ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

ΑΝΑΓΝΩΡΙΣΗ ΚΑΙ ΕΠΕΞΕΡΓΑΣΙΑ ΣΕ ΕΦΑΡΜΟΓΕΣ ΕΝΙΣΧΥΜΕΝΗΣ ΠΡΑΓΜΑΤΙΚΟΤΗΤΑΣ ΠΟΛΛΑΠΛΩΝ ΠΡΟΤΥΠΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΣΩΤΗΡΗ ΑΘΑΝΑΣΙΟΥ

Βόλος, Φεβρουάριος 2008

Η σελίδα αυτή είναι σκόπιμα λευκή.

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΡΤΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΗΛΕΠΙΚΟΙΚΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

ΠΟΛΛΑΠΛΗ ΕΠΕΞΕΡΓΑΣΙΑ ΠΡΟΤΥΠΩΝ
Multi - Pattern Matching

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΣΩΤΗΡΗ ΑΘΑΝΑΣΙΟΥ

Επιβλέπων : Αθανάσιος Γκαϊτατζής

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 18η Φεβρουαρίου 2007.

(Υπογραφή)	*(Υπογραφή)*
.....	.....
Ηλίας Ν. Χούστης	Μποζάνης Παναγιώτης
Καθηγητής	Επίκουρος Καθηγητής
Παν. Θεσσαλίας	Παν. Θεσσαλίας

Βόλος, Φεβρουάριος 2007

(Υπογραφή)

.....

ΣΩΤΗΡΗΣ ΑΘΑΝΑΣΙΟΥ

Διπλωματούχος Μηχανικός Ηλεκτρονικών Υπολογιστών, Τηλεπικοινωνιών και Δικτύων Πανεπιστημίου Θεσσαλίας

Περίληψη

Σκοπός αυτής της διπλωματικής εργασίας ήταν η μελέτη συστημάτων Augmented Reality και τα προβλήματα που αντιμετωπίζουν στην ταυτόχρονη αναγνώριση πολλαπλών προτύπων (marker). Οι εφαρμογές Augmented Reality είναι από την φύση τους πολύ απαιτητικές καθώς η παραμικρή καθυστέρηση στην επεξεργασία κάθε frame έχει μεγάλο αντίκτυπο στην διεπαφή με τον χρήστη.

Στην παρούσα μελέτη δόθηκε μεγάλη σημασία στην κάθετη ανάλυση του όλου συστήματος, μελετώντας όλες τις πιθανές βελτιστοποιήσεις που μπορούν να γίνουν και. λαμβάνοντας υπόψη, ότι το σύστημα πρέπει να δουλεύει σε ένα μέσο Ηλεκτρονικό Υπολογιστή χωρίς χρήση εξειδικευμένου υλικού όπως βιομηχανικές κάμερες η επεξεργαστές DSP. Χρήσιμα συμπεράσματα δημιουργήθηκαν από το γεγονός ότι μπορούμε να επιτύχουμε μεγάλη βελτιστοποίηση μιας απαιτητικής εφαρμογής χωρίς να καταφύγουμε σε εξειδικευμένες λύσεις. Γίνεται ανάλυση μιας από τις πιο διαδεδομένες open source βιβλιοθήκες για augmented reality fiducial systems, του ARtoolkit πάνω στο οποίο βασίζονται και τα τελικά συμπεράσματα. Σαν target application ορίστηκε η δημιουργία ενός παιχνιδιού σκακιού, λόγω της ύπαρξης πολλών marker για την υλοποίηση του.

Η μεθοδολογία που αναπτύχθηκε βασίστηκε σε εξειδικευμένη χρήση της βιβλιοθήκης OpenGL, δυνατοτήτων της αρχιτεκτονικής x86 και μπορεί να εφαρμοστεί σε απαιτητικά προβλήματα που είναι απαραίτητο να τρέχουν σε συστήματα τελικού χρήστη. Η μεθοδολογία αποκαλύπτει ότι με λεπτομερή γνώση όλων των δυνατοτήτων ενός συστήματος συνδυαστικά μπορούμε να έχουμε σημαντικές βελτιστοποιήσεις στην τελική εφαρμογή.

Λέξεις Κλειδιά: <<Augmented Reality, x86, OpenGL, optimization, pattern matching>>

Η σελίδα αυτή είναι σκόπιμα λευκή.

Abstract

The purpose of this thesis was to study augmented reality systems and the problems dealing with multiple patterns. Augmented reality applications are high resource consuming in nature, mostly because of the fact that per frame processing delays have great impact on user interaction.

In this thesis a great deal of effort was put into studying all potential optimizations that could be applied, granted that the final system had to perform well on a PC not using specialized industrial cameras or DSPs. Useful conclusions were drawn from the fact that a great deal of optimization can be achieved without resolving into specific solutions. There is an analysis of one of the most important libraries in the field, ARtoolkit upon which the final conclusions are drawn. As a target application a game of chess was chosen because of the need to support multiple markers.

The methodology that was developed was based on specific usage of OpenGL library, and x86 architecture and can be used in numerous similar problems that require to be run on end user machines. The methodology reveals that with an extensive knowledge of the capabilities of a system a great deal of optimization can be achieved in the final application.

Keywords: <<Augmented Reality, x86, OpenGL, optimization, pattern matching>>

Η σελίδα αυτή είναι σκόπιμα λευκή.

Πίνακας περιεχομένων

1 **Εισαγωγή** **1**

1.1 Augmented Reality 1

1.2 Αντικείμενο διπλωματικής 2

1.2.1 Συνεισφορά *3*

1.3 Οργάνωση κειμένου 4

2 **Σχετικές δραστηριότητες και εργασίες.** **5**

2.1 Χρησιμότητα της τεχνολογίας, παραδείγματα δυνατοτήτων. 5

2.2 Υπάρχουσες βιβλιοθήκες γενικού σκοπού. 7

2.3 Fiducials, Markers. 7

3 **Θεωρητικό και τεχνολογικό υπόβαθρο** **11**

3.1 Ψηφιακές εικόνες 11

3.2 Λειτουργία fiducial system based, εφαρμογών Augmented Reality 12

3.2.1 Αρχιτεκτονική του ARtoolkit *13*

3.2.2 Άλλα χαρακτηριστικά του ARtoolkit *15*

3.3 Threads / νήματα 16

3.4 Η αρχιτεκτονική x86 16

3.4.1 Assembly *17*

3.4.2 SIMD *18*

3.5 Η βιβλιοθήκη OpenGL 20

3.6 Λοιπές Τεχνολογίες 22

4 **Ανάλυση Απαιτήσεων Συστήματος** **24**

4.1 Πειραματική ανάλυση δυνατοτήτων του ARtoolkit 24

4.1.1 Συμπεράσματα πειραμάτων: *31*

4.2 Σημεία ενδιαφέροντος 31

4.3 Προγραμματιστικές επιλογές 33

4.4 Αρχικός σχεδιασμός της μη βελτιστοποιημένης εφαρμογής noopt. Λειτουργία και χρήση της βιβλιοθήκης 34

