχρονα (γνωστό και ως “EyeVision” βλ. http://www.youtube.com/watch?v=ohdhYEcCGVo).

Παρακάτω μπορούμε να δούμε κάποιες εικόνες της εφαρμογής μας όταν ο δείκτης έχει δύο διαφορετικούς προσανατολισμούς. Παρατηρούμε οτι δίνεται η ψευαισθηση οτι καθώς κινούμαστε αλλάζει και η άποψη απο την οποία βλέπυμε τα αντικείμενα:

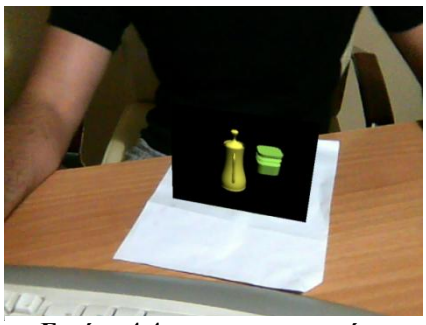



Εικόνες 4.3.α & β : η επιφάνεια μας απο δύο διαφορετικές προοπτικές

Εξίσου κρίσιμο για τη σωστή εμπειρία του χρήστη, είναι οτι πέρα απο τη μετάβαση στο κατάλληλο βίντεο ανάλογα με τον προσανατολισμό του δεικτη πρέπει το βίντεο αυτό να συνεχίζει απο το σημείο που σταμάτησε το προηγούμενο, κάτι που έχει προβλεφθεί για την εφαρμογή μας.

Τα βίντεο αυτά μπορούν να διατεθούν απο τον χρήστη (τοποθετώντας τα στο σωστό υποφάκελο) και ο αριθμός τους μπορεί να είναι οποισδηποτε. Το πρόγραμμα θα φροντίσει να τα ταξινομήσει ετσι ώστε η εμπειρία να είναι η σωστή. Το μόνο που απαιτείται απο το χρήστη είναι η σωστή ονοματοδοσία (περισσοτερα για αυτό παρακάτω)

Τέλος, στην εφαρμογή μας προστέθηκαν και καποια άλλα, δευτερεύοντα χαρακτηριστικά όπως ο τρισδιάσταος ήχος και texture masking (blending). Αμφότερα θα αναλυθούν στα αντίστοιχα κεφάλαια.



Εικόνα 4.4: απενεργοποιημένη λειτουργία blending

Εικόνα 4.5: ενεργοποιημένη λειτουργία blending

4.2 Τα βασικά τμήματα του προγράμματος μας

Το πρόγραμμά μας βασίστηκε στην απλή εφαρμογή ARToolKit (simple.exe) ο κώδικας της οποίας μπορεί να βρεθεί στο παράρτημα. Τα βασικά τμήματα του κώδικα παρουσιάζονται στο παρακάτω σχηματικό:

<table>
<tr><td>Αρχικοποίηση</td><td>1. Αρχικοποίηση της σύλληψης βίντεο και ανάγνωση των αρχείων σχεδίων δείκτη και των παραμέτρων της κάμερας</td></tr>
<tr><td rowspan="4">Main
 
Loop</td><td>2. Σύλληψη ενός καρέ απο το βίντεο.</td></tr>
<tr><td>3. Εντοπισμός του δείκτη και των αναγνωρίσιμων σχεδίων απο το το καρέ.</td></tr>
<tr><td>4. Υπολογισμός του μετασχηματισμού της κάμερας σε σχέση με τους εντοπισμένους δείκτες.</td></tr>
<tr><td>5. Σχεδιασμός των εικονικών αντικειμένων πάνω στους εντοπισμένους δείκτες</td></tr>
<tr><td>Τερματισμός</td><td>6.Τερματισμός της σύλληψης βίντεο (video capture)</td></tr>
</table>

Κατα τα επόμενα κεφάλαια θα αναλύσουμε όλες τις συναρτήσεις που χρειάστηκε να δημιουργήσουμε για τους σκοπούς της εφαρμογής μας. Προς το παρόν θα εξηγήσουμε συνοπτικά τα κομμάτια του κώδικα που εκτελούν τις παραπάνω βασικές διεργασίες.

4.2.1) Αρχικοποίηση

Αρχικά, πριν το πρόγραμμα μπει στη main loop απαιτούνται κάποια βήματα αρχικοποίησης. Παρατίθεται ο αντίστοιχος κώδικας:

```
int main(int argc, char **argv)
{


	videoNu=CountVideos();

	if(videoNu==-1) //if there are no videos, exit
	{
		showError();
		return(-1);
	}
```

```
        init();
        alutInit(&argc, argv);
        alGetError();
        SetListenerValues();
        LoadALData();
        arVideoCapStart();
        atexit(KillALData);

        printf("%d video(s) have been found....\n",videoNu);

    argMainLoop( NULL, keyEvent, mainLoop );

        return (0);
}
```

Βλεπουμε οτι πρίν τη main loop (που καλείται με την εντολή *argMainLoop(NULL,keyEvent, mainLoop)*) εκτελούνται κάποιες λειτουργίες που δε θα χρειαστεί να ξαναεκτελεστούν κατα τη διάρκεια του προγράμματος. Για παράδειγμα, μια απο αυτές τις λειτουργίες είναι η απαρίθμηση των βίντεο που ο χρήστης έχει τοποθετήσει στον κατάλληλο υποφάκελο για χρήση, όπως και το μήνυμα σφάλματος αν δεν έχει τοποθετήσει κανένα. Τέλος, αρχικοποίηση απαιτούν και τα API[(1)] που χρησιμοποιήσαμε (για παράδειγμα οι *alutInit(), alGetError(), SetListenerValues(), LoadALData()* για το OpenAL)

4.2.2) Main Loop

Πρόκειται για το πιό βασικό κομμάτι στον κώδικά μας. Είναι ο κώδικας που τρέχει επανειλλημένα μέχρι να τερματίσει το πρόγραμμα και όπου εκτελούνται όλες οι βασικές διεργασίες. Καλείται με την εντολή *argMainLoop(NULL,keyEvent, mainLoop),* οπου το δεύτερο όρισμα ορίζει τη συνάρτηση που θα χρησιμοποιηθεί ως event listerner (βλ. Ενότητα 4.2.3) και το τρίτο τη συνάρτηση που θα αποτελεί τη main loop. Παρακάτω θα αναλύσουμε τα βασικά του τμήματα (ολοκληρωμένος ο κώδικας μπορεί να βρεθεί στο παράρτημα 3):

4.2.2.1) Συλληψη καρέ απο την κάμερα

Το πρόγραμμα, αφού εισέλθει στη *main loop* προσπαθεί να «συλλάβει» βίντεο απο τον πραγματικό κόσμο, δηλαδή ό,τι βλέπει η κάμερα, και να το προβάλλει στο παράθυρο της εφαρμογής. Αυτό γίνεται κάθε φορά στην κύρια επαναληπτική διαδικασία. Παρακάτω ακολουθεί ο αντίστοιχος κώδικας:

```
/* grab a vide frame */
    if( (dataPtr = (ARUint8 *)arVideoGetImage()) == NULL ) {
        arUtilSleep(2);
        return;
    }
    if( count == 0 ) arUtilTimerReset();
    count++;

    argDrawMode2D();
    argDispImage( dataPtr, 0,0 );
```

Η σύλληψη απο την κάμερα γίνεται με τη συνάρτηση *arVideoGetImage()*, η οποία επιστρέφει ένα buffer με την προς σύλληψη εικόνα του βίντεο. Επειτα, για την απεικόνιση των καρέ αυτών φροντίζει η συνάρτηση *argDispImage().*

4.2.2.2) Εντοπισμός του δείκτη και των αναγνωρίσιμων σχεδίων απο το το καρέ

Η επόμενη λειτουργία που λαμβάνει χώρα είναι ο εντοπισμός του δείκτη μεσα στο κάρε καθώς και η αναγνώριση του σχεδίου (pattern). Αυτό υλοποιείται με το παρακάτω κομμάτι κώδικα:

```
/* detect the markers in the video frame */
    if( arDetectMarker(dataPtr, thresh, &marker_info, &marker_num) < 0 ) {
        cleanup();
        exit(0);
    }
```

Η λειτουργία όπως είναι φανερό γίνεται απο τη συνάρτηση *arDetectMarker().* Ως πρώτο όρισμα (dataPtr) δέχεται εναν pointer που δείχνει στην εικόνα στην οποία πρόκειται να αναζητηθεί ο δείκτης, ως δέυτερο μια παράμετρο (thresh) που καθορίζει πώς η εικόνα θα μετατραπεί σε διαδικό δεδομένο, ως τρίτο μία δομή (marker_info) στην οποία θα επιστραφούν όλες οι πληροφορίες σχετικά με τον εντοπισμένο δείκτη και τέλος μία ακέραια μεταβλητή (marker_num) στην οποία επιστρέφεται ο αριθμός των εντοπισμένων δεικτών (στην περίπτωση μας έχουμε μόνο ενα δείκτη).

Ιδιαίτερη μνεία πρέπει να δοθεί στη δομή ARMarkerInfo, (η μεταβλητή marker_info είναι τέτοιου τύπου) καθώς αποτελεί τη εφαρμογή της θεωρίας στην πράξη και θα βοηθήσει στην εις βάθος κατανόηση του προγράμματος και της λογικής του ARToolKit. Ας δούμε το πως ορίζεται στη γλώσσα προγραμματισμού:

```
typedef struct {
    int     area;
    int     id;
    int     dir;
    double  cf;
    double  pos[2];
    double  line[4][3];
    double  vertex[4][2];
} ARMarkerInfo;
```

Παρακάτω θα παρουσιάσουμε εναν πίνακα που επεξηγεί κάθε μεταβλητή που ορίζεται στη δομή αυτή:

Τύπος Δεδομένων	**Ονομασία**	**Περιγραφή**
int	area	Ο αριθμός των πίξελ στην περιοχή του δείκτη
int	id	Αριθμός ταυτότητας δείκτη
int	dir	Πιθανές τιμές είναι οι 0, 1, 2 ή 3. Καθιστά δυνατή την ευρεση της σειράς των γραμμών ώστε να βρεθεί η πρώτη κορυφή. Είναι πολυ σημαντική για τον υπολογισμό του πίνακα μετασχηματισμού
double	cf	τιμή εμπιστοσύνης (πιθανότητα να είναι δείκτης
double	pos[2]	κέντρο του δείκτη (σε ιδανικές συντεταγμένες οθόνης)
double	line[4][3]	Εξισώσεις γραμμής για τις τέσσερις πλευρές του δείκτη ax + by + c = 0
double	vertex[4][2]	Οι κορυφές του δείκτη

Παρατηρούμε οτι στη δομή αυτή περιέχονται μεγέθη που αναλύσαμε στη θεωρία (βλ. Κεφ. 2)

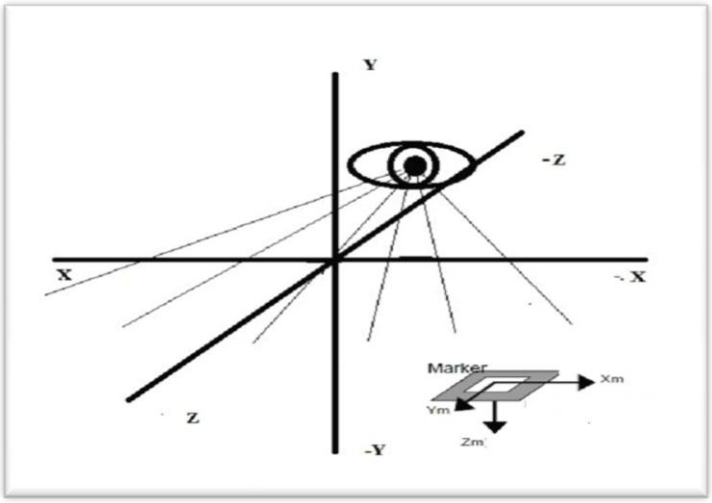
4.2.2.3) Υπολογισμός του μετασχηματισμού της κάμερας σε σχέση με τους εντοπισμένους δείκτες.

Το επόμενο βήμα είναι ο υπολογισμός του μετασχηματισμού κάμερας, που είναι απαραίτητος ώστε αργότερα, τα εικονικά σχήματα που είναι προς σχεδίαση να σχεδιαστούν

στη σωστή θέση (δηλαδή, πάνω απο το δείκτη). Αυτο πραγματοποιείται με τον παρακάτω τμήμα κώδικα:

```
/* get the transformation between the marker and the real camera */
    if( mode == 0 || contF == 0 ) {
        arGetTransMat(&marker_info[k], patt_center, patt_width, patt_trans);
    }
    else {
        arGetTransMatCont(&marker_info[k], patt_trans, patt_center, patt_width,
patt_trans);
    }
    contF = 1;
```

Η συνάρτηση που επιτελεί τη λειτουργία του μετασχηματισμού είναι η *arGetTransMat()*, που έχει ως πρώτο όρισμα τις πληροφορίες που συλλέξαμε κατα τον εντοπισμό του δείκτη. Ο μετασχηματισμός επιστρέφει στο τέταρτο όρισμα (patt_trans) που πρόκειται για έναν πίνακα 3X4, ο οποίος περιέχει την απόλυτη θέση του δείκτη σε σχέση με την κάμερα, καθώς και τις συνιστώσες των τριών μοναδιαίων διανυσμάτων που ορίζουν το σύστημα συντεταγμένων του δείκτη, στο σύστημα συντεταγμένων της κάμερας



Εικόνα 4.6: Ο δείκτης (marker) στο συστ. συντεταγμένων της κάμερας

Ο πίνακας που επιστρέφεται (patt_trans) είναι ο παρακάτω:

$$\begin{pmatrix} a1 & a2 & a3 & a4 \\ a5 & a6 & a7 & a8 \\ a9 & a10 & a11 & a12 \end{pmatrix}$$

Τα στοιχεία (*a1, a5, a9)* είναι οι συνιστώσες (X,Y,Z) του διανύσματος X_m του δείκτη στο σύστημα συντεταγμένων της κάμερας. Αντίστοιχα, τα στοιχεία (*α2, α6, α10*) είναι για το διάνυσμα Y_m, ενώ τα (*α3, α7, α11*) για το διάνυσμα Z_m. Η τελευταία στήλη (*α4, α8, α12*) περιέχει την απόλυτη θέση του δείκτη στο συστημα συντεταγμένων της κάμερας.