4.4.1 Camera calibration *35*

4.4.2 Pattern Training ... *41*

4.4.3 Ο κώδικας του noopt ... *44*

4.4.4 Αποτελέσματα – feedback ... *46*

4.4.4.1 noopt test-run 1 15sec ... 46

4.4.4.2 noopt test-run 2 15sec ... 49

4.4.4.3 noopt test-run 3 15sec ... 52

4.5 Αρχιτεκτονική συστήματος ... 56

4.6 Περιγραφή Λειτουργιών ... 58

4.6.1 Initialization/GUI Subsystem ... *59*

4.6.2 Marker Detection Subsystem ... *61*

4.6.3 Marker Transformation Calculations ... *61*

4.6.4 Object Rendering Subsystem ... *62*

4.7 Γενική εικόνα ... 63

5 Σχεδίαση Συστήματος ... 66

5.1 Αρχιτεκτονική βελτιστοποιημένου συστήματος ... 66

5.1.1 Αντιμετώπιση του IMCR ... *66*

5.1.2 Marker Detection Optimization ... *71*

5.1.3 Pattern Matching Optimization και Marker Coordinates Transformation Optimization ... *74*

5.1.4 Object Rendering Optimization (ακόμα και κάποια αποτυχία υλοποίησης είναι σημαντικόνα αναφερθεί) ... *76*

5.2 Περιγραφή Συναρτήσεων ... 84

5.2.1 IMCR Specific ... *84*

5.2.1.1 thrHammingD ... 84

5.2.1.2 realHammingD ... 84

5.2.1.3 hammingD ... 84

5.2.2 Allopt Specific ... *85*

5.2.2.1 Check Hardware Functions (initcheck, checkCPU, checkGPU) ... 85

5.2.2.2 Display List Functions (load_glm_obj) ... 85

5.2.2.3 Detect Marker Optimized Functions (arDetectMarkerO, arDetectMarker2O…) ... 86

5.2.2.4 Marker Transformation Functions (arGetTransMatO, arGetTransMatContO) ... 86

5.2.2.5 Monitoring Functions (init_time_file, timer_start, timer_end, timer_print, grab_info) ... 86

5.2.2.6 Rendering Functions (draw_thread, datamalloc, fetch) ... 86

5.3 Κωδικοποίηση αρχείων και δημιουργία τρισδιάστατων μοντέλων ... 87

6 **Υλοποίηση** **92**

6.1 Λεπτομέρειες υλοποίησης 92

6.1.1 Βελτιστοποιήσεις SSE *92*

6.1.1.1 Αρχικός κώδικας σε ένα βρόγχο 92

6.1.1.2 Βελτιστοποίηση κώδικα με χρήση SSE intrinstics 94

6.1.1.3 Τελική βελτιστοιποίηση με χρήση καταχωρητών SSE και inline assembly 97

6.1.2 Βελτιστοποιήσεις MMX *101*

6.1.2.1 Αρχικός κώδικας σε ένα βρόγχο 102

6.1.2.2 Τελική βελτιστοιποίηση με χρήση καταχωρητών MMX και inline assembly 103

6.1.3 Βελτιστοποιήσεις υψηλού επιπέδου. *105*

6.2 Πλατφόρμες και προγραμματιστικά εργαλεία 106

7 **Αξιολόγηση** **108**

7.1 Μεθοδολογία ελέγχου 108

7.2 Αναλυτική παρουσίαση ελέγχου 108

7.2.1 Benchmarks *108*

8 **Επίλογος** **111**

8.1 Σύνοψη και συμπεράσματα 111

8.2 Μελλοντικές επεκτάσεις 111

9 **Βιβλιογραφία** **113**

Εισαγωγή

Augmented Reality

Ο γενικός τομέας της διπλωματικής εργασίας είναι η "Ενισχυμένη πραγματικότητα" (Augmanted Reality). Ο τομέας αυτός πρέπει να τονιστεί, ότι είναι διαφορετικός από τον τομέα της εικονικής πραγματικότητας (Virtual Reality) ως προς το γεγονός, ότι ορίζουμε ως εικονική πραγματικότητα ένα φανταστικό περιβάλλον που ο χρήστης καλείται να πιστέψει ότι προσεγγίζει σε αληθοφάνεια τον πραγματικό κόσμο. Αντιθέτως ο τομέας του Augmanted Reality είναι σαφώς πιο εξειδικευμένος καθώς αποτελείται πρωτίστως από 3 βασικά στοιχεία:

1) Συνδυασμός πραγματικού και εικονικού περιβάλλοντος κάνοντας υπέρθεση (superposition) πραγματικών και εικονικών αντικειμένων.
2) Αλληλεπίδραση εφαρμογής-χρήστη σε πραγματικό χρόνο
3) (Συνήθως) Τρισδιάστατες απεικονίσεις αντικειμένων

Με λίγα λόγια **το αντικείμενο του Augmented Reality είναι συστήματα πραγματικού χρόνου όπου εμφανίζονται ταυτόχρονα πραγματικά και εικονικά αντικείμενα.**

Στην δική μας περίπτωση ο τομέας Augmanted Reality που μας ενδιαφέρει είναι καθαρά *η περίπτωση όπου το σύστημα μας επικοινωνεί με το περιβάλλον μόνο μέσω μιας κάμερας.* Δηλαδή η μόνη πληροφορία που έχουμε είναι βίντεο πραγματικού χρόνου. Οι 2 βασικοί τομείς που καλούνται να αντιμετωπίσουν τέτοιου τύπου εφαρμογές Augmanted Reality είναι:

1. Παρακολούθηση θέσης και μεγέθους αντικειμένων μέσα από βίντεο πραγματικού χρόνου.
2. rendering εικονικών αντικειμένων με λεπτομερείς φωτισμούς, σκιές και περιβαλλοντικές επιδράσεις σε θέσεις προκαθορισμένες από τον χρήστη ανάλογα με το βίντεο εισόδου.

Πρέπει να τονίσουμε ότι η πολυπλοκότητα αυτού του τύπου των εφαρμογών είναι πολύ μεγάλη, και ουσιαστικά **μέσα σε πολύ μικρό χρόνο πρέπει να επιτελέσουμε πάρα πολλούς υπολογισμούς**. Επιπροσθέτως, οι διάφορες βιβλιοθήκες γενικού σκοπού που υπάρχουν , (όσες από αυτές είναι ανοιχτού κώδικα) επικεντρώνονται περισσότερο στην γρήγορη δημιουργία τέτοιων (απλών) εφαρμογών και όχι στην ταχύτητα.

Αντικείμενο διπλωματικής

Αρχικά το πρόβλημα που κληθήκαμε να αντιμετωπίσουμε ήταν η ανάλυση της συμπεριφοράς και λειτουργίας εφαρμογών βασισμένων σε ήδη υπάρχουσες βιβλιοθήκες Augmented Reality και κατά κύριο λόγο η συμπεριφορά τους σε περιπτώσεις παράλληλης επεξεργασίας πολλαπλών προτύπων.

Διαλέξαμε να εργαστούμε πάνω στο ARtoolkit καθώς πρόκειται για βιβλιοθήκη ανοιχτού κώδικα (open source) στοχεύοντας στον σχεδιασμό μιας εφαρμογής αναγνώρισης πολλαπλών προτύπων και την βελτιστοποίηση της αποδοτικότητάς της. Αρχικά, κρίθηκε σκόπιμη η ανάπτυξη κάποιων απλών παραδειγμάτων χωρίς αλλαγές στην δομή της βιβλιοθήκης για εξοικείωση με τον χώρο της ενισχυμένης πραγματικότητας, καθώς και τρέξιμο benchmarks για την μελέτη των δυνατοτήτων υλικού και λογισμικού.

Ο τρόπος με τον οποίο αναγνωρίζεται ένα πρότυπο από την συγκεκριμένη βιβλιοθήκη είναι εύκολο να αναλυθεί. *Πρέπει βέβαια να λάβουμε υπ όψιν το γεγονός ότι πρόκειται για μια βιβλιοθήκη πολύ υψηλού επιπέδου που αλληλεπιδρά με τους οδηγούς (drivers) της κάμερας με την βιβλιοθήκη για απεικόνιση που χρησιμοποιείται (OpenGL) με άλλες βιβλιοθήκες (glut) ακόμα και με ενδιάμεσες βιβλιοθήκες όπως το direct show.* Το θετικό ήταν πως δεν δόθηκε εμβάθυνση σε όλους αυτούς τους τομείς αλλά επικεντρωθήκαμε σε ένα πιο περιορισμένο τομέα σύμφωνα με επιλογές του γράφοντος.

Ουσιαστικώς η μελέτη συγκεντρώνεται σε fiducial systems πολλαπλών markers. (Σημ: fiducial, marker, pattern είναι συνώνυμα και χρησιμοποιούνται από τον συγγραφέα εναλλάξ). **Το κύριο πρόβλημα που καλούμαστε να αντιμετωπίσουμε, είναι η βελτιστοποίηση multi pattern based fiducial systems , διαλέγοντας ως εφαρμογή - στόχο ένα παιχνίδι σκακιού.** Οι λόγοι που η εφαρμογή του σκακιού μας βοηθάει στην συγκεκριμένη περίπτωση να μελετήσουμε multi pattern based συστήματα είναι η μεγάλη πολυπλοκότητα του προβλήματος , πόσο περίπλοκη μπορεί όμως να είναι μια παρτίδα σκακιού; Αναφορικά ο Shannon υπολόγισε σε 10^{120} το κάτω όριο της πολυπλοκότητας ενός παιχνιδιού σκακιού [Shan50] .

Αργότερα κρίθηκαν σκόπιμες πολλές αλλαγές στον τρόπο λειτουργίας της βιβλιοθήκης, θεωρητική μελέτη του συστήματος και η τελική εφαρμογή (του παιχνιδιού σκακιού) που έγιναν οι τελικές και πιο ενδιαφέρουσες μετρήσεις. Αν μπορούσαμε να συνοψίσουμε σε πολύ γενικές γραμμές τα προβλήματα που κληθήκαμε να αντιμετωπίσουμε για την βελτιστοποίηση τέτοιου τύπου εφαρμογών (βασισμένες πάνω στην βιβλιοθήκη ARtoolkit):

1. Ύπαρξη μεγάλου αριθμού βιβλιοθηκών – wrapers σε όλα τα σημεία της βιβλιοθήκης.
2. Μη αξιοποίηση πιθανόν αρχιτεκτονικών δυνατοτήτων του επεξεργαστή , (MMX, SSE, Multi- Threading)
3. Μη αξιοποίηση πιθανόν αρχιτεκτονικών δυνατοτήτων της κάρτας γραφικών
4. Μη αποδοτικός προγραμματισμός σε κάποια σημεία.