Οπως γίνεται κατανοητό, ο προσανατολισμός (που θα μας απασχολήσει ιδαίτερα αργότερα) μπορεί να προκύψει μόνο απο τα διανύσματα του δείκτη και όχι απο τη θέση αυτού.

4.2.2.4) Επιλογή του κατάλληλου βίντεο, σχεδιασμός της σκηνής και ανανέωση των παραμέτρων ήχου

Σε αυτό το σημείο (που προκειται και για το τελ διαφοροποιείται το προγραμμά μας δραματικά σε σχέση με το απλο πρόγραμμα που απεικονίζει ενα απλό κύβο. Ας ρίξουμε μια ματιά στον κώδικα:

```
angleToScene();      //choose a scene based on the angle


       if (FirstTime)//run only the first time
       {
              chooseAVI();
              getVideoProperties(input_video);
              FirstTime=!FirstTime;
       }

       //if the scene has changed, switch to the correspondent video
       if (new_scene!=old_scene)
              chooseAVI();
       SetSourceValues();
       draw();
       argSwapBuffers();
```

Σαν πρώτη συνάρτηση συναντάμε την *angleToScene()*, η οποία ουσιαστικά χωρίζει το επίπεδο έτσι ώστε ανάλογα με την κλίση του δείκτη, να προβάλλεται και διαφορετικό βίντεο. Αυτό που κάνει είναι να έχει ώς είσοδο την κλίση αυτή και «σαρώνοντας» το οριζόντιο ημι-επίπεδο X-Z (με Z θετικό, καθως για Z αρνητικό ο δεικτης θα βρισκεται πίσω απο τη κάμερα και άρα θα είναι μη ορατός) να διαλεγει το κατάλληλο βίντεο. Ο διαχωρισμός αυτός γίνεται βάσει του αριθμού των βίντεο που βρίσκονται στον καταλληλο υποφάκελο του προγράμματος, οπως θα δούμε αργότερα πιο αναλυτικά. Τα τμήματα που χωρίζεται το επίπεδο ονομάζονται (για τους σκοπύς της εργασίας) *σκηνές (scenes).*

Επειτα, έχουμε μια συνθήκη που εκτελείται μόνο στην πρώτη επανάληψη της main loop. Ο λόγος που οι εντολες που βρίσκονται μέσα στη συνθήκη αυτη δεν εκτελουνται κατα τη φάση της αρχικοποίησης είναι το γεγονός οτι τα δεδομένα που χρειάζονται ανακτώνται όταν η main loop εκτελείται για αυτήν την πρώτη φορά.

Πρόκειται για τη συναρτηση *chooseAVI()*, μια συνάρτηση που ανοίγεί το stream την για αναπαραγωγή του κατάλληλου βίντεο, αφου έχει γινει γνωστό ποιο πρέπει να ειναι αυτό, απο την *angleToScene().* Η *getVideoProperties(),* όπως υποδηλώνει και η ονομασία της, ανακτά κάποιες πληροφορίες σχετικά με το βίντεο (οπως τη διάρκεια, τα καρέ/δευτ.). Εκτελειται μόνο στην αρχή, καθώς τα βίντεο μας είναι ακριβώς τα ίδια σε τεχνικά χαρακτηριστικά. Οποτε, κατα την εναλλαγη μεταξύ των βίντεο δεν τίθεται κάποιο πρόβλημα λανθασμένων πληροφοριών

Πολύ σημαντικό κομμάτι στον κώδικα, αποτελεί ο έλεγχος *if(new_scene!=old_scene)*, το οποίο δικαιολογεί και το γιατί οι συναρτήσεις *angleToScene()* και *ChooseAVI()* είναι ξεχωριστές και όχι μία συνάρτηση. Ο έλεγχος αυτός ουσιαστικά ελέγχει εάν άλλαξε σκηνή (δηλαδη εαν πρέπει να αλλάξει το βίντεο) και μόνο τότε ανοίγει stream για κάποιο άλλο βίντεο. Αν όχι, το πρόγραμμα συνεχίζει ως έχει. Γίνεται κατανοητό, ότι αν οι δυο συναρτήσεις που αναφέρθηκαν ήταν μία, και δε γινόταν ο παραπάων έλεγχος, σε κάθε επανάληψη της main loop θα άνοιγε συνεχώς stream ακόμα και προς το ίδιο βίντεο, που θα είχε ως αποτέλεσμα να μην εκτελείται σωστά η εφαρμογή μας.

Προχωρώντας παρακάτω, η συνάρτηση *SetSourceValues()* ταυτίζει τις συντεταγμένες της πηγής ήχου με αυτές του δείκτη για τη σωστή αναπαραγωγή τρισδιάστατου ήχου (περισσότερα στο αντίστοιχο κεφάλαιο) ενώ τέλος, η *draw()* σχεδιάζει τα εικονικά αντικείμενα.

4.2.3) Τερματισμός και Event Listener

Όταν το πρόγραμμα μας πρόκειται να τερματιστεί, κάποιες συναρτήσεις «εκκαθάρισης» πρέπει να εκτελεστούν ώστε να αποφευχθούν πιθανά προβλήματα στον υπολογιστή της χρήστη οπως memory leaks κλπ.

Γεννάται το ερώτημα για το πως είναι δυνατόν να διακόψουμε τη συνεχή επανάληψη της main loop. Αυτό γίνεται με τον event listener, ενα κομμάτι κώδικα που εκτελείται όταν υπάρχει μια είσοδος στον υπολογιστή (απο συσκευες εισόδου όπως το πληκτρολόγιο) κατα τη διάρκεια της εκτέλεσης. Ο κωδικας αυτός έχει προσαρμοστεί ωστε να εκτελεί μια σειρά εντολών στο πάτημα συγκεκριμένων πληκτρων. Ας δούμε τον κώδικα:

```
static void   keyEvent( unsigned char key, int x, int y)
{

    /* quit if the ESC key is pressed */
    if( key == 0x1b ) {
        printf("*** %f (frame/sec)\n", (double)count/arUtilTimer());
        cleanup();
              exit(0);
       }

       /*turn on/off audio if 'p' key is pressed*/
       if( key == 0x70 ) {
              if (audio)
                     alSourcePause(Source);
              else
                     alSourcePlay(Source);
              audio=!audio;
       }

       /*turn blending on/off if 'b' is pressed*/
       if( key == 0x62 )
              blend=!blend;

}
```

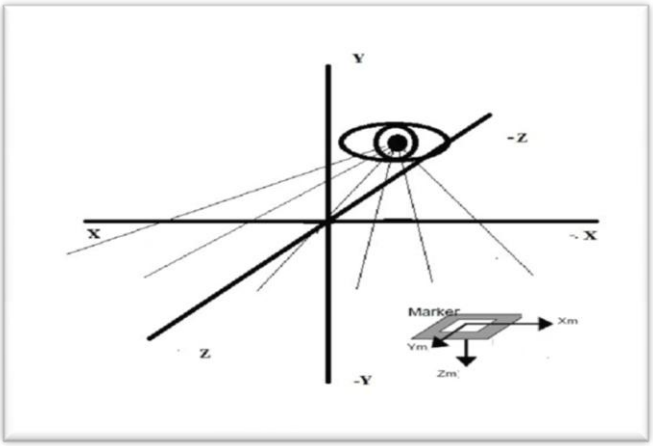
Ουσιαστικά, πρόκειται για μια συνάρτηση που ελέγχει εαν πατηθηκε κάποιο απο τα επιθυμητά πλήκτρα. Ο πρώτος έλεγχος αφορα στο πάτημα του πλήκτρου “esc” (κώδικας ASCII = 0x1b), που οδηγεί στην εμφάνιση ενός μηνύματος που περιέχει καποια στατιστικά στοιχεία καθως και στην εκτέλεση των συναρτήσεων «εκκαθάρισης» (*cleanup()*). Τέλος, τερματίζει το πρόγραμμα.

Επιπλέον για τους σκοπούς της εφαρμογης μας έχουν προστεθεί και κάποιες επιπροσθετες λειτουργίες. Εάν για παράδειγμα πατηθεί ο χαρακτήρας "c" (κωδικας ASCII = 0x70) τότε ξεκινά/σταματά η αναπαραγωγή του ήχου (*alSourcePlay ή alSourcePause()* αντίστοιχα), ενώ αν πατηθεί ο χαρακτήρας "b" τότε ενεργοποιείται/απενεργοποιείται η λειτουργία blending, μέσω της global μεταβλητής "blend" (βλ. Περισσότερα στο κεφάλαιο 4.3.4)

4.3 Μια πιο αναλυτική ματιά στον κώδικα

4.3.1) Πώς μοιράζεται το επίπεδο

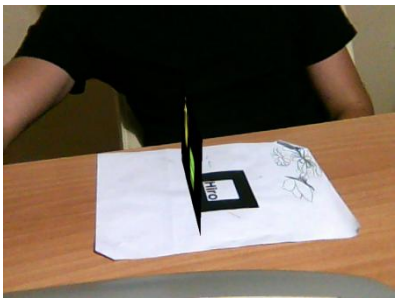
Όπως αναφέραμε και σε προηγούμενες ενότητες, για να δοθεί μια τρισδιάστατη αίσθηση στο χρήστη, πρέπει να διαθέτουμε εναν αριθμό βίντεο της ίδιας σκηνής απο διαφορετική οπτική γωνία. Οπότε είναι απαραίτητο να «μοιράσουμε» το επίπεδο σε κάποια τμήματα, ετσι ώστε οταν ο δείκτης δείχνει μέσα σε αυτό να αναπαράγεται και το κατάλληλο βίντεο



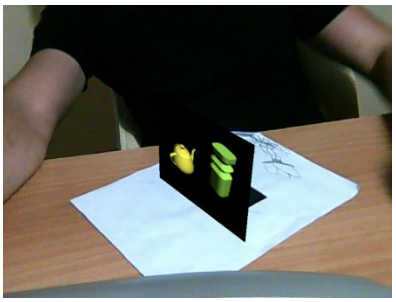
Εικόνα 4.7: Το σύστημα συντεταγμένων της κάμερας και τα μοναδιαία διανύσματα του δείκτη

Για να επιτευχθεί αυτό διαλέγουμε ένα απο τα διανύσματα του δείκτη (για λόγους ευκολίας θα εργαστούμε με το X_m), και υπολογίζουμε τη γωνία που αυτό σχηματίζει σε σχέση με την κάμερα/παρατηρητή, στο οριζόντιο επίπεδο XZ (η κλίση στο YZ δε μας ενδιαφέρει).

Η σάρωση ξεκινά απο το σημείο που το διανυσμα X_m έχει τη κατεύθυνση του άξονα Z , και κίνειται αριστερόστροφα μέχρι να έχει τη κατεύθυνση του άξονα –Z. Για να γίνει πιο κατανοητό, ουσιαστικά ξεκινάμε απο το σημείο που ο δείκτης «κοιτά» αριστερά (οπως ο άξονας –X) και καταλήγει να κοιτά στο X.



Εικόνα 4.8.α



Εικόνα 4.8.β

Εικόνα 4.8.γ

Εικόνα 4.8.δ

Εικόνα 4.8.ε

Στις παραπάνω εικόνες βλεπουμε αυτή τη μετάβαση: Στην εικόνα 4.8.α βλέπουμε οτι ο δείκτης κοιτά αριστερά (δεξιά για τον παρατηρητή), ενω κινείται ορολογιακά (αντι-ορολογιακά για τον παρατηρητή) και καταλήγει να κοιτα δεξιά (αριστερά για τον παρατηρητή (εικόνες 4.8 α-ε)

Στο πρόγραμμα μας επιλέξαμε να σχεδιάζεται μόνο η εμπρός όψη της επιφάνειας και γι' αυτό ο δείκτης «σαρώνει» μόνο σε διάστημα 180 μοιρών και όχι 360.

Ας ρίξουμε μια ματιά στις συναρτήσεις που υλοποιούν αυτή τη διαδικασία:

4.3.1.1) Συνάρτηση *CountVideos()*

Πρωτού μοιράσουμε το επίπεδο θα πρέπει να γνωρίζουμε σε πόσα τμήματα θα γίνει αυτός ο διαμοιρασμός. Αυτό εξαρτάται απο τον αριθμό των αρχείων βίντεο που θέλει να χρησιμοποιήσει ο χρήστης στην εφαρμογή (εαν έχουμε για παράδειιγμα 6 βίντεο το διαστημα των 180 μοιρών θα διαμοιραστεί σε 6 τμηματα των 30 μοιρών). Υπεύθυνη για την καταμέτρηση του πληθους των αρχείων είναι η *CountVideos(),* ο κώδικας της οποίας παρουσιάζεται παρακάτω:

```
int CountVideos()
{
	static int count=0,name,check;
	hFind=FindFirstFile("Data/*.avi", &FindFileData);
	if(hFind==INVALID_HANDLE_VALUE)
	{
		return -1;
	}
	check=sscanf(FindFileData.cFileName,"%d",&name);
	if(check!=0)
	{
		videoNames[count]=name;
		count++;
	}
	while (FindNextFile(hFind, &FindFileData) != 0)
	{
		check=sscanf(FindFileData.cFileName,"%d",&name);
		if(check!=0)
		{
			videoNames[count]=name;
			count++;
		}
	}
	qsort (videoNames, count, sizeof(int), compare);
	return count;

}
```

Η συνάρτηση αυτή χρησιμοποιεί ως επι το πλειστον συναρτήσεις του λειτουργικού συστήματος Windows©. Να τονίσουμε οτι η συνάρτηση αναζητά αρχεία που έχουν ως

όνομα καποιον ακέραιο αριθμό, ενώ τα υπολοιπα αρχεία αγνοούνται. Η συνάρτηση *FindFirstFile("Data/*.avi",&FindFIleData)* αναζητά το πρώτο αρχείο που βρίσκεται στον υποφακελο "Data" και που έχει ως ονομα οτιδήποτε, αρκεί να έχει καταληξη '.avi'. Οι πληροφορίες αποθηκεύονται στην παράμετρο 'FindFileData'. Επειτα, με τη βοήθεια της *sscanf* ελέγχεται αν το όνομα περιέχει αριθμό. Αν ναι αυτός αποθηκεύεται στον πίνακα με τα ονόματα των βιντεο ("videonames"). Αν δε βρεθεί αριθμός στο όνομα, τότε το αρχείο αγνοείται. Αυτό γίνεται με σκοπό την εύκολη ταξινόμηση των αρχείων.