Δόθηκε βάση όχι μόνο στο μαθηματικό υπόβαθρο της εφαρμογής, αλλά και στην αρχιτεκτονική του όλου συστήματος καθώς το πρόβλημα δεν είναι καθαρά μαθηματικό αλλά μάλλον περισσότερο υπολογιστικό και θέμα αρχιτεκτονικής του συστήματος.

Συνεισφορά

Η μελέτη που έγινε μπορεί να χωριστεί σε 3 βασικές κατηγορίες:

1. Όλη την διαδικασία για να μπορέσουμε να χρησιμοποιήσουμε την βιβλιοθήκη σε μια εφαρμογή. Τις απαραίτητες ρυθμίσεις στην βιβλιοθήκη, συγγραφή κώδικα για κάποια απλά παραδείγματα. Σε επόμενο στάδιο έγινε χρήση βιβλιοθηκών για φόρτωση μοντέλων και πειράματα σε σχέση με την απόδοση υλικού και λογισμικού (μέγιστη απόσταση αναγνώρισης προτύπων , συμπεριφορά κατά την πολλαπλή αναγνώριση προτύπων, συμπεριφορά κατά την αναγνώριση κινούμενων προτύπων κ.τ.λ.)
2. Σε αυτό το στάδιο ξεκίνησε η διαδικασία των αλλαγών. Πρέπει να κατανοήσουμε ότι λόγω της ύπαρξης ενός μεγάλου επιπέδου layers και βιβλιοθηκών που αλληλεπιδρούσαν μεταξύ τους κρίθηκε αδύνατη η πλήρης ενσωμάτωση όλων των αλλαγών στον κώδικα της βιβλιοθήκης και μέρος των αλλαγών ενσωματώθηκαν σε διαφορετικά header files έτσι ώστε ο χρήστης να μπορεί να καλέσει τις νέες συναρτήσεις αντί των παλαιών. Πρέπει επίσης να

τονιστεί ότι όπου ήταν δυνατό προτιμήθηκε χρήση γλωσσών χαμηλότερου επιπέδου και εισαγωγή συναρτήσεων για συγκεκριμένο υλικό.

3. Στο τελευταίο στάδιο έγινε ο έλεγχος για το κατά πόσο οι αλλαγές αυτές βελτίωσαν την συμπεριφορά του προγράμματος. Αυτό έγινε σε πολλά επίπεδα. Είτε με χρήση προγραμμάτων όπως το intel Vtune το gl debugger και ενίοτε και σε επίπεδο assembly όποτε αυτό κρίθηκε σκόπιμο. Σαν τελικό στάδιο δημιουργήθηκε μια εφαρμογή για απόδειξη των παραπάνω ισχυρισμών.

Οργάνωση κειμένου

Η οργάνωση του κειμένου έχει ως εξής

- κεφάλαιο 2: Εδώ γίνεται μια βιβλιογραφική ανασκόπηση στις δυνατότητες και υλοποιήσεις Augmented Reality εφαρμογών
- κεφάλαιο 3: Μελετούμε τις βασικές τεχνολογίες που χρησιμοποιήθηκαν στην διπλωματική κάνοντας μια θεωρητική επεξήγηση αυτών.
- κεφάλαιο 4: Σχεδιάζουμε πως πρέπει να δουλεύει το σύστημα δίνοντας βάση στους αλγορίθμους, αλληλεπίδραση των επιμέρους τμημάτων του συστήματος και κάνοντας εκτενή αναφορά στις τεχνικές που χρησιμοποιήθηκαν. Παρουσιάζουμε τα αποτελέσματα πειραμάτων που κάναμε πάνω στην βιβλιοθήκη του ARtoolkit που χρησιμοποιήσαμε σαν βάση της μελέτης μας.
- κεφάλαιο 5: Επεξηγείται λεπτομερώς πως δουλεύει το κάθε υποσύστημα. Εδώ γίνεται και οι μαθηματικές επεξηγήσεις καθώς και το dataflow. Αναφέρουμε γιατί κάποιες υλοποιήσεις απέτυχαν εφαρμογής.
- κεφάλαιο 6: Παρουσιάζουμε και εξηγούμε τα πιο σημαντικά τμήματα της υλοποίησης, βήμα βήμα - παρουσίαση του κώδικα.
- κεφάλαιο 7: Παρουσίαση των benchmark καθώς και formal verification τμημάτων του συστήματος.
- κεφάλαιο 8: Σύνοψη

Σχετικές δραστηριότητες και εργασίες.

Η διπλωματική έχει σαν γενικό τομέα το Augmented Reality. Οι εφαρμογές Augmented Reality είναι ουσιαστικά περίπλοκα προγράμματα υψηλού επιπέδου τα οποία χρησιμοποιούν στοιχεία από πολλές και διαφορετικές επιστήμες.

Ποιοτικά μπορούν να αναχθούν σε περίπλοκα συστήματα επεξεργασίας βίντεο, σε προβλήματα εύρεσης οπτικών χαρακτηριστικών η ακόμα και σε προβλήματα ψηφιακής επεξεργασίας σήματος. Σε αυτήν την θεματική ενότητα όμως θα δώσουμε βάση στις εφαρμογές της τεχνολογίας σε διάφορους κλάδους επιστημών και στην γενική εικόνα μιας εφαρμογής Augmented Reality για να κατανοήσουμε τις δυνατότητες αυτού του είδους των προγραμμάτων. Θα μελετηθούν τα κύρια χαρακτηριστικά που απαρτίζουν μια τέτοια εφαρμογή για να παρουσιάσουμε και τις διάφορες τάσεις που επικρατούν σε κάποιους τομείς. Ειδική εισαγωγή γίνεται στην χρήση των Markers που παίζουν καθοριστικό ρόλο στην εφαρμογή μας.

Χρησιμότητα της τεχνολογίας, παραδείγματα δυνατοτήτων.

Ο τομέας του Augmented reality είναι σχετικά καινούριος στην επιστημονική κοινότητα. Παλαιότερα το Augmented reality θεωρούνταν σαν υποκατηγορία του virtual reality. Το πρώτο διεθνές συνέδριο για Augmented reality έγινε τον Σεπτέμβριο του 2002 στη Γερμανία. Έκτοτε οι πιο σημαντικοί θεσμοί στον τομέα έχουν αναδειχτεί το ISAR (International Symposium on Augmented Reality) και ISMAR (International Symposium on Mixed and Augmented Reality). Μέσα σε 5 χρόνια έγιναν πολλές σημαντικές μελέτες σχετικά με τον τομέα και γενικότερα τους τρόπους που μπορεί αυτή η τεχνολογία να επηρεάσει πολλές πτυχές της ζωής μας.

Ικανός αριθμός μελετών έχει γίνει ,για την πιο προφανή από τις χρήσεις συστημάτων Augmented reality, δηλαδή για διασκέδαση. Η πιο γνωστή δημοσίευση είναι η [TCDS00] οι συγγραφείς του οποίου προσπάθησαν να μετατρέψουν το γνωστό παιχνίδι πρώτου προσώπου, Quake έτσι ώστε να παίζεται σε πραγματικό περιβάλλον. Η δημοσίευση [WLHJ04] αναφέρει

πρακτικούς λόγους και δυσκολίες στην οργάνωση των τηλεοπτικών προγραμμάτων για τους οποίους η τεχνολογία αυτή μόνο μερικώς χρησιμοποιείται από την βιομηχανία διασκέδασης

Figure 1 – User interface with tracked gloves and menus

GUI εφαρμογής Augmented reality. Στην συγκεκριμένη περίπτωση πρόκειται για σύστημα που εκτός από κάμερα διαθέτει αισθητήρες στα γάντια που φοράει ο χρήστης καθώς και GPS. Η παραπάνω εικόνα είναι από την δημοσίευση: [PT03]

Σήμερα η τεχνολογία αυτή κάνει δειλά- δειλά κάποιες εμφανίσεις σε δελτία ειδήσεων και σε ποδοσφαιρικούς αγώνες. Αξίζει να σημειωθεί ότι στη δημοσίευση [PWCG01] γίνεται μια ανάλυση για την πρώτη εφαρμογή Augmented reality σε μια ζωντανή συναυλία. Η δημοσίευση [KZK06] παρουσιάζει 5 απλά παιχνίδια που φτιάχτηκαν με το Artoolkit και φτάνει στο συμπέρασμα ότι ένα νέο είδος παιχνιδιών, βασισμένων σε μεγαλύτερη διαδραστικότητα θα μπορούσε να αναπτυχθεί με χρήση ήδη υπαρχόντων τεχνολογιών. Παρακάτω θα μιλήσουμε για τα προβλήματα που αντιμετωπίζουν τέτοιες εφαρμογές augmented reality. και μερικούς τρόπους αντιμετώπισης τους.