Η ίδια λειτουργία επιτελείται και για τα επόμενα βίντεο μέχρι η διαδικασία να μη βρίσκει άλλο αρχείο. Τελος, τα ονόματα των βίντεο ταξινομούνται σε άυξουσα σειρα (για παράδειγμα το $2^{ο}$ στοιχείο του πίνακα "videonames" θα έχει ως όνομα αριθμό μικρότερο απο αυτο του $3^{ου}$ στοιχείου). Αυτό επιτελείται με τη συνάρτηση *qsort(),* και είναι απαραίτητο καθώς οι συναρτήσεις του λειτουργικού δεν εξασφαλίζουν οτι τα αρχεία θα αναζητηθουν βάσει ονόματος.

Στο τέλος η συνάρτηση επιστέφει τον αριθμό των βίντεο.

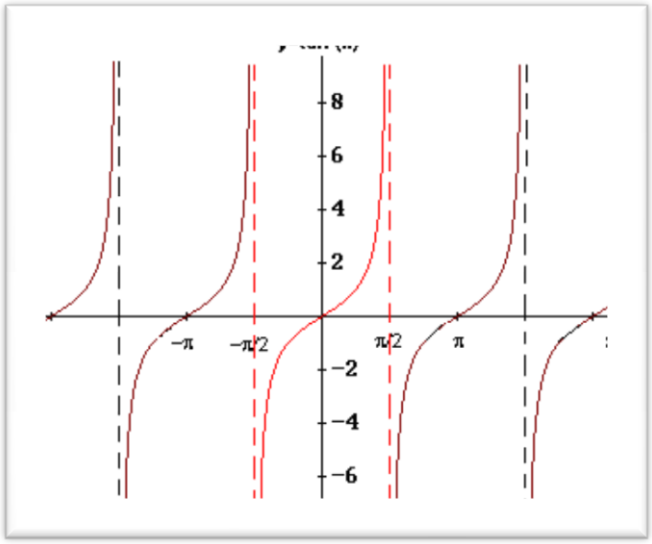
4.3.1.2) Συνάρτηση *angleToScene()*

Η συνάρτηση angleToScene() είναι αυτή η συνάρτηση που μας γνωστοποιεί σε ποιό απο τα τμήματα του επιπέδου κοιτα το διάνυσμα X_m, οποτε το προγραμμα θα γνωρίζει ποιό βίντεο θα πρέπει να αναπαραχθεί. Παρατίθεται ο κώδικας:

```
static void angleToScene (void) //matches the angle to a scene
{
	static int i;

	//calculate the camera direction
	marker_angle=patt_trans[2][0]/patt_trans[0][0];
	for (i=0;i<videoNu;i++)
	{

		if ( marker_angle<tan((3*PI/2)+(i+1)*(PI)/videoNu) )
		{
			new_scene=i;
			break;
		}
	}

}
```

Παρατηρούμε οτι η πρώτη εντολή αφορά την κλίση του διανύσματος X_m (υπενθυμίζουμε οτι κλιση ισούται με την εφαπτομένη η οποία στην περίπτωση μας είναι ιση με $\frac{Z}{X}$). Έπειτα καλείται μια επαναληπτική διαδικασία που εκτελείται τόσες φορές όσες το πλήθος των αρχείων βιντεο. Σε επόμενη φάση εκτελείται μια εντολή συνθήκης η οποία εξετάζει σε ποιο διαστημα ανήκει η κλιση. Για παράδειγμα, αν έχουμε 4 βίντεο, κατα την πρώτη επανάληψη εξετάζεται εαν η κλιση ειναι μικρότερη απο tan(-π/4), ενω στις επομενες μικροτερη απο tan(0), tan(π/4) και tan(π/2). Διαπιστώνουμε οτι το επίπεδο μοιράζεται σε 4 ίσα διαστήματα των π/4 (4*π/4 = π).

Στο σημείο αυτό γίνεται προφανές γιατί διαλέξαμε το διανυσμα X_m. Οπως είναι γνωστό απο την τριγωνομετρία, η εφαπτομένη είναι αύξουσα απο –π/2 εως π/2 (διαστημα το οποίο τυχαίνει να είναι και η περίοδος της). Με αυτόν τον τρόπο μπορουμε να κάνουμε τον έλεγχο σε μια γραμμή κώδικα, χωρίς περιττές εντολές.



Εικόνα 4.9: η γραφική παράσταση της εφαπτομένης (tan)

Κατόπιν, όταν βρεθεί το διάστημα, η συνθήκη τερματίζει και ορίζεται ως νεα σκηνή η κατάλληλη. Η κωδικοποίηση των διαστημάτων επομένως είναι η εξής: το πρώτο διάστημα ονομάζεται *σκηνη 0*, το δευτερο *σκηνή 1* ενω για ν βιντεο το ν-οστο διάστημα *σκηνή (ν-1).*

Να υπενθυμίσουμε οτι η συνάρτηση *angleToScene()* εκτελείται σε κάθε επανάληψη της main loop καθώς πρέπει ανα πάσα στιγμή τον προσανατολισμό του δείκτη, κάτι που δε συμβαίνει με την επόμενη συνάρτηση.

4.3.2) Αναπαραγωγή των αρχείων βιντεο

Τα αρχεία βίντεο αναπαράγονται με τη βοήθεια του OpenCV API[(1)], που μας δίνει μεγάλη ευελιξία πάνω σε αυτά, καθώς με απλές εντολές μπορεί ο προγραμματιστής να επιτελέσει διάφορες λειτουργίες οπως το ανοιγμα, κλείσιμο του αρχείου καθώς και να ανακτήσει χαρακτηριστικά όπως τη διάρκεια, τα καρε/δευτ κλπ. Οι επόμενες συναρτήσεις κάνουν έντονη χρήση αυτού του API[(1)]

4.3.2.1) συνάρτηση *ChooseAVI()*

Η συνάρτηση αυτή είναι υπευθυνη για την εκκίνηση του stream απο το σωστό αρχείο βίντεο. Παρακάτω παρουσιάζεται ο κώδικας αυτής:

```
static void chooseAVI (void) //chooses the right .avi file
{
	static char   path[20];
	static double FrameStamp;
	old_scene=new_scene;

	//save the point (exact frame) where the last video has stopped
	FrameStamp=cvGetCaptureProperty(input_video, CV_CAP_PROP_POS_FRAMES );

	// each scene maps to the correspodent video (the first scene maps to the
first video etc.)

	sprintf(path,"Data/%d.avi",videoNames[new_scene]);

	//open the video stream
	input_video=cvCaptureFromAVI(path);
	input_video2=cvCaptureFromAVI(path);

	//continue from the point where the last video has stopped
	cvSetCaptureProperty(input_video,CV_CAP_PROP_POS_FRAMES,FrameStamp);
	cvSetCaptureProperty(input_video2,CV_CAP_PROP_POS_FRAMES,FrameStamp);
}
```

Ξεκινώντας την ανάλυση του κώδικα, βλεπουμε αρχικά οτι η παλιά σκηνή (δηλαδή το παλίο διάστημα της εφαπτομένης) γίνεται ίσο με τη σκηνή στην οποία εισείλθαμε, ετσί ώστε ο έλεγχος που γίνεται στη main loop να λειτουργήσει οπως πρέπει. Σε περίπτωση που δε γινόταν αυτή η ανάθεση το πρόγραμμα δε θα ήξερε οτι δεν έχει αλλάξει η σκηνή και η συναρτηση *chooseAVI()* θα καλούνταν συνεχώς με ανεπιθύμητα αποτελέσματα.

Σε επόμενο βήμα, εκτελείται μια συνάρτηση που αποθηκεύει το σημείο του βίντεο που αναπαραγόταν πρίν αλλάξει η σκηνή, ώστε το επόμενο βίντεο να συνεχίσει απο το σημείο εκείνο και όχι ξανά απο την αρχή. Το σημείο αποθηκεύεται στην μεταβλητή *FrameStamp,* και πρόκειται για το τελευταίο κάρε που αναπαράχθηκε προτού αντικατασταθεί το βίντεο (ο λόγος για τη συνάρτηση cvGetCaptureProperty() με πρώτο όρισμα το stream του βίντεο (input_video) και δεύτερο τη συμβολική παράμετρο του OpenCV, CV_CAP_PROP_POS_FRAMES).

Συνεχίζοντας, σε επόμενη φάση εκτελείται η εκκίνηση του stream απο το βίντεο. Οπως είδαμε και στη συνάρητηση *CountVideos(),* τα βίντεο πρέπει να έχουν ως όνομα κάποιον ακέραιο αριθμό ακολουθούμενο απο την επέκταση .avi (για παραδειγμα 4.avi). Ολα τα ονόματα των βίντεο αποθηκεύονται σε έναν πίνακα (videoNames) με αύξουσα σειρα (αν έχουμε π.χ. 3 αρχεία βίντεο με ονόματα 6.avi, 3.avi, 0.avi αυτά θα τοποθετηθούν στον πίνακα ως εξής: 0.avi, 3.avi, 6.avi). ως αποτέλεσμα το μ-οστο στοιχείο του πίνακα θα περιέχει το μ μικρότερο βασει ονόματος βίντεο και θα ανιστοιχεί στη σκηνή (μ-1). Οπως θα διαπιστώσουμε αυτή η αντιστοίχιση διευκολύνει τον χρήστη, όταν επιθυμει να τοποθετήσει τα δικά του αρχεία βίντεο

Παρατηρούμε οτι το όνομα του κατάλληλου βίντεο (που είναι ακεραιος αριθμός) εισάγεται σε μια συμβολοακολουθία (μεσω της *sprintf()*) η οποία αποτελεί το path προς το αρχείο (πχ αν το στοιχείο του πινακα περιέχει το 23 τότε το path θα είναι το “Data/23.avi”).

Να τονίσουμε οτι δημιουργούμε 2 streams προς το βίντεο, με το δευτερο να είναι απαραίτητο για τη λειτουργία “blending” οπως θα αναλύσουμε αργότερα.

Τέλος, η συνάρτηση μεταφέρει το σημείο αναπαραγωγής στο ίδιο σημείο που σταμάτησε το προηγούμενο βίντεο ωστε να έχουμε την ομαλή εκτέλεση που επιθυμούμε. (αυτο πραγματοποιείται με τη συνάρτηση *cvSetCaptureProperty*()).

4.3.2.2) Συνάρτηση *GetVideoProperties()*

Η συνάρτηση αυτή βοηθά στην ανάκτηση πληροφοριών απο τα αρχεία βίντεο όπως τα καρέ/δευτ. Καθώς και τον συνολικό αριθμό των καρέ που περιέχουν αυτά. Οι πληροφορίες αυτές είναι ζωτικής σημασίας καθώς χρησιμοποιούνται σε πολλά σημεία στον κώδικά μας. Παρακάτω η συναρτηση:

```
void getVideoProperties(CvCapture *input_video)
{

              //Get the total frame number from the video
              NumFrames = (int) cvGetCaptureProperty(input_video,
CV_CAP_PROP_FRAME_COUNT);

              //get the video fps
              FPS =  cvGetCaptureProperty(input_video,  CV_CAP_PROP_FPS);
}
```

Ο αριθμός των καρέ του βίντεο επιστρέφεται στη μεταβλητή 'NumFrames' απο τη συνάρτηση *cvGetCaptureProperty(),* με όρισμα το stream καθώς και την παράμετρο CV_CAP_PROP_FRAME_COUNT. Με την ίδια συναρτηση επιστρέφονται και τα καρέ/δευτ., αυτή τη φορά με όρισμα το CV_CAP_PROP_FPS, στη μεταβλητή 'FPS'. Η συνάρτηση αυτή εκτελείται μόνο μια φορά στη main loop, καθώς τα βίντεο έχουν πανομοιότυπα τεχνικά χαρακτηριστικά και περαιτέρω εκτελέσεις της δεν ικανοποιούν κάποια σκοπιμότητα.