Ένας από τους τομείς που παρατηρείται πολύ μεγάλη χρήση του augmented reality είναι το industrial assembling/maintenance. Στην δημοσίευση [JSLJ01] μηχανικοί της BOEING έκαναν πειράματα πάνω στην αύξηση της αποδοτικότητας των εργαζομένων που ήταν υπεύθυνοι για την συντήρηση εξοπλισμού στην εταιρία. Σημαντική είναι επίσης και η μελέτη [HNSI06] η οποία προτείνει ένα υβριδικό σύστημα augmented reality με αισθητήρες αφής για τον χειρισμό ημιαγωγών, ενώ η [LXYF04] εφαρμόζει μια παρόμοια προσέγγιση αλλά για τον χειρισμό νάνο-υλικών.

Υπάρχουσες βιβλιοθήκες γενικού σκοπού.

Εδώ αξίζει να γίνει και μια επιπρόσθετη αναφορά σχετικά με το augmented reality. Το augmented reality σαν κλάδος δεν θέτει αυστηρές προϋποθέσεις για τον τρόπο που μπορεί να λειτουργεί μια εφαρμογή. Αυτό που δίνεται ιδιαίτερη βάση είναι το αποτέλεσμα.

Αν και τα σύνορα μεταξύ augmented reality και mixed reality είναι πολύ θολά στην επιστημονική κοινότητα, γενικά συνήθως το mixed reality θεωρείται υπερσύνολο καθώς συστήματα τέτοιου τύπου συχνότερα βασίζονται και σε άλλες πληροφορίες εκτός από εικόνα για αλληλεπίδραση χρήστη-περιβάλλοντος. Ο τομέας του augmented reality βασίζεται κατά κύριο λόγο στην **αναγνώριση χαρακτηριστικών στο περιβάλλον (Unique feature detection)** και στην **ταυτοποίηση τους (Verification/Identification).** [Fia05]

Κατά το βήμα της αναγνώρισης χαρακτηριστικών, ψάχνουμε στην εικόνα συγκεκριμένα χαρακτηριστικά που έχουμε από πριν αποφασίσει ότι μας ενδιαφέρουν. Σε αυτό το βήμα υπάρχουν 2 διαφορετικές εντελώς φιλοσοφίες: το pattern recognition και το feature-based. Στο feature-based το πρόβλημα αναγνώρισης ενός αντικειμένου αντιμετωπίζεται ψάχνοντας ένα είδος γεωμετρικών χαρακτηριστικών, που μπορεί να είναι για παράδειγμα μια διάταξη ακμών. Η πιο σημαντική συνεισφορά στον τομέα είναι ο αλγόριθμος SIFT. Γενικά η feature-based προσέγγιση του προβλήματος δεν έχει αναπτυχθεί τόσο ακόμα. Οι περισσότερες εφαρμογές augmented reality τρέχουν αλγορίθμους οι οποίοι βασίζονται στο pattern recognition και μπορούμε να αναφερόμαστε συνολικά σε αυτές ως fiducial systems/marker based systems.

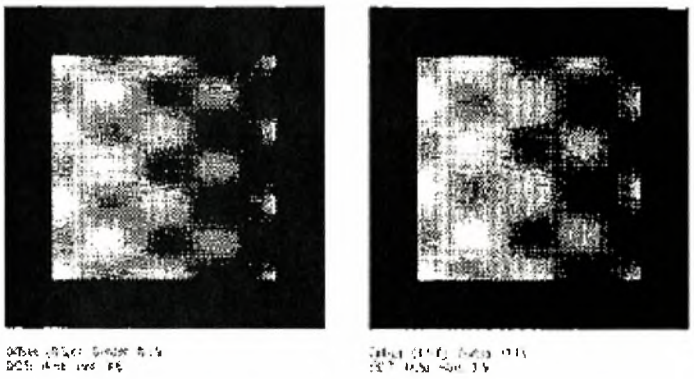
Η επιλογή του pattern recognition για την διαδικασία του Unique Feature Detection δεν επηρεάζει μόνο τους αλγορίθμους που χρησιμοποιούνται. Είναι απαραίτητη η εκτύπωση κάποιων σχεδίων/εικόνων τα οποία στη βιβλιογραφία αναφέρονται ως markers/fiducials/patterns. Ένας μεγάλος αριθμός βιβλιοθηκών για δημιουργία fiducial systems έχει αναπτυχθεί, καθεμία από τις οποίες παρουσιάζουν σημαντικές διαφορές μεταξύ τους. Ένα από τα επικρατέστερα, που έχει εμφανιστεί σε πολλές δημοσιεύσεις είναι το ARtoolkit [KB99]. Άλλες υλοποιήσεις περιλαμβάνουν το ARtag το CyberCode αλλά και ad-hoc υλοποιήσεις, οι οποίες δεν μπορούν να λειτουργήσουν σαν βιβλιοθήκες γενικού σκοπού ενώ έχουν καλύτερη απόδοση σε συγκεκριμένα περιβάλλοντα χρήσης. [TKO07] , [MYN06] .

Fiducials, Markers.

Πρέπει να δώσουμε ιδιαίτερη βάση στις δημοσιεύσεις που έχουν γίνει σχετικά με τα markers/fiducials/patterns που μπορούν να χρησιμοποιηθούν σε εφαρμογές Augmented

reality. Τα markers ουσιαστικά είναι εικόνες (συνήθως ασπρόμαυρες) οι οποίες εκτυπώνονται σε χαρτί. Τις εικόνες αυτές όπως προαναφέρθηκε τις τοποθετούμε στο περιβάλλον σε διάφορα σημεία που μας ενδιαφέρουν με τρόπο ad-hoc. Κάθε βιβλιοθήκη Augmented Reality διατηρεί τους δικούς της τύπους Marker, ενώ μερικές δίνουν στον χρήστη την ελευθερία να φτιάξει δικούς του και να τους κάνει train έτσι να αναγνωρίζονται από το σύστημα.

Είναι απαραίτητο σε αυτό το σημείο να πούμε ότι η μορφή που έχουν οι markers ενός fiducial system επηρεάζουν ριζικά τον τρόπο λειτουργίας του εκάστοτε εργαλείου. Είναι επίσης αλήθεια ότι τα περισσότερα fiducial systems λειτουργούν με τετράγωνους markers. Αυτό όμως δεν είναι απαραίτητο καθώς υπάρχουν και συστήματα που λειτουργούν με έγχρωμα δαχτυλίδια [TRIP]. Στο [OXM01] γίνεται μια ανάλυση που αποκαλύπτει κάποιες πολύ σοβαρές πτυχές της τεχνολογίας αυτής και βασιζόμενοι στο γεγονός ότι το ανθρώπινο μάτι δεν είναι απαραίτητο πάντα να ξεχωρίζει τους Markers μεταξύ τους και προτείνεται η χρήση patterns που το εσωτερικό τους δημιουργείται από ημιτονοειδείς συναρτήσεις.



παράδειγμα marker.

η παραπάνω εικόνα είναι από την δημοσίευση: [OXM01]

Υπάρχουν και άλλες διαφορετικές προσεγγίσεις στο παραπάνω θέμα. Καταρχήν το [TKO07] διαπραγματεύεται την εμφωλευμένη τοποθέτηση Marker έτσι ώστε ανάλογα με την απόσταση του viewport από το Marker να χρησιμοποιείται διαφορετικού μεγέθους Marker. Αυτό μπορεί να βοηθήσει κατά την κίνηση του viewport προς το Marker η σε περιπτώσεις που παρουσιάζονται φαινόμενα partial occlusion. Η ίδια μεθοδολογία μπορεί να βοηθήσει στο καλύτερο calibration της εικόνας.

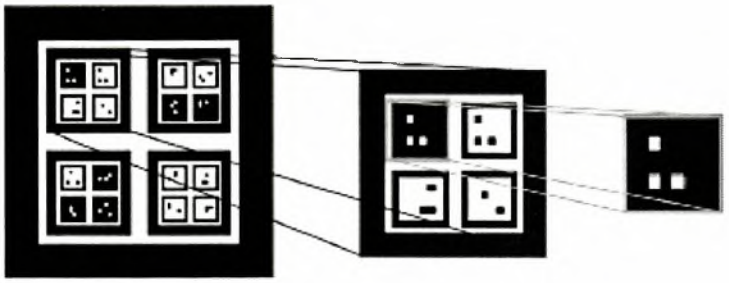
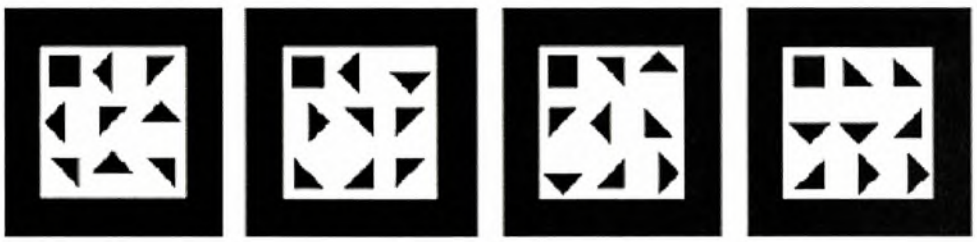


Figure 1. Nested Marker

παράδειγμα nested marker.

η παραπάνω εικόνα είναι από την δημοσίευση: [TKO07]

Άλλες μελέτες [MYN06] θέτουν ως κύριο στόχο την γρήγορη δημιουργία τυποποιημένων Markers , δίνοντας έμφαση στο μεγάλο μέγεθος πιθανών αποτελεσμάτων (2^{18} = 262,144). Η συγκεκριμένη υλοποίηση χρησιμοποιεί τα 2 τελευταία σύμβολα για έλεγχο, διατηρώντας το checksum για εύρεση πιθανών λαθών.



παράδειγμα TRICODES

η παραπάνω εικόνα είναι από την δημοσίευση: [MYN06]

Μετά από αυτή την παρουσίαση διάφορων μελετών στον τομέα των fiducial systems κρίνεται σκόπιμη η αναφορά ότι άλλες λύσεις δεν περιλαμβάνονται στο παρόν κείμενο. Οι λόγοι είναι η μη επαρκής πληροφόρηση και το ότι μερικά από αυτά τα συστήματα είναι εμπορικού σκοπού και όχι ανοιχτού κώδικα.

Ίσως το πιο διαδεδομένο σύστημα που όμως περνά απαρατήρητο είναι αυτό του barcode. Το barcode είναι ένα fiducial system που όμως δεν προορίζεται για χρήση σε εφαρμογές augmented reality. Ο μοναδικός σκοπός του είναι η αποκωδικοποίηση κάποιων χαρακτηριστικών που βρίσκονται πάνω στο προϊόν. Ο marker σε αυτήν την περίπτωση παρίσταται από τις κατακόρυφες γραμμές οι οποίες σχηματίζουν ένα παραλληλόγραμμο.

Κάθε γραμμή αντιπροσωπεύει έναν αριθμό που μεταφράζεται από το σύστημα, παράγοντας τον κωδικό του προϊόντος.

Θεωρητικό και τεχνολογικό υπόβαθρο

Σε αυτό το σημείο είναι κρίσιμο να εμβαθύνουμε σε κάποιους τεχνολογικούς τομείς που η κατανόηση της λειτουργίας τους είναι κρίσιμη για την κατανόηση του σκοπού που επιτελεί η διπλωματική. Κατά βάση το παρόν κεφάλαιο επικεντρώνεται στον τρόπο λειτουργίας των fiducial systems που είναι υπεύθυνα για δημιουργία εφαρμογών Augmented Reality.

Γίνεται ανάλυση αρχικά σε επίπεδο θεωρίας και των μεθοδολογιών που βασίζονται τέτοια συστήματα, και δίνουμε βάση στην συγκεκριμένη βιβλιοθήκη που μελετήθηκε, το ARtoolkit. Μετά δίνεται μια ανάλυση κάποιων χαρακτηριστικών ήδη υπαρχουσών τεχνολογιών που σχετίζονται έμμεσα με την υλοποίηση τέτοιων εφαρμογών , και συγκεκριμένα της αρχιτεκτονικής x86 ,της βιβλιοθήκης OpenGL και προγραμματιστικών εργαλείων όπως τα νήματα (Threads).

Η λεπτομερής ανάλυση όλων των παραπάνω αποδεικνύεται χρήσιμη για την κατανόηση της επίδρασης συγκεκριμένων τεχνολογιών πάνω στο πρόβλημα μας. Επίσης η εις βάθος γνώση τους μας βοηθάει να κατανοήσουμε, γιατί κάποιες μέθοδοι οι οποίες αναλύονται σε επόμενα κεφάλαια απέτυχαν να επιλύσουν το συγκεκριμένο πρόβλημα

Ψηφιακές εικόνες

Μια ψηφιακή εικόνα στην δική μας περίπτωση δημιουργείται από δειγματοληψία μιας αναλογικής. Για αυτήν την διαδικασία είναι υπεύθυνη η κάμερα. Αφαιρετικά μπορούμε να θεωρήσουμε ότι μια εικόνα με μέγεθος $640*480$ αποτελείται από ένα δισδιάστατο πίνακα με διαστάσεις $x=640\ y=480$. Κάθε στοιχείο του πίνακα το οποίο ονομάζουμε pixel περιέχει κάποιες πληροφορίες για την εικόνα.

Τα πράγματα όμως είναι κάπως πιο περίπλοκα. Οι εικόνες μπορούν να έχουν διάφορες μορφές: Binary (δυαδικές), Grayscale (ασπρόμαυρες), έγχρωμες

(RGB,RGBI,RGBA...) Ανάλογα με το είδος απεικόνισής τους ο δισδιάστατος πίνακας που απεικονίζει τις εικόνες αλλάζει.

Για μια εικόνα σε Binary μορφή ο πίνακας είναι $x*y*2$ bits. Η Binary εικόνα θεωρούμε ότι έχει 1 bit colour. Δηλαδή 2^m όπου m=1 το bit depth ή colour depth. Η μεταβλητή m είναι πολύ σημαντική καθώς καθορίζει τον αριθμό των bits που χρησιμοποιούνται για να απεικονίσουν την απόχρωση ενός μοναδικού pixel. Κάνοντας την πράξη 2^m μπορούμε να βρούμε τον αριθμό αποχρώσεων που μπορούν να απεικονιστούν. Στην περίπτωση της Binary εικόνας $2^1=2$ άρα απεικονίζουμε 2 αποχρώσεις. Γενικά ο τύπος για την εύρεση του μεγέθους του πίνακα που παριστάνει μια εικόνα είναι $x*y*2^m$. Αν τώρα έχουμε μια εικόνα έγχρωμη σε μορφή RGB χρειαζόμαστε $3*x*y*2^m$. Το 3 δηλώνει ότι ουσιαστικά χρειαζόμαστε 3 πίνακες μεγέθους $x*y*2^m$, που καθένας από αυτούς απεικονίζει ένα από τα 3 βασικά χρώματα R(red), G(green), B(blue). Για καθέναν από αυτούς τους πίνακες διατηρούμε 2^m πληροφορία για κάθε bit.

Έτσι συνολικά ένα οποιοδήποτε χρώμα μπορεί να θεωρηθεί γραμμικός συνδυασμός κόκκινου, πράσινου και κίτρινου:

$$f_c(\bar{r}) = a_R * f_R(\bar{r}) + a_G * f_G(\bar{r}) + a_B * f_B(\bar{r})$$

Σαν παράδειγμα των παραπάνω μπορούμε να αναφέρουμε πως μια εικόνα διαστάσεων $640 \square 480$ σε binary μορφή (χωρίς κανενός είδους συμπίεση) χρειάζεται χώρο: $(640*480*2^1)/(8*10^3) = 76.8\text{KByte}$.