4.3.2.3) Συνάρτηση *UpdateTexture()*

Σε αυτό το σημείο πρέπει να γίνει γνωστό οτι το βίντεο δεν απεικονίζεται απευθείας στην εικονική επιφάνεια της εφαρμογής. Αυτο που συμβαίνει ειναι οτι κάθε καρέ του διαβάζεται ξεχωριστά και έπειτα μετατρέπεται σε υφή (texture) ωστε να εφαρμοστεί στην επιφάνεια μας. Η συνάρτηση που πραγματοποιεί τη λειτουργία αυτή είναι η *UpdateTexture().*

Εκτός αυτού, η συνάρτηση αυτή φροντίζεί για τη ν επαναλαμβανόμενη αναπαραγωγή του βιντεο, δηλαδή όταν η αναπαραγωγή φτάσει στο τέλος της διάρκειας τοτε ξαναεπιστρέφει στην αρχή (γνωστό και ως "looping") Παρατίθεται ο κώδικας:

```
void   UpdateTexture(CvCapture* input_video, IplImage  *image, bool mask)
       //Updates the video texture
{
       if(cvGetCaptureProperty(input_video,  CV_CAP_PROP_POS_FRAMES)>NumFrames-
3)//loop the video
              cvSetCaptureProperty(input_video,CV_CAP_PROP_POS_AVI_RATIO,0);//then
restart it

       //grab the frame from the video
       image = cvQueryFrame(input_video);

       // Convert to RGB
       cvCvtColor(image, image, CV_BGR2RGB);

       //we check for masking
       if(mask)
              createMask(image->imageData);

       // Create Texture
       gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGB, image->width, image->height,
GL_RGB, GL_UNSIGNED_BYTE, image->imageData);

       // Update View port
       glutPostRedisplay();
}
```

Στην αρχή του κώδικα γίνεται έλεγχός για το αν το βίντεο έχει φτάσει στο τέλος του. Αυτο πραγματοποιείται με την παραμετρο CV_CAP_PROP_POS_FRAMES μέσω της συνάρτησης *cvGetCaptureProperty().* Με αυτόν τον τρόπο ελέγχουμε αν βρισκόμαστε στα τελευταία καρέ του βίντεο. Αν ναι, τοτε μεταφέρουμε την αναπαραγωγή στην αρχή με τη συνάρτηση *cvSetCaptureProperty(),* με όρισμα CV_CAP_PROP_POS_AVI_RATIO και τιμή 0. Ολες οι παραπάνω εντολές συνεισφέρουν στο να εκτελείται συνεχώς και επαναλαμβανόμενα το βίντεο.

Το "AVI RATIO" (λόγος AVI) είναι ενα μεγεθος το οποίο καθορίζει που βρίσκεται η αναπαραγωγή. Ειναι κανονικοποιημένο, δηλαδή αν έχει την τιμή 0 σημαίνει οτι το βίντεο βρίσκεται στην αρχή, αν εχεί τιμή 0.5 σημαίνει οτι βρισκεται στο 50% επι της διάρκειας του βιντεο, δηλαδή στη μέση της διάρκειας.

Αυτη τη στιγμή πρεπει να εχει γίνει εμφανές οτι όταν θέλουμε να εξάγουμε καποιες πληροφορίες απο το βιντεο χρησιμοποιούμε τη συνάρτηση *cvGetCaptureProperty()* με όρισμα την κατάλληλη παράμετρο, ενω εαν θέλουμε να αλλάξουμε καποια παραμετρο της αναπαραγωγής τότε χρησιμοποιούμε τη *cvSetCaptureProperty().*

Επειτα, η συναρτηση ανακτά το τρέχον καρέ μεσω της συνάρτησης *cvQueryFrame(),* το οποίο αποθηκέυεται στην μεταβλητή 'image'. Επειδή το OpenCV επιστρέφει τα καρέ σε χρωματικό σχήμα "BGR" (blue green red), είναι απαραίτητο αυτό να μετατραπεί σε "RGB" ωστε να το χειριστεί σωστά το OpenGL. Αυτό πραγματοποιείται με τη συνάρτηση *cvCvtColor()* με όρισμα CV_BGR2RGB.

Στην τελική φάση, πρεπει να δημιουργηθεί η υφή, σε μορφή συμβατή με το OpenGL, κατι που γίνεται με τη συνάρτηση *gluBuild2DMaps(),* η οποία δημιουργεί ενα mipmap[3] απο το καρέ που εξήχθη.

Τέλος, να αναφέρουμε οτι η συνάρτηση *UpdateTexture()* δεν βρίσκεται αυτούσια μέσα στη main loop, αλλα στη συνάρτηση *draw().*

(Η συνάρτηση createMask() θα αναλυθεί στο κεφάλαιο blending).

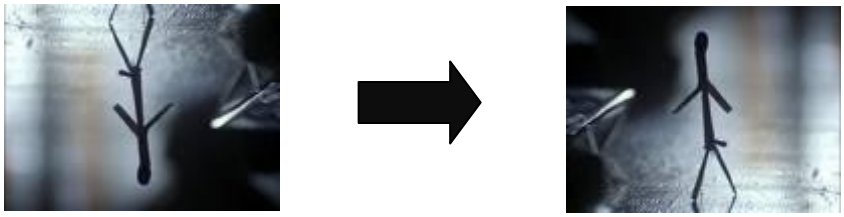
4.3.3) Σχεδίαση της εικονικής σκηνής

Το τελευταίο βήμα της εφαρμογής μας είναι η σχεδίαση της εικόνικής σκηνής, που στην περίπτωση μας είναι μια τετράγωνη επιφάνεια που έχει ως υφή ένα βίντεο. Η συνάρτηση που το υλοποιεί είναι η *draw().* Παρακάτω παρουσιάζεται το τμήμα κώδικα που χρήζει επεξήγησης:

```
glMatrixMode(GL_TEXTURE);//flip texture from bottop-up to top-down
glScalef(1.0, -1.0, 1.0);

   /* load the camera transformation matrix */
argConvGlpara(patt_trans, gl_para);
glMatrixMode(GL_MODELVIEW);
glLoadMatrixd( gl_para );


   if (patt_trans[2][1]>=0)
   {
          UpdateTexture(input_video2,image2,true);
          glBlendFunc(GL_DST_COLOR,GL_ZERO);

          glBegin(GL_QUADS);
   glTexCoord2f(0.0f, 0.0f);glVertex3f(-160.0f, -40.0f, 0.0f);
          glTexCoord2f(1.0f, 0.0f);glVertex3f( 80.0f, -40.0f, 0.0f);
          glTexCoord2f(1.0f, 1.0f);glVertex3f( 80.0f, -40.0f, 120.0f);
          glTexCoord2f(0.0f, 1.0f);glVertex3f(-80.0f, -40.0f, 120.0f);
          glEnd();
```

```
UpdateTexture(input_video,image,false);
        glBlendFunc(GL_ONE,GL_ONE);

        glBegin(GL_QUADS);
        glTexCoord2f(0.0f, 0.0f);glVertex3f(-80.0f, -40.0f, 0.0f);
        glTexCoord2f(1.0f, 0.0f);glVertex3f( 80.0f, -40.0f, 0.0f);
        glTexCoord2f(1.0f, 1.0f);glVertex3f( 80.0f, -40.0f, 120.0f);
        glTexCoord2f(0.0f, 1.0f);glVertex3f(-80.0f, -40.0f, 120.0f);
        glEnd();
}
```

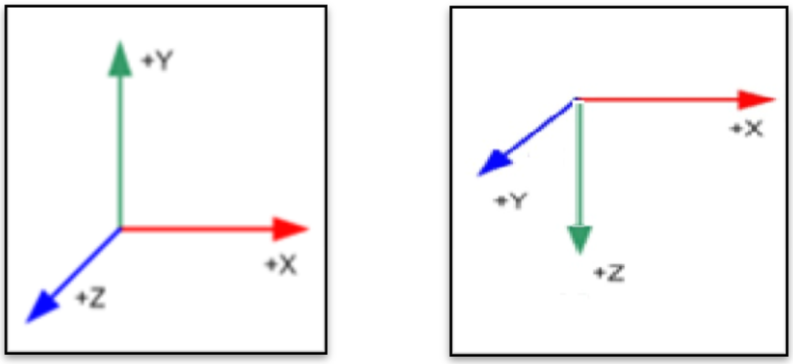
Για αρχή να αναφέρουμε οτι οι πίνακες μετασχηματισμού OpenGL καλούνται με τη συνάρτηση *glMatrixMode()*. Οι πρώτες δύο εντολές έχουν ως σκοπό την αναστροφή της υφής που προκύπτει απο το OpenCV, ωστε να μπορέσει το OpenGL να το χειριστεί σωστά. Η πρώτη εντολή καλεί τον πίνακα μετασχηματισμού GL_TEXTURE, ενω η δεύτερη (*glScaleφ()*) αναστρέφει τη μεταβλητή Y. Παρακάτω παρουσιάζεται ενα παράδειγμα. (εικόνα 4.10)



Εικόνα 4.10: Η υφή προκύπτει ανεστραμμένη απο το openCV και απαιτεί μετασχηματισμό

Οι επόμενες εντολές ειναι αρκετά σημαντικές. Η συνάρτηση *argConvGLpara*() μετατρέπει τον πίνακα μετασχηματισμού του ARToolKit στη μορφή που τον χειρίζεται η OpenGL. Να αναφέρουμε οτι η OpenGL αποθηκευει τους 4X4 πίνακες μετασχηματισμού ως πίνακες 1X16, δηλαδή όλα τα στοιχεία αποθηκεύονται σε μία γραμμή. Επειτα, ο πίνακας αυτός φορτώνεται στον πίνακα μετασχηματισμού GL_MODELVIEW (βλ. Ενότητα 3.2.7).

Απο εδώ και στο εξής, εφόσον έχει εφαρμοστεί ο μετασχηματισμός *ModelView*, οσον αφορά στις συντεταγμένες και τη σειρά δήλωσης τους, οσο και διανύσματα του δείκτη, ισχύει οτι ισχύει στο ARToolKit

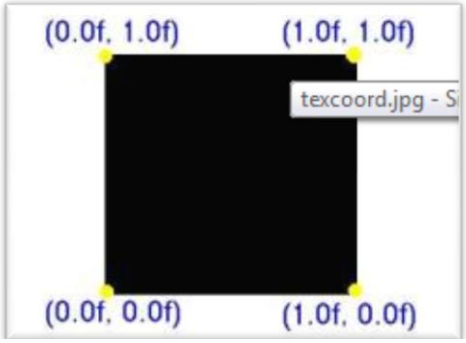


Εικόνα 4.11: τα μοναδιαία διανύσματα στην OpenGL (αριστερά) και τα μοναδιαία διανύσματα του δείκτη στο ARToolKit (δεξια)

Επομένως τα ορίσματα θα ακολοθούν πλέον τη σειρά του ARToolkit.

Επειτα, φορτώνεται το καρέ στην υφη με τη συνάρτηση *UpdateTexture().* (στην πρώτη επιφάνεια φορτώνεται η «μασκα», ορος που θα εξηγηθεί στην επόμενη ενότητα, ενώ στη δεύτερη η κανονική επιφάνεια.)

Στη συνέχεια, σχεδιάζονται δυό πανομοιότυπα τετράγωνα (το πρώτο αποσκοπεί, όπως θα δουμε στην επόμενη ενότητα στην εφαρμογή της λειτουργίας blending και το δευτερο για την απεικόνιση του κανονικού βίντεο), που όμως φαίνονται σαν ένα. Ο ορισμός μιας επιφάνειας ξεκινά με την εντολή *glbegin()* και λήγει με το *glEnd().* Η πρώτη έχει ως όρισμα μια παράμετρο που υποδηλώνει το τί σχημα θα σχεδιαστεί (πιθανές επιλογές ειναι απλά σημεία, τρίγωνα, τετράγωνα, πολύγωνα, κλπ), και που στην περίπτωση μας είναι τετράγωνο. Επειτα πρέπει να καθοριστούν τα σημεία και το πώς θα εφαρμοστεί η υφή επάνω στην επιφάνεια αυτή. Ακολουθεί σχήμα:

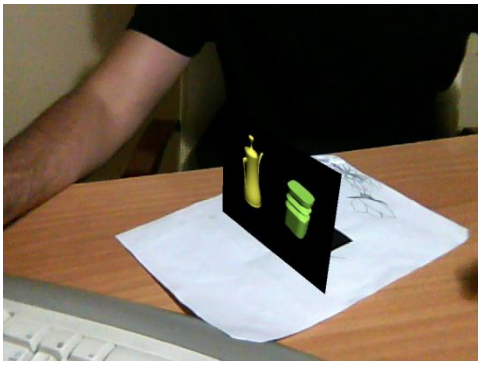


Εικόνα 4.12: οι συντεταγμένες της υφής

Εικόνα 4.13: οι συντεταγμένες της επιφάνειας στον χώρο

Παραπάνω βλέπουμε τις συντεταγμένες της υφής καθώς και της επιφάνειας μας στο χώρο. Παρατηρούμε οτι οι άκρες τις υφής εκφράζονται με δύο συντεταγμένες, καθως πρόκειται για δισδιάστατο αντικείμενο. Για να τοποθετηθεί η υφή σωστά πάνω στην επιφάνεια πρέπει να «ταιρίαξουμε» τα σημεία αμφότερων. Οπότε, ξεκινάμε απο την κάτω αριστερά ακρη , δηλώνοντας πρώτα το σημείο της υφής (με την εντολή *glTextCoord2f(0.0f, 0.0f))* και κατόπιν το σημείο της επιφάνειας (με την εντολή *glVertex3f(-80.0f, -40.0f, -0.0f)*). Αυτό συνεχίζεται και για τα τέσσερα σημεία, ακολουθώντας αντιωρολογιακή φορά.