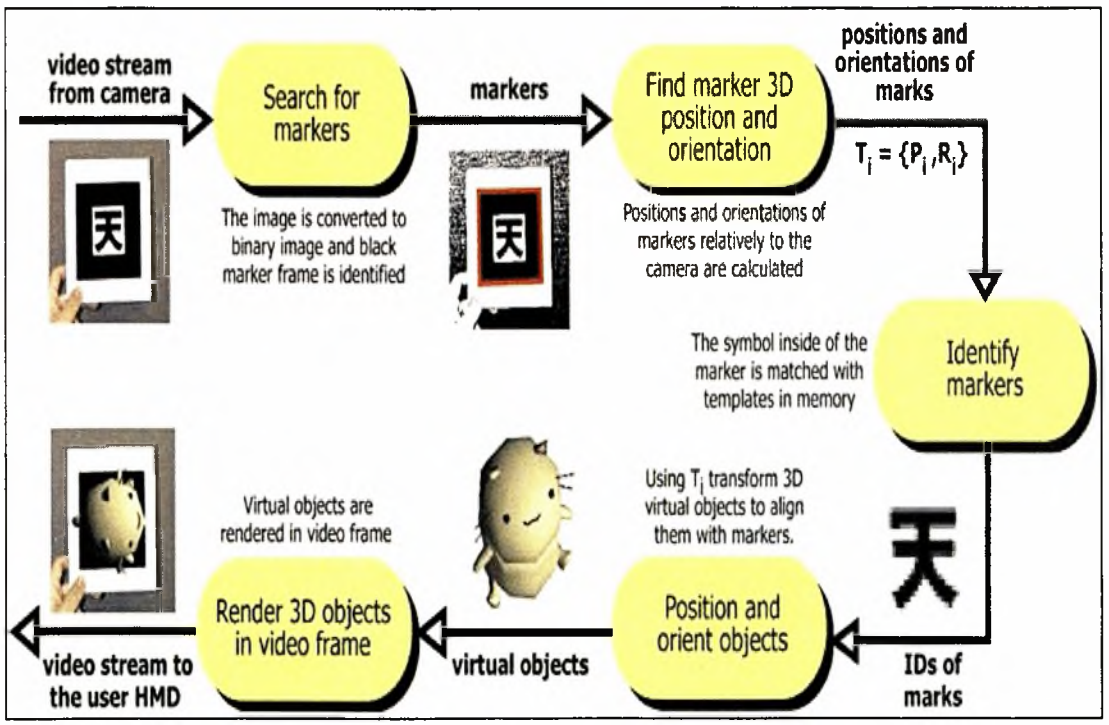
Λειτουργία fiducial system based, εφαρμογών Augmented Reality

Γενικά στα fiducial systems πολύ μεγάλο ρόλο παίζουν τα fiducials καθεαυτά. Όπως έγινε φανερό από την προηγούμενη ενότητα οι πιο πολλές εφαρμογές augmented reality δεν αποτελούνται μόνο από κώδικα καθώς η χρήση fiducials και το γεγονός ότι η εκάστοτε τοποθέτηση τους στον πραγματικό κόσμο επηρεάζει την ροή του προγράμματος αυξάνει την πολυπλοκότητα μελέτης μιας τέτοιας εφαρμογής καθώς ουσιαστικά μπορούμε να την θεωρήσουμε (σε συνδυασμό με το πρόβλημα του σκακιού που είναι και το ζητούμενο στην δική μας περίπτωση) ένα ανοιχτό σύστημα. Αυτό συνεπάγεται πολλά προβλήματα τα οποία μελετούνται διεξοδικά στην επόμενη ενότητα και είναι από μόνα τους ένας τομέας προς μελέτη.

Σε αυτό το σημείο πρέπει να κάνουμε μια ανάλυση κάποιων ήδη υπαρχουσών τεχνολογιών Augmented Reality που αναλύθηκαν σε αυτή την διπλωματική, ξεκινώντας από την βιβλιοθήκη ARtoolkit.

Όλες οι εφαρμογές Augmented Reality λαμβάνουν σε πραγματικό χρόνο βίντεο από κάποια κάμερα. Ουσιαστικά όλες οι εφαρμογές Augmented Reality τέτοιου τύπου μπορούν να αναχθούν σε ένα πρόβλημα Real Time Video Processing (επεξεργασία βίντεο σε πραγματικό χρόνο). Γιατί όμως πραγματικό χρόνο; Το σύστημα χρειάζεται μέσα σε ελάχιστο χρόνο να επεξεργαστεί την είσοδο του, και να επιστρέψει το αποτέλεσμα έτσι ώστε πιθανές ενέργειες του χρήστη (πχ αλλαγή θέσης της κάμερας) να έχουν άμεσο feedback.

Ας επικεντρωθούμε στον τρόπο λειτουργίας του ARtoolkit για να κατανοήσουμε γιατί είναι τόσο σημαντικός παράγοντας ο χρόνος, στον συγκεκριμένο τύπο εφαρμογών. Μια κάμερα στέλνει ένα video feed στον υπολογιστή. Το video feed αυτό δεν διαβάζεται από δευτερεύουσα μνήμη η από άλλο αποθηκευτικό μέσο στην περίπτωση μας αλλά απ' ευθείας από την κάμερα. Το ARtoolkit χρησιμοποιεί την βιβλιοθήκη DSVL [Pin05] η οποία ουσιαστικά λειτουργεί σαν ένας wrapper της βιβλιοθήκης DirectShow.



Σχεδιάγραμμα λειτουργίας του Artoolkit.

η παραπάνω εικόνα είναι από τον δικτυακό τόπο: "http://www.hitl.washington.edu/artoolkit/"

Το video feed αυτό αποτελείται από frames, δηλαδή εικόνες οι οποίες εναλλάσσονται πολύ γρήγορα και μας δίνουν την ψευδαίσθηση ότι η εικόνα κινείται. Αυτό οφείλεται στην ιδιότητα του ματιού να διατηρεί την εικόνα για λίγο χρόνο αφού την έχει "δει". Το φαινόμενο αυτό ονομάζεται **persistence of vision.** Frame rate ονομάζουμε τον αριθμό frames (που αποτελείται ένα βίντεο) στην μονάδα του χρόνου. Η ποσότητα δηλαδή 30FPS μπορεί να μεταφραστεί σαν $30\frac{frames}{sec}$ με κάθε frame να εναλλάσσεται ανά $\frac{1}{30} = 0.033333333\ sec.$

Εφόσον λοιπόν φτάσει το πρώτο frame από το video feed στον υπολογιστή, ξεκινάει η διαδικασία. Γίνεται capture λοιπόν αρχικά ένα frame. Ύστερα το frame μετατρέπεται σε Binary image μέσω thresholding. Το **thresholding** είναι μια διαδικασία κατά την οποία κάθε pixel του frame που έχουμε συγκρίνεται με μια τιμή (την οποία στο ARtoolkit μπορεί να καθορίσει ο χρήστης). Αν η τιμή του pixel είναι μικρότερη από την τιμή του threshold τότε το pixel γίνεται mark σαν object pixel. Αλλιώς το θεωρούμε σαν backround pixel (και δεν ασχολούμαστε πια με αυτό). Ψάχνουμε επίσης σε αυτό το στάδιο να εντοπίσουμε ενωμένο αριθμό από pixel τα οποία ουσιαστικά είναι τα αντικείμενα μας.

Ελέγχουμε στο Binary image που παράχθηκε το περίγραμμα των αντικειμένων και ελέγχουμε αν μπορεί να "χωρέσει" σε τέσσερις ευθείες (οι οποίες σχηματίζουν το marker μας αν έχει εντοπιστεί σωστά). [KB99] Προφανώς μας ενδιαφέρουν μόνο οι περιοχές που σημειώθηκαν με 1 και στην ουσία είναι οι μαύρες περιοχές του binary image. Αποθηκεύουμε τα σημεία τομής των τεσσάρων ευθειών, σε σχέση πάντα με το viewport της κάμερας. Τα σημεία τομής εφόσον όντως πρόκειται για markers ορίζουν ένα επίπεδο μονοσήμαντα και επίσης μας βοηθούν στο λεγόμενο **pose estimation**.