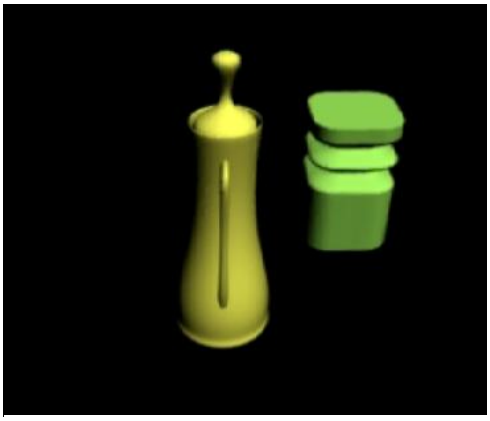
Τέλος να αναφέρουμε παρατηρούμε οτι οι εντολές για τη σχεδίαση αμφότερων των επιφανειών, βρίσκονται μεσα σε μια συνθήκη. Αυτη η συνθήκη αποσκοπεί στο να μη σχεδιάζεται η επιφάνεια οταν κανονικά, αν σχεδιαζοταν, θα βλέπαμε την πίσω όψη. (ή οταν βγαίνει απο το ημι-επίπεδο οπου γίνεται η σάρωση). Αυτή η επιλογή είναι καθαρά σχεδιαστική



Εικόνες 4.14 & 4.15: όταν ο δείκτης «κοιτά» μπροστά, η επιφάνεια σχεδιάζεται κανονικά, ενώ αν «κοιτα» προς τα πίσω, τοτε αυτο δε συμβαίνει. Ουσιαστικά, δε σχεδιάζουμε την πίσω όψη της επιφάνειας

4.3.4) Η λειτουργία “blending”

Η λειτουργία αυτή έχει ως σκοπό να μετατρέψει καποια τμήματα της επιφάνειας που επιθυμούμε σε διαφανή. Για να πραγματοποιηθεί αυτό απαιτείται πέρα απο την κανονική υφή, και μια «μασκα», που ουσιαστικά είναι μια ασπρόμαυρη εκδοχή της κανονικής. Τα σημεία που θέλουμε να είναι διαφανή έχουν λευκό χρώμα στη μάσκα, ενω τα αδιαφανή μαυρό. Παρατίθεται ένα παράδειγμα:

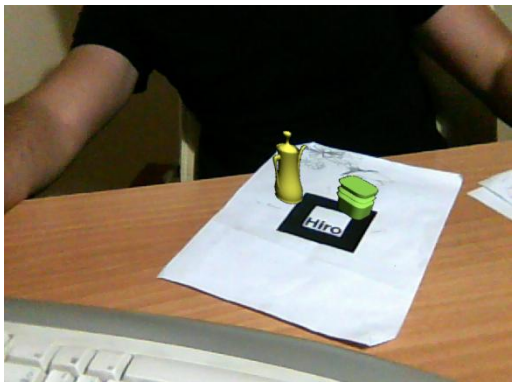


Εικόνα 4.16: Η κανονική υφή

Εικόνα 4.17: η μάσκα

Στην περίπτωση μας, λόγω της απλότητας του βίντεο που χρησιμοποιήσαμε (θέλουμε διαφάνεια στο φόντο, που τυχαίνει να είναι μαύρο) η μάσκα προέκυψε απο επεξεργασία του αρχικού βίντεο (με τη χρήση της συνάρτησης *createMask()*). Γενικώς, ο χρήστης πρέπει να παρέχει το βιντεο μάσκα. (ο κώδικας μας υποστηρίζει κατι τέτοιο, με μικρές τροποποιήσεις).

Για να επιτύχουμε τη λειτουργία "blending", πρέπει να εχουμε δυο πανομοιότυπες επιφάνειες (που ο χρηστης τις εκλαμβάνει ως μια), καθως κάθε επιφάνεια δέχεται μόνο μια υφη. Επειτα, στη πρώτη υφή εφαρμόζουμε τη μάσκα, και στη δεύτερη τη κανονική υφή (πραγματοποιείται με τη συνάρτηση *UpdateTexture()*). Ο συνδυασμός γίνεται με τη βοήθεια της συνάρτησης της OpenGL, *glBlendFunc(),* με τα κατάλληλα ορίσματα. Το τελικό αποτέλεσμα είναι το παρακάτω:



Εικόνα 4.18: η λειτουργία blending στην πράξη

Παρουσιάζεται η συνάρτηση που δημιουργεί τη μάσκα απο το αρχικό βίντεο:

4.3.4.1) η συνάρτηση *createMask()*

Η συνάρτηση αυτή ελέγχει το τρέχον καρέ σε επίπεδο πίξελ, και μετατρέπει τα μαύρα πίξελ σε λευκά, ενώ τα υπόλοιπα σε μαύρα, δημιουργώντας με αυτόν τον τρόπο τη μάσκα. Η συνάρτηση αυτή αποτελείται απο εντολές συμβολικής γλώσσας (assembly):

```
void createMask(void* buffer)
{
	void* b = buffer;
	__asm
	{
		mov ecx, 320*240
	mov ebx, b
		label:
			mov ax,0
			add ax,[ebx+0]
			add ax,[ebx+1]
			add ax,[ebx+2]
			cmp ax,0
			jne label2


			mov [ebx+0],255
			mov [ebx+1],255
			mov [ebx+2],255
			jmp label3
		label2:
			mov [ebx+0],0
			mov [ebx+1],0
			mov [ebx+2],0
		label3:
			add ebx,3
			dec ecx
			jnz label
	}
}
```

Η μεταβλητή *b* είναι ενας δείκτης προς το buffer της τρέχουσας εικόνας, οποτε η μεταβλητή *ebx* περιέχει τα περιεχόμενα του τρέχοντος πίξελ. Οι πληροφορίες των πίξελ, δηλαδή τα χρώματα, αποθηκεύονται στις 3 επόμενες θέσεις μνήμης. Έτσι στη θέση [ebx+0] περιέχεται η τιμή του κόκκινου χρώματος του τρέχοντος πίξελ, ενώ αντίστοιχα, στη θέση [ebx+1] του πρασίνου και στη θέση [ebx+2] του μπλε (εφόσον τα καρέ ακολουθούν το χρωματικό σχήμα RGB).

Οι τιμές και των τριών αυτών θέσεων αποθηκέυονται διαδοχικα στη μεταβλητή *ax*. Οποτε, αν το πίξελ είναι μαύρο, τότε το άθροισμα θα ισούται με μηδέν, διαφορετικά θα είναι διάφορο του μηδενός. Αν είναι μηδέν τοτε και στις τρεις θέσεις περνά η τιμή 255, που αντιστοιχεί στο λευκό χρώμα, ενω διαφορετικά περνούν οι τιμές 0. Ο έλεγχος γίνεται σε όλα τα πίξελ και με αυτόν τον τρόπο προκύπτει η μάσκα

Η μέθοδος αυτή έχει κάποια μειονεκτήματα (για παραδειγμα, οι σκιές,όντας μαύρες θεωρούνται διαφανείς) και για αυτόν το λόγο θα ήταν καλύτερα η μάσκα να δίνεται απο το χρήστη. Στην περίπτωση μας όμως λειτουργεί επαρκώς.

4.3.4.2) μια αναφορά στη συνάρτηση *glBlendFunc()*

Η συνάρτηση αυτή αποτελεί μέρος της βιβλιοθήκης της OpenGL, και είναι απαραίτητη για τη λειτουργία "blending". Δεχεται δυο ορίσματα, εκ των οποιων το πρώτο είναι ο παράγοντας με τον οποίο πολλαπλασιάζεται η υφή που εισέρχεται στο frame buffer[4] (source), ενώ το δεύτερο ο παράγοντας με τον οποίον πολλαπλασιάζεται η υφή που υπάρχει ήδη σε αυτό (destination). Επειτα, ό,τι προκύπτει προστίθεται. Οι πράξεις γίνονται μεταξύ των παραγόντων R(ed), G(reen), B(lue) και A(lpha). Ο μαθηματικός τύπος είναι ο εξής:

$$Τελικό\ χρώμα =$$

$$χρώμα_{πηγής} * παράγοντας_{πηγής} + χρώμα_{προορισμού} * παράγοντας_{προορισμού}$$

Οπότε, αν ο χρήστης καλέσει τη συνάρτηση ως εξής: glBlendFunc(GL_ONE, GL_ONE) (η παράμετρος GL_ONE αντιστοιχεί στη μονάδα) και ισχύουν και τα παρακάτω

$$χρώμα_{πηγής} = R: 0.2\ \ G: 0.4\ \ B: 0.1$$

$$χρώμα_{προορισμού} = R: 0.1\ \ G: 0.1\ B: 0.6$$

Τοτε θα προκύψει:

$$Τελικό\ χρώμα = R: 0.3\ \ G: 0.5\ B: 0.7$$

4.4 Εφαρμογή τρισδιάστατου ήχου

Ενα χαρακτηριστικό του προγράμματός μας είναι οτι μπορεί να αναπαράγει καποιον επιθυμητό ήχο, σαν αυτός να προέρχεται απο το βίντεό μας (που προβάλλεται πανω στον δείκτη). Αυτό γίνεται με τη βοήθεια του OpenAL, που μας δίνει τη δυνατότητα για πλήρη τρισδιάστατο ήχο. Ετσι ανάλογα με τη κίνηση του δείκτη, αντίστοιχα κινείται και ο ήχος στο χώρο. Ας δουμε παρακατω καποιες πληροφορίες για το OpenAL καθώς και για τις συναρτήσεις που χρησιμοποιήθηκαν στον κώδικα.

4.4.1) Το OpenAL API[(1)]

Το **OpenAL (Open A**udio **L**ibrary) αποτελεί ενα διαπλατφορμικό API[(1)] ήχου ελεύθερης χρήσης. Έχει σχεδιαστεί με σκοπό τον αποτελεσματικό σχεδιασμό πολυκαναλικού τρισδιάστατου ήχου θέσεως. Το στυλ του σκόπιμα θυμίζει αυτό του OpenGL, όσον αφορά την σύνταξη και τις παραδοχές

4.4.2) Δομή και λειτουργικότητα του API(1)

Η γενική λειτουργικότητα του OpenAL είναι κωδικοποιημένη σε *αντικείμενα-πηγές (source objects), buffer ηχου (audio buffers)* και σε ένα απλό *ακροατή (listener).* Ένα αντικείμενο πηγή περιέχει ενα δείκτη σε ένα buffer, την ταχύτητα, τη θέση και την κατεύθυνση του ήχου καθώς και την έντασή του. Ο ακροατής εμπεριέχει την ταχύτητα, τη θέση και την κατεύθυνση του ακροατή, και το γενικό κέρδος (gain) που εφαρμόζεται σε όλους τους ήχους. Τα buffers περιέχουν δεδομένα ήχου σε φορμά PCM, είτε 8-bit είτε 16-bit, σε μονοφωνική ή στερεοφωνική μορφή. Η μηχανή σχεδίασης πραγματοποιεί όλους τους απαραίτητους υπολογισμούς όπως η εξασθένηση λόγω απόστασης (distance attenuation), το φαινόμενο Doppler, κλπ.

Το τελικό αποτέλεσμα όλων των παραπάνω για τον τελικό χρήστη, σε μια ορθώς γραμμένη εφαρμογή OpenAL, είναι οτι οι ήχοι συμπεριφέρονται φυσικά καθώς ο χρήστης μετακινείται μέσα στον τρισδιάστατο χώρο του εικονικού κόσμου. Απο τη σκοπιά του προγραμματιστή, χρειάζεται λίγη προσπάθεια για να εφαρμοστεί κάτι τέτοιο σε μια ήδη υπάρχουσα εφαρμογη τρισδιάστατων γραφικών βασισμένη σε OpenGL

4.4.3) Μοντέλα απόστασης

Ιδιαίτερη μνεία πρέπει να γίνει στα διαφορα μοντέλα απόστασης (distance models) που μπορούν να εφαρμοστουν στην πηγή μας, και τα οποία καθορίζουν το πώς θα γίνεται η εξασθένιση του ήχου σε σχέση με την απόσταση απο τον ακροατή.

Τα σημαντικά μεγέθη όταν ορίζεται ένα μοντέλο είναι η *απόσταση αναφοράς* που αντιστοιχεί στην παράμετρο "AL_REFERENCE_DISTANCE" , η *μέγιστη απόσταση* που αντιστοιχεί στο "AL_MAX_DISTANCE" και ο *παράγοντας roll-off,* που αντιστοιχεί στην παράμετρο "AL_ROLLOFF_FACTOR".

Η *απόσταση αναφοράς* ορίζεται ως η απόσταση οπου το κέρδος του ήχου μειώνεται στο μισό. Στην εφαρμογή μας αυτή έχει την τιμή 300 (τιμή κατάλληλη για τους σκοπούς του προγράμματος).

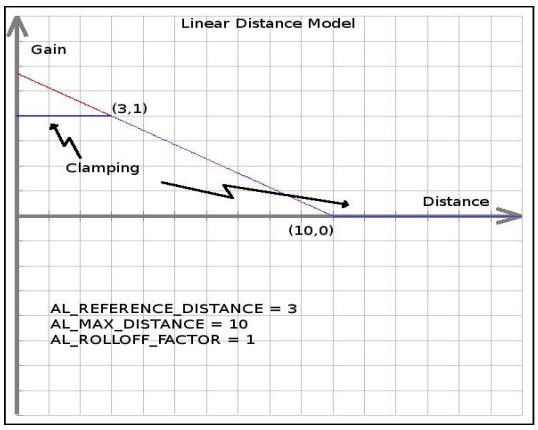
Η *μέγιστη απόσταση* καθορίζει την απόσταση στην οποία το κέρδος γίνεται ελάχιστο, και πέρα απο αυτήν, δεν υπάρχει περαιτερω μείωσή του.

Ο παράγοντας *roll-off* καθορίζει το πόσο απότομη θα είναι η κλιση της καμπύλης ή ισοδύναμα το πόσο απότομα θα μειώνεται το κέρδος του ήχου με την απόσταση. Μεγαλύτερη τιμη της παραμέτρου αντιστοιχεί σε πιο απότομη κλίση. Οπως φαίνεται και στον κώδικα η τιμή της παραμέτρου έχει οριστεί στην τιμή 2.

Πριν παρουσιάσουμε τα τρία βασικά μοντέλα (γραμμικό, εκθετικό, αντίστροφο με clamping, που υποδηλώνει οτι αν η αποσταση ξεπερασει την απόσταση αναφοράς, το κέρδος μειώνεται δραματικά) να αναφέρουμε οτι η εφαρμογή μας χρησιμοποιεί το αντίστροφο μοντέλο:

4.4.3.1) Γραμμικό μοντέλο απόστασης (με clamping)

Στο γραμμικό μοντέλο απόστασης η μείωση του κέρδους γίνεται γραμμικά σε συνάρτηση με την απόσταση μεταξύ του παρατηρητή και της πηγής



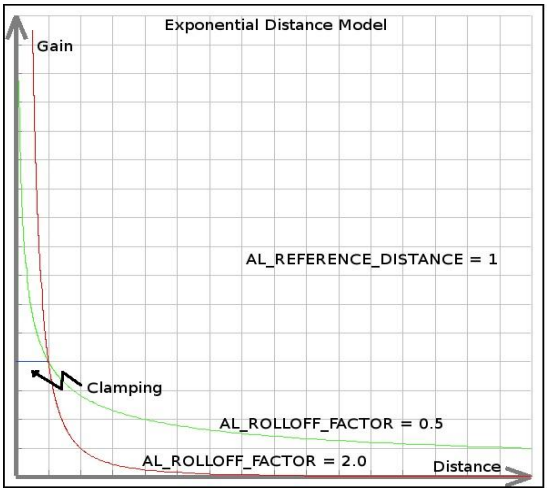
Εικόνα 4.19: Γραμμικό μοντέλο απόστασης

Οι βασικοί τύποι είναι:

- Απόσταση: min (απόσταση, AL_MAX_DISTANCE)
- Κέρδος: (1-AL_ROLLOFF_FACTOR*(Απόσταση-AL_REFERENCE_DISTANCE)/(AL_MAX_DISTANCE-AL_REFERENCE_DISTANCE)

4.4.3.2) Εκθετικό μοντέλο απόστασης (με clamping)

Εδώ, η πτώση του κέρδους γίνεται εκθετικά σε σχέση με την απόσταση μεταξύ παρατηρητή και πηγής



Εικόνα 4.20: Εκθετικό μοντέλο απόστασης

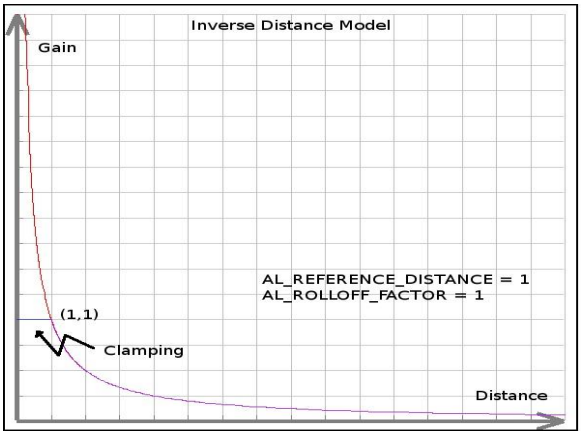
Οι τύποι σε αυτήν την περίπτωση ορίζονται ως εξής:

- Κέρδος: (Απόσταση/AL_REFERENCE_DISTANCE)^(-AL_ROLLOFF_FACTOR)

4.4.3.3) Αντίστροφο μοντέλο απόστασης (με clamping)

Στο μοντέλο αυτό ισχύουν οι εξής τύποι:

- Απόσταση: max (απόσταση, AL_REFERENCE_DISTANCE)
- Απόσταση: min (απόσταση, AL_MAX_DISTANCE)
- Κέρδος: AL_REFERENCE_DISTANCE/(AL_REFERENCE_DISTANCE + AL_ROLLOFF_FACTOR * (Απόσταση-AL_REFERENCE_DISTANCE))



Εικόνα 4.21: μοντέλο αντίστροφης απόστασης

Το μοντέλο αυτό είναι εκείνο που χρησιμοποιεί η εφαρμογή μας.

4.4.4) Ανάλυση των συναρτήσεων ήχου

Παρακάτω θα αναλυθούν οι συναρτήσεις που χρησιμοποιήθηκαν για την εφαρμογή ήχου στο πρόγραμμα μας. Οπως έχει αναφερθεί και προηγουμένως, το API[(1)] που χρησιμοποιήθηκε για αυτόν το σκοπό είναι το OpenAL. Ας δουμε παρακάτω τις λεπτομέρειες.

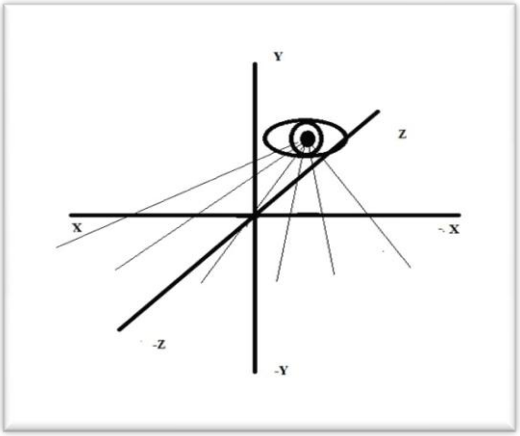
4..4.3.1) Αρχικοποίηση των μεταβλητών ήχου

Το OpenAL απαιτεί τη χρήση ειδικών μεταβλητών που εκφράζουν κάποια στοιχειώδη μεγέθη (θεση, ταχύτητα, προσανατολισμός πηγής/ακροατή) και η δήλωση τους μεσα στον κώδικα είναι απαραίτητη ώστε να εξασφαλιστεί η σωστή λειτουργία του τρισδιάστατου ήχου.. Παρουσιάζεται ο αντίστοιχος κώδικας:

```
// Buffers hold sound data.
ALuint		Buffer;

// Sources are points emitting sound.
ALuint		Source;

// Position of the source sound.
ALfloat		SourcePos[3];

// Velocity of the source sound.
ALfloat		SourceVel[]={0.0,0.0,0.0};


// Position of the listener.
ALfloat		ListenerPos[] = { 0.0, 0.0, 0.0 };

// Velocity of the listener.
ALfloat		ListenerVel[] = { 0.0, 0.0, 0.0 };

// Orientation of the listener. (first 3 elements are "at", second 3 are "up")
ALfloat		ListenerOri[] = { 0.0, 0.0, -1.0,  0.0, 1.0, 0.0 };
```

Οπως είναι εμφανές, η θέση, η ταχύτητα και ο προσανατολισμός (μεέγεθος που είναι χρήσιμο μόνο για τον ακροατή) είναι μεγέθη που εκφράζονται σε διανύσματα τριών συνιστωσών (αφού μιλάμε για τρισδιάστατο χώρο). Παρατηρούμε οτι η ταχύτητα της πηγής ορίζεται σαν μηδενική (και παραμένει σταθερή, αν και οπως θα δούμε παρακάτω αυτό ουσιαστικά δε συμβαίνει) όπως και η του ακροατή. Επίσης, θεωρούμε οτι μόνο η πηγή κινείται, άρα ορίζουμε τη θέση και την ταχύτητα του ακροατή στο μηδέν.

Οσον αφορά τον προσανατολισμό, επειδή το OpenAL χρησιμοποιεί δεξίοστροφο σύστημα συντεταγμένων, , με τη δηλωση που κάνουμε (στις τρεις πρώτες παραμέτρους) δηλώνουμε οτι ο παρατηρητής κοιτά προς τον άξονα –Z, που ταυτίζεται με τη θέση της κάμερας, αν θέσουμε Z=-Z καθώς το ARToolkit χρησιμοποιεί αριστερόστροφο συστημα συντεταγμένων (οσον αφορά την κάμερα). (ο άξονας Y είναι κάθετος στο οριζόντιο επίπεδο XZ)



Εικόνα 4.22: οπτικό πεδίο της κάμερας (που ταυτίζεται με του παρατηρητή/ακροατή)

Τέλος, οι τελευταίες τρεις παράμετροι υποδηλώνουν τον προσανατολισμό του επιπέδου XZ (που προκύπτει απο το κάθετο σε αυτό διάνυσμα), το οποίο είναι το μοναδιαίο διάνυσμα που ταυτίζεται με τον άξονα .

4.4.4.2) Συνάρτηση *LoadALData()*

Ο κώδικας της συνάρτησης είναι ο παρακάτω:

```
ALboolean LoadALData(void)//creates an audio buffer and binds it to an audio source
{
        // Load wav data into a buffer
    alGenBuffers(1, &Buffer);
    if (alGetError() != AL_NO_ERROR)
        return AL_FALSE;

       Buffer=alutCreateBufferFromFile("Data/sound.wav");

       // Bind buffer with a source.
    alGenSources(1, &Source);

    if (alGetError() != AL_NO_ERROR)
        return AL_FALSE;
       //initialize the parameters
       //initialize the parameters
    alSourcei (Source, AL_BUFFER,   Buffer   );//bind the buffer to a source
    alSourcef (Source, AL_PITCH,    1.0f     );
    alSourcef (Source, AL_GAIN,     1.0f     );
    alSourcef (Source, AL_REFERENCE_DISTANCE,300.0f);//set the distance model
    alSourcef (Source, AL_ROLLOFF_FACTOR, 2.0f);
    alSourcefv(Source, AL_VELOCITY, SourceVel);
    alSourcei (Source, AL_LOOPING,  AL_TRUE);

        // Do another error check and return.
    if (alGetError() == AL_NO_ERROR)
        return AL_TRUE;

    return AL_FALSE;
}
```

Η συνάρτηση αυτή είναι η πιο σημαντική όσον αφορά την εφαρμογή του ήχου στην εφαρμογή μας. Αρχικά, δημιουργεί ένα buffer (με τη βοήθεια της συνάρτησης *alGenBuffers()*), και έπειτα το συνδέει με το επιθυμητό αρχείο ήχου (στην περίπτωση μας με το αρχείο “sound.wav” στον υποφάκελο “Data”) μέσω της συνάρτησης *alutCreateBufferFromFile().*

Τέλος, είναι απαραίτητη και η δημιουργία ενός αντικειμένου πηγής (μέσω της συνάρτησης *alGenSources()*) και της σύνδεσης του με το buffer (μέσω της *alSourcei* με δεύτερο όρισμα το εν λόγω buffer. Ολα τα παραπάνω έχουν ως αποτέλεσμα τη δημιουργία μιας πηγής, η οποία είναι συνδεδεμένη με το αρχείο ήχου μας.

Στις επόμενες γραμμές κώδικα, ορίζονται κάποιες παράμετροι της πηγής (η γενική μορφή είναι η εξής alSource (Πηγή, παράμετρος, τιμή)).

4.4.4.3) Συνάρτηση *SetSourceValues()*

Ο κώδικας της εν λόγω συνάρτησης ακολουθεί:

```
void SetSourceValues(void)
{
	//set the audio source position the same as the marker position
	SourcePos[0]=patt_trans[0][3];
	SourcePos[1]=patt_trans[1][3];
	SourcePos[2]=-patt_trans[2][3];

	//Update the audio source position
	alSourcefv(Source, AL_POSITION, SourcePos);
```

Η συνάρτηση αυτή ουσιαστικά «λέει» στο OpenAL τη θέση του δείκτη, σύμφωνα με τα δεδομένα που έχουν εξαχθεί απο το ARToolKit, . Ως γνωστόν, οι συντεταγμένες του δείκτη περίεχονται σε έναν πίνακα 3X4, οπου η τέταρτη στήλη του περιέχει τη θέση του δείκτη σε σχέση με τη θέση της κάμερας (ο προσανατολισμός του δείκτη, που περιέχεται στις υπόλοιπες θέσεις του πίνακα δε μας ενδιαφέρει). Ως αποτέλεσμα, η θέση της πηγής ήχου ταυτίζεται με αυτή του δείκτη. (να τονίσουμε οτι το «μειον» στην Τρίτη συντεταγμένη οφείλεται στο οτι το σύστημα συντεταγμένων της κάμερας ειναι αριστερόστροφο ενω αυτο που χρησιμοποιεί το OpenAL δεξιόστροφο

Να τονίσουμε οτι η συνάρτηση αυτή καλείται μέσα στη Main Loop του προγράμματος μας, καθώς είναι προφανές οτι απαιτούμε κατα την κίνηση του δείκτη να έχουμε και την αντίστοιχη μετατόπιση της πηγής ήχου.

4.4.4.4) Συνάρτηση SetListenerValues()

Αντίστοιχα ορίζονται και οι τιμές για τον ακροατή:

```
void SetListenerValues()
{
    alListenerfv(AL_POSITION,    ListenerPos);
    alListenerfv(AL_VELOCITY,    ListenerVel);
    alListenerfv(AL_ORIENTATION, ListenerOri);
}
```

Οι τιμές του ακροατή είναι σταθερές, αφου ουσιαστικά ο ακροατής είναι η κάμερα, και ως σύμβαση θεωρούμε οτι το μόνο που κινείται είναι η πηγή (δηλαδή ο δείκτης). Αρα, δεν υπάρχει ανάγκη να υπάρχει η συνάρτηση αυτή στη Main Loop του προγράμματος, παρα μόνο στη φάση της αρχικοποίησης (initialization).

4.4.5) Άλλες χρήσιμες πληροφορίες σχετικά με τον ήχο και το OpenAL

Να αναφέρουμε σε αυτό το σημείο, οτι το OpenAL, για τη διευκόλυνση του προγραμματιστή, χρησιμοποιεί συμβολικές παραμέτρους, οι οποίες αντιστοιχούν σε κάποιες ακέραιες τιμές. Αυτό συμβαίνει επειδή η απομνημόνευση αρκετών μεγεθών και παραμέτρων ειναι πιο εύκολη όταν γίνεται ονομαστικά παρα με κωδικούς. Παράμετροι οπως οι “AL_LOOPING” ή “AL_REFERENCE_MODEL” δεν είναι τίποτα άλλο παρα κάποιες ακέραιες τιμές. Είναι πολύ πιο έυκολο ομως για τον χρηστη να τοποθετήσει σαν όρισμα για παράδειγμα το “AL_LOOPING” στην κατάλληλη συναρτηση αν θελει να ενεργοποιήσει ή να απενεργοποιήσει την επαναληψη της αναπαραγωγής, παρα να θυμάται τον αριθμο που αντιστοιχεί στη λειτουργία αυτή.