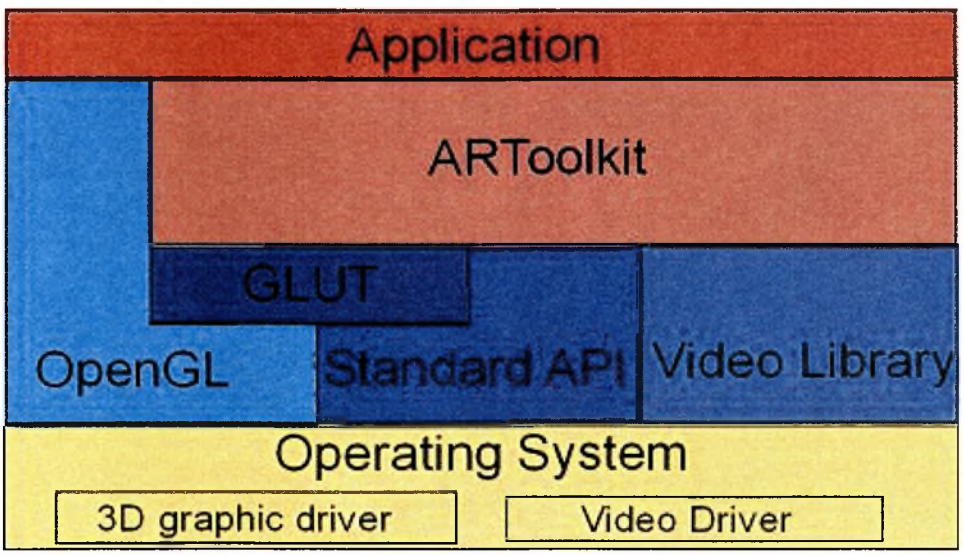
Γίνεται μια δειγματοληψία τώρα του εσωτερικού χώρου κάθε μιας από τις περιοχές που η περίμετρος τους είναι τετράγωνη. Συνήθως γίνεται σε έναν 16x16 πίνακα δηλαδή 256 συνολικά δείγματα. Χρησιμοποιούμε μετά τις συντεταγμένες τομής των ευθειών για να πολλαπλασιάσουμε τους παραπάνω πίνακες και ουσιαστικά να τους φέρουμε στη σωστή θέση για σύγκριση με τα γνωστά patterns. Οι περιοχές που περικλείουν οι 4 γραμμές, και αποτελούν πιθανούς markers γίνονται normalized και συγκρίνονται με τους markers που έχει δώσει σαν είσοδο στο πρόγραμμα ο χρήστης.(περισσότερα για το pattern training πιο κάτω). Πρέπει να τονίσουμε πως οι markers που βρέθηκαν στο frame μπορεί να έχουν υποστεί διάφορες διανυσματικές πράξεις (translation, rotation, scale) σε σχέση με τους markers που έχουμε στην ήδη μνήμη. Έτσι ο πίνακας που χρησιμοποιούμε για το normalization υπολογίζεται λαμβάνοντας υπόψη τις συντεταγμένες των 4 κορυφών (που

τέμνονται οι ευθείες) και τις συντεταγμένες της οθόνης. Η διαδικασία δηλαδή που χρησιμοποιούμε για object recognition είναι το **correlation**.

Εφόσον λοιπόν εντοπιστεί ένας marker από αυτούς που μας ενδιαφέρουν χρησιμοποιώντας τα αποτελέσματα από τα προηγούμενα βήματα που μας βοήθησαν κατά τον υπολογισμό της σχετικής θέσης του marker σε σχέση με το camera POV **σχεδιάζουμε** κάποιο μοντέλο με την βοήθεια OpenGL, η ακόμα και στέλνουμε ένα video feed πάνω στην θέση του marker κάνοντας **superposition** πάνω στο αρχικό frame. Στην δική μας περίπτωση μας ενδιαφέρει η προβολή τρισδιάστατων μοντέλων πιονιών σκακιού πάνω σε κάθε marker.

Άλλα χαρακτηριστικά του ARtoolkit

Εδώ πρέπει να τονίσουμε ότι η βιβλιοθήκη προσφέρει τρόπο για την **εκπαίδευση νέων προτύπων (pattern training)** δηλαδή την δημιουργία καινούριων προτύπων και η εισαγωγή τους στην βιβλιοθήκη. Κατά την διαδικασία αυτή αρχικά σχεδιάζεται το πρότυπο που μας ενδιαφέρει σε ένα πρόγραμμα επεξεργασίας εικόνας (πχ photoshop). Μετά εκτυπώνεται και τοποθετείται μπροστά από την κάμερα. Ο χρήστης μετά καλείται να τρέξει την κατάλληλη ρουτίνα έτσι ώστε να αναγνωριστεί το πρότυπο και αφού υποστεί μια μετατροπή να αποθηκευτεί σαν αρχείο στον σκληρό δίσκο. Ουσιαστικά αυτό που γίνεται είναι ότι μετά από εξαγωγή του frame, μέσα στο αρχείο αποθηκεύεται το εσωτερικό του marker σε **4 διαφορετικά στιγμιότυπα**. Κάθε στιγμιότυπο έχει υποστεί μια περιστροφή 90° έτσι ώστε να γίνονται γρηγορότερα οι συγκρίσεις κατά το στάδιο του verification. Μετά κατά την δημιουργία εφαρμογής που κάνει χρήση της βιβλιοθήκης ο χρήστης καλείται να εισάγει σαν ορίσματα τα αρχεία αυτά που έχουν δημιουργηθεί κατά την εκπαίδευση προτύπων και να αναλάβει να επιλέξει την συμπεριφορά του προγράμματος σε περίπτωση εμφάνισης τους σε ένα frame.



Η αρχιτεκτονική του ARtoolkit

η παραπάνω εικόνα είναι από τον δικτυακό τόπο: "http://www.hitl.washington.edu/artoolkit/"

Εδώ πρέπει να τονιστεί η όλη **πολυπλοκότητα της βιβλιοθήκης** καθώς και οι εξαρτήσεις από άλλες βιβλιοθήκες πράγμα που θα μπορούσε να την κατατάξει σαν βιβλιοθήκη υψηλού επιπέδου. (βλέπε παραπάνω σχήμα). Αυτό δυσκόλεψε την ομαλή και οργανωμένη μελέτη του προβλήματος καθώς ένα μεγάλο ποσοστό της απόδοσης εξαρτιόταν από άλλες βιβλιοθήκες χαμηλότερου επιπέδου. Η χρήση για παράδειγμα της βιβλιοθήκης GLUT δυσκόλεψε ως ένα σημείο την μελέτη καθώς, λειτουργώντας ως wrapper του API του λειτουργικού (στην συγκεκριμένη περίπτωση του WIN32) δεν έδινε πρόσβαση σε μεγάλο αριθμό μεταβλητών χρήσιμων κατά την μετατροπή του προγράμματος σε multi-threaded.

Τέτοια προβλήματα αντιμετωπίστηκαν με αντικατάσταση ολόκληρων μεθόδων της βιβλιοθήκης με κώδικα χαμηλού επιπέδου αλλά και με αλλαγές στην όλη διαδικασία λειτουργίας της βιβλιοθήκης. Τελικώς κρίθηκε σκόπιμη η συγγραφή του τελικού προγράμματος από την αρχή για καλύτερο έλεγχο και πρόσβαση σε μεγάλο αριθμό μεταβλητών που χρησιμοποιούνταν.

Threads / νήματα

Τα νήματα (Threads) είναι ένας αφηρημένος τύπος δεδομένων που μπορεί να βελτιώσει την απόδοση προγραμμάτων. Ο πολυνηματικός προγραμματισμός μπορεί να έχει πολύ μεγάλη διαφορά σε αποδοτικότητα, ανάλογα με το υλικό που έχει ο εκάστοτε υπολογιστής. Τα Threads μοιράζονται κοινές θέσεις στη μνήμη και έτσι μπορούν να ανταλλάξουν πολύ εύκολα δεδομένα μεταξύ τους από τις διεργασίες (processes), ενώ η εναλλαγή μεταξύ Threads της ίδιας διεργασίας είναι σαφώς πιο γρήγορη από τις εναλλαγές μεταξύ διαφορετικών διεργασιών. Επίσης πρέπει να τονίσουμε ότι η ταυτόχρονη εκτέλεση κώδικα από Threads μπορούν να βελτιώσουν την αποδοτικότητα μιας εφαρμογής ακόμα και σε επεξεργαστές μονού πυρήνα, ενώ στους καινούριους multi-core επεξεργαστές η βελτιστοποίηση είναι σαφώς μεγαλύτερη καθώς το λειτουργικό δίνει σε κάθε πυρήνα να εκτελέσει από ένα Thread.

Η αρχιτεκτονική x86

Ένα αρκετά σημαντικό κομμάτι της διπλωματικής είχε να κάνει με βελτιστοποίηση. Ο πλέον αποδεδειγμένος τρόπος για βελτιστοποίηση μιας εφαρμογής πραγματικού χρόνου (real time) είναι η όσο γίνεται καλύτερη αξιοποίηση του υλικού από το

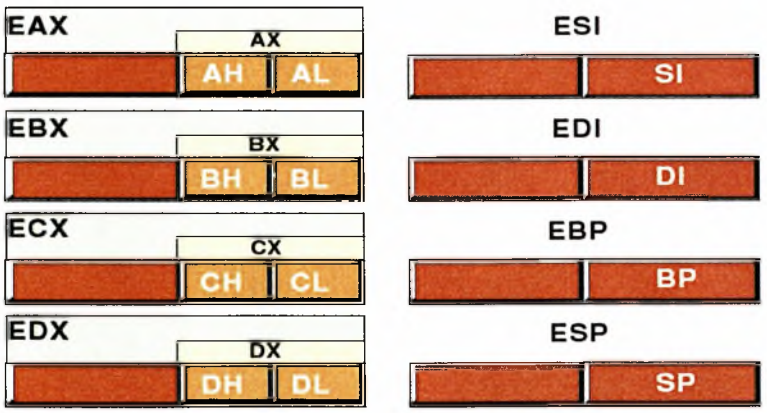
λογισμικό. Είναι όντως αλήθεια πως οι σημερινοί μεταγλωττιστές (compilers) κάνουν πολύ καλό optimization ανάλογα με τις παραμέτρους που τους περνάς. Όμως αυτό αφαιρεί τον απαιτούμενο έλεγχο που χρειάζεται να έχει ένας προγραμματιστής. Σε εφαρμογές real time ο σκοπός είναι η μέγιστη απόδοση του προγράμματος στο διαθέσιμο hardware έτσι ώστε να φαίνεται σαν η εφαρμογή να δουλεύει σε πραγματικό χρόνο κάτι πρακτικά αδύνατον καθώς ΠΑΝΤΑ θα χρειάζεται να δαπανηθεί χρόνος για επεξεργασία. Η αρχιτεκτονική x86 μπορούμε να θεωρήσουμε ότι ανήκει στην κατηγορία μηχανών αρχιτεκτονικής Von Neumann. Ο υπολογιστής σε αυτήν την περίπτωση μπορεί να μοντελοποιηθεί σαν 3 στοιχεία που συνδέονται με το system bus. Τον επεξεργαστή, την κύρια μνήμη και τις συσκευές εισόδου εξόδου.