4.5 Οδηγίες χρήσης της εφαρμογής

Παρακάτω δίνονται κάποιες οδηγίες για την εκτέλεση της εφαρμογής. Πρώτα απ’ όλα ο χρήστης πρέπει να εγκαταστήσει τα απαραίτητα API και βιβλιοθήκες ωστε να καταστεί δυνατή η εκτέλεση της εφαρμογής. Αυτά είναι:

- Microsoft Visual C++ 2010 Redistributable package (http://www.microsoft.com/download/en/details.aspx?id=5555)
- OpenCV 2.3 API (http://sourceforge.net/projects/opencvlibrary/files/opencv-win/2.3/)
- OpenAL (http://connect.creativelabs.com/openal/Downloads/Forms/AllItems.aspx)

Επειτα, δίνεται η επιλογη στον χρήστη να τοποθετήσει τα βιντεο της επιλογής του, το οποία πρέπει να πληρούν τις παρακάτω προδιαγραφές:

- Τα αρχεία βίντεο πρέπει να είναι τύπου .AVI
- Πρέπει να έχουν για όνομα κάποιον ακέραιο αριθμό
- Πρέπει να τοποθετηθούν στον υποφάκελο «Data» της εφαρμογής
- Οι ονομασίες τους πρέπει να είναι τέτοιες ωστε αν ταξινομηθούν βάσει ονομασίας, η προοπτική να κινείται αριστερόστροφα

Συν τοις άλλοις, δίνεται η επιλογή στον χρήστη να τοποθετήσει ένα αρχείο ήχου ωστε να γίνει η αναπαραγωγή του απο την εφαρμογή. Αυτό πρέπει να πληροί τις εξής προδιαγραφές:

- Πρέπει να είναι τύπου .wav
- Να έχει ως όνομα “sound.wav”
- Να τοποθετηθεί στον υποφάκελο “Data” τηε εφαρμογής

Κατα την εκτέλεση ο χρήστης μπορεί να χρησιμοποιήσει κάποιες λειτουργίες πατώντας τα κατάλληλα πλήκτρα. Αυτές είναι:

- Ενεργοποίηση/απενεργοποίηση της λειτουργίας “blending” με το πάτημα του πλήκτρου “b” (προεπιλεγμένα απενεργοποιημένη)
- Ενεργοποίηση/απενεργοποίηση της αναπαραγωγής ήχου με το πάτημα του πλήκτρου “p” (προεπιλεγμένα απενεργοποιημένη)
- Έξοδος απο την εφαρμογή με το πλήκτρο “ESC”

Παράρτημα 1 (Ορολογίες)

(1) API: Διαπροσωπία προγραμματισμού εφαρμογών (application programming interface). Αποτελεί ένα συγκεκριμένο σύνολο κανόνων και προδιαγραφών που χρησιμοποιούν τα προγράμματα για να επικοινωνούν μεταξύ τους. Εξυπηρετεί σα διαπροσωπία μεταξύ διαφορετικών προγραμμάτων και διευκολύνει την αλληλεπίδρασή τους, με τον ίδιο τρόπο που η διαπροσωπία χρήστη (user interface) διευκολύνει την αλληλεπίδραση του ανθρώπου με τον υπολογιστή.

(2) Κανονικό διάνυσμα (normal vector): Το κανονικό διάνυσμα μιας επιφάνειας είναι το διάνυσμα εκείνο που είναι κάθετο στην επιφάνεια αυτήν και συνήθως είναι μοναδιαίου μέτρου.

(3) Mipmap: Πρόκειται για μία συλλογή προ-υπολογισμένων, βελτιστοποιημένων εικόνων που συνοδεύουν τη βασική υφή, έχοντας ως σκοπό την αύξηση της ταχύτητας σχεδίασης και τον περιορισμό φαινομένων ‘aliasing’. Καθε εικόνα του mipmap είναι μια εκδοχή της βασικής υφής, αλλα με μειωμένο επίπεδο λεπτομέρειας. Η βασική υφή χρησιμοποιείται όταν η προοπτική ειναι η κατάλληλη, όταν ομως η αποσταση είναι μεγάλη ή σχεδιάζεται σε μικρό μέγεθος, χρησιμοποιείται η κατάλληλη εικόνα της συλλογής mipmap

Εικόνα: παράδειγμα mipmap

(4) Frame Buffer: Πρόκειται για ένα buffer, οπου αποθηκέυονται οι τιμές των χρωμάτων κάθε πίξελ στην οθόνη. Συνήθως , δημιουργείται στη μνήμη της κάρτας γραφικών (εκτός εαν το σύστημα έχει καρτα γραφικών ενσωματωμένη στη μητρική πλακέτα, οπου τότε αποθηκεύεται στη βασική μνήμη τους συστήματος)

Παράρτημα 2 (κώδικας simple.exe)

```
#ifdef _WIN32
#include <windows.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#ifndef __APPLE__
#include <GL/gl.h>
#include <GL/glut.h>
#else
#include <OpenGL/gl.h>
#include <GLUT/glut.h>
#endif
#include <AR/gsub.h>
#include <AR/video.h>
#include <AR/param.h>
#include <AR/ar.h>

/* set up the video format globals */

#ifdef _WIN32
char                 *vconf = "Data\\WDM_camera_flipV.xml";
#else
char                 *vconf = "";
#endif

int             xsize, ysize;
int             thresh = 100;
int             count = 0;

int             mode = 1;

char           *cparam_name    = "Data/camera_para.dat";
ARParam         cparam;

char           *patt_name      = "Data/patt.hiro";
int             patt_id;
int             patt_width     = 80.0;
double          patt_center[2] = {0.0, 0.0};
double          patt_trans[3][4];

static void   init(void);
static void   cleanup(void);
static void   keyEvent( unsigned char key, int x, int y);
static void   mainLoop(void);
static void   draw( double trans[3][4] );


int main(int argc, char **argv)
{
       glutInit(&argc, argv);
    init();

    arVideoCapStart();
    argMainLoop( NULL, keyEvent, mainLoop );
       return (0);
}
```

```
static void   keyEvent( unsigned char key, int x, int y)
{
    /* quit if the ESC key is pressed */
    if( key == 0x1b ) {
        printf("*** %f (frame/sec)\n", (double)count/arUtilTimer());
        cleanup();
        exit(0);
    }

    if( key == 'c' ) {
        printf("*** %f (frame/sec)\n", (double)count/arUtilTimer());
        count = 0;

        mode = 1 - mode;
        if( mode ) printf("Continuous mode: Using arGetTransMatCont.\n");
         else      printf("One shot mode: Using arGetTransMat.\n");
    }
}

/* main loop */
static void mainLoop(void)
{
    static int      contF = 0;
    ARUint8         *dataPtr;
    ARMarkerInfo    *marker_info;
    int             marker_num;
    int             j, k;

    /* grab a vide frame */
    if( (dataPtr = (ARUint8 *)arVideoGetImage()) == NULL ) {
        arUtilSleep(2);
        return;
    }
    if( count == 0 ) arUtilTimerReset();
    count++;

    argDrawMode2D();
    argDispImage( dataPtr, 0,0 );

    /* detect the markers in the video frame */
    if( arDetectMarker(dataPtr, thresh, &marker_info, &marker_num) < 0 ) {
        cleanup();
        exit(0);
    }

    arVideoCapNext();

    /* check for object visibility */
    k = -1;
    for( j = 0; j < marker_num; j++ ) {
        if( patt_id == marker_info[j].id ) {
            if( k == -1 ) k = j;
            else if( marker_info[k].cf < marker_info[j].cf ) k = j;
        }
    }
    if( k == -1 ) {
        contF = 0;
        argSwapBuffers();
        return;
    }
```

```
  /* get the transformation between the marker and the real camera */
    if( mode == 0 || contF == 0 ) {
        arGetTransMat(&marker_info[k], patt_center, patt_width, patt_trans);
    }
    else {
        arGetTransMatCont(&marker_info[k], patt_trans, patt_center, patt_width,
patt_trans);
    }
    contF = 1;

    draw( patt_trans );

    argSwapBuffers();
}

static void init( void )
{
    ARParam  wparam;

    /* open the video path */
    if( arVideoOpen( vconf ) < 0 ) exit(0);
    /* find the size of the window */
    if( arVideoInqSize(&xsize, &ysize) < 0 ) exit(0);
    printf("Image size (x,y) = (%d,%d)\n", xsize, ysize);

    /* set the initial camera parameters */
    if( arParamLoad(cparam_name, 1, &wparam) < 0 ) {
        printf("Camera parameter load error !!\n");
        exit(0);
    }
    arParamChangeSize( &wparam, xsize, ysize, &cparam );
    arInitCparam( &cparam );
    printf("*** Camera Parameter ***\n");
    arParamDisp( &cparam );

    if( (patt_id=arLoadPatt(patt_name)) < 0 ) {
        printf("pattern load error !!\n");
        exit(0);
    }

    /* open the graphics window */
    argInit( &cparam, 1.0, 0, 0, 0, 0 );
}

/* cleanup function called when program exits */
static void cleanup(void)
{
    arVideoCapStop();
    arVideoClose();
    argCleanup();
}


static void draw( double trans[3][4] )
{
    double    gl_para[16];
    GLfloat   mat_ambient[]     = {0.0, 0.0, 1.0, 1.0};
    GLfloat   mat_flash[]       = {0.0, 0.0, 1.0, 1.0};
    GLfloat   mat_flash_shiny[] = {50.0};
```

```
GLfloat   light_position[]  = {100.0,-200.0,200.0,0.0};
    GLfloat   ambi[]            = {0.1, 0.1, 0.1, 0.1};
    GLfloat   lightZeroColor[]  = {0.9, 0.9, 0.9, 0.1};

    argDrawMode3D();
    argDraw3dCamera( 0, 0 );
    glClearDepth( 1.0 );
    glClear(GL_DEPTH_BUFFER_BIT);
    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LEQUAL);

    /* load the camera transformation matrix */
    argConvGlpara(trans, gl_para);
    glMatrixMode(GL_MODELVIEW);
    glLoadMatrixd( gl_para );

    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    glLightfv(GL_LIGHT0, GL_AMBIENT, ambi);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, lightZeroColor);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_flash);
    glMaterialfv(GL_FRONT, GL_SHININESS, mat_flash_shiny);
    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
    glMatrixMode(GL_MODELVIEW);
    glTranslatef( 0.0, 0.0, 25.0 );
    glutSolidCube(50.0);
    glDisable( GL_LIGHTING );

    glDisable( GL_DEPTH_TEST );
}
```

Παράρτημα 3 (κώδικας της εφαρμογής)

```
#ifdef _WIN32
#include <windows.h>
#endif

#pragma comment(lib, "OpenGL32.lib")
#include <stdio.h>
#include <stdlib.h>
#ifndef __APPLE__
#include <GL/gl.h>
#include <GL/glut.h>

#else
#include <GL/gl.h>

#endif
#include <AR/gsub.h>
#include <AR/video.h>
#include <AR/param.h>
#include <AR/ar.h>
#include <opencv/highgui.h>
#include <opencv/cv.h>
#include <conio.h>
#include <al.h>
#include <alc.h>
#include <alut.h>
#include <math.h>




#ifndef CDS_FULLSCREEN            // CDS_FULLSCREEN Is Not Defined By Some
#define CDS_FULLSCREEN 4          // Compilers. By Defining It This Way,
#endif                            // We Can Avoid Errors



//
// Camera configuration.
//
#ifdef _WIN32
char          *vconf = "Data\\WDM_camera_flipV.xml";
#else
char          *vconf = "";
#endif

int             xsize, ysize;
int             thresh = 100;
int             count = 0;
int             mode = 1;

char           *cparam_name    = "Data/camera_para.dat";
ARParam         cparam;

char           *patt_name      = "Data/patt.hiro";
int             patt_id;
double          patt_width     = 80.0;
double          patt_center[2] = {0.0, 0.0};
```

```
double        patt_trans[3][4];//marker transformation matrix
double        marker_angle;// the angle of the marker
int           new_scene,old_scene;

CvCapture     *input_video, *input_video2;
IplImage      *image, *image2;
int           NumFrames;
bool          FirstTime=true;
int           counter=5;
float         delay,FPS;
int           videoNu;
WIN32_FIND_DATA FindFileData;
HANDLE        hFind;
int           videoNames[40];//array with the names of the video files
bool          audio=false, blend=false;

const double PI = 3.141592;

// Buffers hold sound data.
ALuint        Buffer;

// Sources are points emitting sound.
ALuint        Source;

// Position of the source sound.
ALfloat       SourcePos[3];

// Velocity of the source sound.
ALfloat       SourceVel[]={0.0,0.0,0.0};


// Position of the listener.
ALfloat       ListenerPos[] = { 0.0, 0.0, 0.0 };

// Velocity of the listener.
ALfloat       ListenerVel[] = { 0.0, 0.0, 0.0 };

// Orientation of the listener. (first 3 elements are "at", second 3 are "up")
ALfloat       ListenerOri[] = { 0.0, 0.0, -1.0,  0.0, 1.0, 0.0 };

static void          init(void);
static void          cleanup(void);
static void          keyEvent( unsigned char key, int x, int y);
static void          mainLoop(void);
static void          draw( void );
static void          UpdateTexture(CvCapture *input_video, IplImage  *image, int
mask);
static void          chooseAVI(void);
static void          angleToScene (void);
ALboolean            LoadALData(void);
void                 KillALData(void);
void                 SetSourceValues(void);
void                 SetListenerValues(void);
int                  CountVideos(void);
void                 getVideoProperties(CvCapture *input_video);
int                  compare (const void * a, const void * b);
void                 createMask(void* buffer);
void                 showError(void);
```

```
int main(int argc, char **argv)
{

       videoNu=CountVideos();

       if(videoNu==-1) //if there are no videos, exit
       {
              showError();
              return(-1);
       }

       init();
       alutInit(&argc, argv);
       alGetError();
       SetListenerValues();
       LoadALData();
       arVideoCapStart();
       atexit(KillALData);

       printf("%d video(s) have been found....\n",videoNu);

    argMainLoop( NULL, keyEvent, mainLoop );

       return (0);
}

static void   keyEvent( unsigned char key, int x, int y)
{

    /* quit if the ESC key is pressed */
    if( key == 0x1b ) {
        printf("*** %f (frame/sec)\n", (double)count/arUtilTimer());
        cleanup();
              exit(0);
       }

       /*turn on/off audio if 'p' key is pressed*/
       if( key == 0x70 ) {
              if (audio)
                     alSourcePause(Source);
              else
                     alSourcePlay(Source);
              audio=!audio;
       }

       /*turn blending on/off if 'b' is pressed*/
       if( key == 0x62 )
              blend=!blend;
}

/* main loop */
static void mainLoop(void)
{
    ARUint8         *dataPtr;
    ARMarkerInfo    *marker_info;
    int             marker_num;
    int             j, k;
    static int      contF = 0;
```

```
/* grab a vide frame */
    if( (dataPtr = (ARUint8 *)arVideoGetImage()) == NULL ) {
        arUtilSleep(2);
        return;
    }
    if( count == 0 ) arUtilTimerReset();
    count++;

    argDrawMode2D();
    argDispImage( dataPtr, 0,0 );

    /* detect the markers in the video frame */
    if( arDetectMarker(dataPtr, thresh, &marker_info, &marker_num) < 0 ) {
        cleanup();
        exit(0);
    }

    arVideoCapNext();


    /* check for object visibility */
    k = -1;
    for( j = 0; j < marker_num; j++ ) {
        if( patt_id == marker_info[j].id ) {
            if( k == -1 ) k = j;
            else if( marker_info[k].cf < marker_info[j].cf ) k = j;
        }
    }
    if( k == -1 ) {
        argSwapBuffers();
        return;
    }
    /* get the transformation between the marker and the real camera */
    if( mode == 0 || contF == 0 ) {
        arGetTransMat(&marker_info[k], patt_center, patt_width, patt_trans);
    }
    else {
        arGetTransMatCont(&marker_info[k], patt_trans, patt_center, patt_width,
patt_trans);
    }
    contF = 1;

       angleToScene();      //choose a scene based on the angle


       if (FirstTime)//run only the first time
       {
              chooseAVI();
              getVideoProperties(input_video);
              FirstTime=!FirstTime;
       }

       //if the scene has changed, switch to the correspondent video
       if (new_scene!=old_scene)
              chooseAVI();
       SetSourceValues();
    draw();
       argSwapBuffers();
}
```

```
static void init( void )
{
    ARParam  wparam;

    /* open the video path */
    if( arVideoOpen( vconf ) < 0 ) exit(0);
    /* find the size of the window */
    if( arVideoInqSize(&xsize, &ysize) < 0 ) exit(0);
    printf("Image size (x,y) = (%d,%d)\n", xsize, ysize);

    /* set the initial camera parameters */
    if( arParamLoad(cparam_name, 1, &wparam) < 0 ) {
        printf("Camera parameter load error !!\n");
        exit(0);
    }
    arParamChangeSize( &wparam, xsize, ysize, &cparam );
    arInitCparam( &cparam );
    printf("*** Camera Parameter ***\n");
    arParamDisp( &cparam );

    if( (patt_id=arLoadPatt(patt_name)) < 0 ) {
        printf("pattern load error !!\n");
        exit(0);
    }

    /* open the graphics window */
    argInit(&cparam, 1.5, 0, 0, 0, 0 );
}

/* cleanup function called when program exits */
static void cleanup(void)
{
    cvReleaseCapture(&input_video);
    arVideoCapStop();
    arVideoClose();
    argCleanup();
}

 void  UpdateTexture(CvCapture* input_video, IplImage  *image, bool mask)
       //Updates the video texture
{
       if(cvGetCaptureProperty(input_video,  CV_CAP_PROP_POS_FRAMES)>NumFrames-
3)//loop the video
              cvSetCaptureProperty(input_video,CV_CAP_PROP_POS_AVI_RATIO,0);//then
restart it

       //grab the frame from the video
       image = cvQueryFrame(input_video);

       // Convert to RGB
       cvCvtColor(image, image, CV_BGR2RGB);

       //we check for masking
       if(mask)
              createMask(image->imageData);

       // Create Texture
       gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGB, image->width, image->height,
GL_RGB, GL_UNSIGNED_BYTE, image->imageData);

       // Update View port
```

```
       // Update View port
       glutPostRedisplay();
}

static void draw( void ) //draws the scene
{
    double    gl_para[16],back[16];
       static int i=0;
    argDrawMode3D();
    argDraw3dCamera( 0, 0 );
    glClearDepth( 1.0 );
    glClear(GL_DEPTH_BUFFER_BIT);
       glEnable(GL_DEPTH_TEST);
       glEnable(GL_TEXTURE_2D);
    glDepthFunc(GL_LEQUAL);

       if(blend)
              glEnable(GL_BLEND);


    glMatrixMode(GL_TEXTURE);//flip texture from bottop-up to top-down
       glScalef(1.0, -1.0, 1.0);

       /* load the camera transformation matrix */
    argConvGlpara(patt_trans, gl_para);
    glMatrixMode(GL_MODELVIEW);
    glLoadMatrixd( gl_para );

       if (patt_trans[2][1]>=0)
       {
              UpdateTexture(input_video2,image2,true);
              glBlendFunc(GL_DST_COLOR,GL_ZERO);

              glBegin(GL_QUADS);
                                  //we create a surface for the mask
                     glTexCoord2f(0.0f, 0.0f);glVertex3f(-80.0f, -40.0f, 0.0f);
       // Bottom Left
                     glTexCoord2f(1.0f, 0.0f);glVertex3f( 80.0f, -40.0f, 0.0f);
       // Bottom Right
                     glTexCoord2f(1.0f, 1.0f);glVertex3f( 80.0f, -40.0f, 120.0f);//
Top Right
                     glTexCoord2f(0.0f, 1.0f);glVertex3f(-80.0f, -40.0f, 120.0f);//
Top Left
              glEnd();

              UpdateTexture(input_video,image,false);
              glBlendFunc(GL_ONE,GL_ONE);
              glBegin(GL_QUADS);
                                  //we create a surface for the video
                     glTexCoord2f(0.0f, 0.0f);glVertex3f(-80.0f, -40.0f, 0.0f);
       // Bottom Left
                     glTexCoord2f(1.0f, 0.0f);glVertex3f( 80.0f, -40.0f, 0.0f);
       // Bottom Right
                     glTexCoord2f(1.0f, 1.0f);glVertex3f( 80.0f, -40.0f, 120.0f);//
Top Right
                     glTexCoord2f(0.0f, 1.0f);glVertex3f(-80.0f, -40.0f, 120.0f);//
Top Left
              glEnd();
       }
```

```
glDisable(GL_BLEND);
	glDisable( GL_LIGHTING );
	glDisable( GL_DEPTH_TEST );
	}

static void chooseAVI (void) //chooses the right .avi file
{
	static char   path[20];
	static double FrameStamp;
	old_scene=new_scene;

	//save the point (exact frame) where the last video has stopped
	FrameStamp=cvGetCaptureProperty(input_video, CV_CAP_PROP_POS_FRAMES );

	// each scene maps to the correspodent video (the first scene maps to the
first video etc.)
	sprintf(path,"Data/%d.avi",videoNames[new_scene]);
	printf("scene: %d videofile: %d.avi no of videos:
%d\n",new_scene,videoNames[new_scene],videoNu);

	//open the video stream
	input_video=cvCaptureFromAVI(path);
	input_video2=cvCaptureFromAVI(path);

	//continue from the point where the last video has stopped
	cvSetCaptureProperty(input_video,CV_CAP_PROP_POS_FRAMES,FrameStamp);
	cvSetCaptureProperty(input_video2,CV_CAP_PROP_POS_FRAMES,FrameStamp);
}

static void angleToScene (void) //matches the angle to a scene
{
	static int i;

	printf("%f %f %f\n",patt_trans[0][3],patt_trans[1][3],patt_trans[2][3]);
	//calculate the camera direction
	marker_angle=patt_trans[2][0]/patt_trans[0][0];
	for (i=0;i<videoNu;i++)
	{

			if ( marker_angle<tan((3*PI/2)+(i+1)*(PI)/videoNu) )
			{
				new_scene=i;
				break;
			}
	}

}

ALboolean LoadALData(void)//creates an audio buffer and binds it to an audio
source
{
	 // Load wav data into a buffer
    alGenBuffers(1, &Buffer);
    if (alGetError() != AL_NO_ERROR)
	 return AL_FALSE;

	Buffer=alutCreateBufferFromFile("Data/Sound.wav");
```

```
// Bind buffer with a source.
    alGenSources(1, &Source);

    if (alGetError() != AL_NO_ERROR)
        return AL_FALSE;
       //initialize the parameters
    alSourcei (Source, AL_BUFFER,   Buffer   );//bind the buffer to a source
    alSourcef (Source, AL_PITCH,    1.0f     );
    alSourcef (Source, AL_GAIN,     1.0f     );
       alSourcef (Source, AL_REFERENCE_DISTANCE,300.0f);//set the distance model
       alSourcef (Source, AL_ROLLOFF_FACTOR, 2.0f);
    alSourcefv(Source, AL_VELOCITY, SourceVel);
    alSourcei (Source, AL_LOOPING,  AL_TRUE);

        // Do another error check and return.
    if (alGetError() == AL_NO_ERROR)
        return AL_TRUE;

    return AL_FALSE;
}

void SetSourceValues(void)
{
       SourcePos[0]=patt_trans[0][3];//set the audio source position the same as
the marker position
       SourcePos[1]=patt_trans[1][3];
       SourcePos[2]=-patt_trans[2][3];

       //Update the audio source position
       alSourcefv(Source, AL_POSITION, SourcePos);

}

void KillALData()
{
    alDeleteBuffers(1, &Buffer);
    alDeleteSources(1, &Source);
    alutExit();
}

void SetListenerValues()
{
    alListenerfv(AL_POSITION,    ListenerPos);
    alListenerfv(AL_VELOCITY,    ListenerVel);
    alListenerfv(AL_ORIENTATION, ListenerOri);
}
int CountVideos()
{
       static int count=0,name,check;
       hFind=FindFirstFile("Data/*.avi", &FindFileData);
       if(hFind==INVALID_HANDLE_VALUE)
       {
              return -1;
       }
       check=sscanf(FindFileData.cFileName,"%d",&name);
       if(check!=0)
       {
              videoNames[count]=name;
              count++;
       }
```

```
	}
	qsort (videoNames, count, sizeof(int), compare);
	return count;

}

int compare (const void * a, const void * b)
{
  return ( *(int*)a - *(int*)b );
}

void getVideoProperties(CvCapture *input_video)
{

		//Get the total frame number from the video
		NumFrames = (int) cvGetCaptureProperty(input_video,
CV_CAP_PROP_FRAME_COUNT);

		//get the video fps
		FPS =  cvGetCaptureProperty(input_video,  CV_CAP_PROP_FPS);
}

void createMask(void* buffer)// Creates a mask for our texture
{
	void* b = buffer;	// Pointer To The Buffer
	__asm		// Assembler Code To Follow

		mov ecx, 320*240// Set Up A Counter (Dimensions Of Memory Block)
		mov ebx, b	// Points ebx To Our Data (b)
		label:		// Label Used For Looping
			mov ax,0
			add ax,[ebx+0]// checks if color is black
			add ax,[ebx+1]
			add ax,[ebx+2]
			cmp ax,0
			jne label2
			mov [ebx+0],255//if not it turns it to white
			mov [ebx+1],255
			mov [ebx+2],255
			jmp label3
		label2:
			mov [ebx+0],0
			mov [ebx+1],0
			mov [ebx+2],0
		label3:
			add ebx,3// Moves Through The Data By 3 Bytes
			dec ecx// Decreases Our Loop Counter
			jnz label// If Not Zero Jump Back To Label
	}
}

void showError(void)
{
	printf("-no videos found or appropriate video naming-\n\n");
	printf("-----(format: x.avi, where x an integer)-----\n\n");
	Sleep(5000);
	printf("exiting...");
	Sleep(5000);
}
```

Βιβλιογραφία

1. http://www.hitl.washington.edu/artoolkit/
2. http://www.hitl.washington.edu/artoolkit/documentation/userarwork.htm : "*How Does ARToolKit Work?"*
3. http://www.hitl.washington.edu/artoolkit/documentation/devprinciple.htm : "*Development Principles"*
4. http://www.hitl.washington.edu/artoolkit/documentation/devframework.htm : *"The ARToolKit Framework"*
5. http://www.hitl.washington.edu/artoolkit/documentation/tutorialcamera.htm : *"Tutorial 2: Camera and Marker Relationships"*
6. http://www.hitl.washington.edu/artoolkit/documentation/cs.htm : *"Coordinate Systems"*
7. http://www.hitl.washington.edu/artoolkit/documentation/hardware.htm : *"Hardware"*
8. http://www.hitl.washington.edu/artoolkit/documentation/vision.htm : *"Computer Vision Algorithm"*
9. Hirokazu Kato and Mark BillingHurst. "*Marker Tracking and HMD Calibration for a Video-based Augmented Reality Conferencing System"*
10. http://en.wikipedia.org/wiki/ARToolKit
11. http://www.gamedev.net/page/resources/_/reference/programming/opengl/ : *"OpenGL Texture Mapping: An Introduction"*
12. http://www.songho.ca/opengl/gl_pipeline.html : "*OpenGL Rendering Pipeline"*
13. http://www.songho.ca/opengl/gl_transform.html : *"OpenGL Transformation"*
14. http://en.wikipedia.org/wiki/OpenGL : *"Design"*
15. http://www.gamedev.net/page/resources/_/reference/programming/opengl/
16. http://msdn.microsoft.com/en-us/library/ms970772.aspx : *"Scratching the surface of texture mapping"*
17. http://nehe.gamedev.net/ : Tutorials 01-40
18. http://www.cs.princeton.edu/courses/archive/fall00/cs426/lectures/clip/ : *"Clipping"*
19. http://www.opengl.org/resources/faq/technical/transparency.htm : *"Transparency, Translucency and blending"*

20. http://www.devmaster.net/articles/openal-tutorials/lesson1.php : *"OpenAL Lesson 1: Simple Static Sound"*
21. http://en.wikipedia.org/wiki/OpenAL : *"API structure and functionality"*
22. http://en.wikipedia.org/wiki/OpenCV