Assembly

Οι επεξεργαστές αρχιτεκτονικής x86 δίνουν πρόσβαση στις εφαρμογές σε έναν αριθμό καταχωρητών. Οι καταχωρητές αυτοί χωρίζονται σε κατηγορίες ανάλογα με το μέγεθος τους.

- EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP μεγέθους 32-bit
- AX, BX, CX, DX, SI, DI, BP, SP μεγέθους 16-bit
- AL, AH, BL, BH, CL, CH, DL, DH μεγέθους 8-bit

Οι καταχωρητές αυτοί δεν είναι ξεχωριστοί μεταξύ τους αλλά υπάρχει επικάλυψη, όπως φαίνεται στο παρακάτω σχήμα:



Καταχωρητές αρχιτεκτονικής x86

Η εικόνα είναι από το βιβλίο [Hyde00]

Στην πραγματικότητα αν και ονομάζονται καταχωρητές γενικού σκοπού στην ουσία ο κάθε ένας συνίσταται για συγκεκριμένη λειτουργία. Για παράδειγμα ο ESP/SP χρησιμοποιείται για την αποθήκευση του stack pointer. Αν κάποιος ακολουθεί την

προτεινόμενη λειτουργία κάθε καταχωρητή, ένας αριθμός εντολών πιθανώς συμπεριφέρονται πιο αποδοτικά.

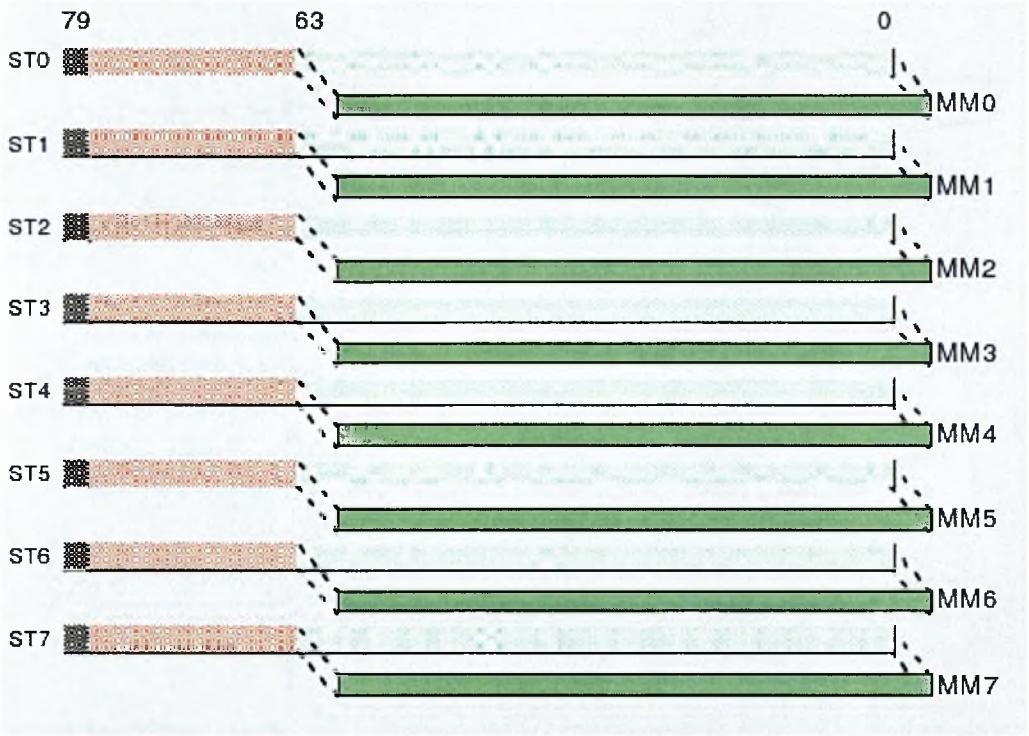
Πρέπει να γνωρίζουμε πως ο compiler αρχικά μετατρέπει το πρόγραμμα σε assembly προτού το μετατρέψει σε γλώσσα μηχανής. Αυτό πολύ συχνά μπορεί να περνάει απαρατήρητο σε απλές εφαρμογές που είτε δεν χρειάζεται να εκτελούν τόσες πράξεις είτε ο χρόνος δεν είναι τόσο καθοριστικός παράγοντας.

Η χρήση assembly μπορεί να έχει πολύ μεγάλες επιπτώσεις στο πρόγραμμα. Μπορούμε να έχουμε πολύ μεγάλη βελτιστοποίηση κατά την χρήση της. Φυσικά στην περίπτωση μας ο σκοπός δεν είναι και ούτε θα μπορούσε να είναι, η συγγραφή όλου του προγράμματος σε assembly αλλά περισσότερο η αλλαγή συγκεκριμένων πράξεων οι λεγόμενες κυρίαρχες πράξεις (dominant operations). Οι κυρίαρχες πράξεις παίζουν μεγάλο ρόλο στην δική μας περίπτωση. Αν επί παραδείγματι χρειάζεται να εφαρμόσουμε μια πράξη σε μια εικόνα 320x240 (που είναι το πραγματικό μέγεθος του βίντεο που λαμβάνουμε από την κάμερα) ουσιαστικά καλούμαστε να εφαρμόσουμε για 76.800 pixels μια πράξη τουλάχιστον, οπότε προγραμματιστικά παίζει πολύ μεγάλο ρόλο τι μπαίνει μέσα στο for loop. Στο κεφάλαιο 7 αναλύουμε διεξοδικά τέτοιου είδους περιπτώσεις.

SIMD

Η τεχνική Single Instruction Multiple Data (SIMD) είναι μια τεχνική παραλληλοποίησης υπολογισμών. Όπως λέει και το όνομα του μιλούμε ουσιαστικά ταυτόχρονες για πράξεις σε πολλά δεδομένα μέσω μιας μοναδικής εντολής. Πολλοί εμπορικού τύπου επεξεργαστές προσφέρουν κάποια μορφή SIMD ξεκινώντας από τον Pentium-MMX.

- **MMX**. Ύπαρξη 8 καταχωρητών *MM0, MM1, MM2, MM3, MM4, MM5, MM6, MM7* μεγέθους 64-bit. Βέβαια πρέπει να τονιστεί ότι δεν είναι ανεξάρτητοι καταχωρητές αλλά πρόκειται ουσιαστικά για μέρος των καταχωρητών του FPU (Floating Point Unit). Αυτό μας αποτρέπει από την ταυτόχρονη χρήση αριθμών κινητής υποδιαστολής μαζί με την τεχνολογία MMX. Έτσι ουσιαστικά ο επεξεργαστής χρειάζεται να εναλλάσσει τρόπο λειτουργίας και να εναλλάσσεται ο τρόπος χρήσης των συγκεκριμένων καταχωρητών μεταξύ των πράξεων της FPU και MMX. [int02] Μέσα σε ένα τέτοιο καταχωρητή μπορεί να αποθηκευτεί είτε ένας πίνακας 8 στοιχείων 8-bit (byte), είτε ένας πίνακας 4 στοιχείων 16-bit (word), είτε ένας πίνακας 2 στοιχείων 32-bit (double word). Ανάλογα με την χρήση που κάνουμε μπορούμε να επιτύχουμε ένα καλό επίπεδο παράλληλης επεξεργασίας. [Hyde00]



επικάλυψη καταχωρητών MMX και FPU

Η εικόνα είναι από το βιβλίο [Hyde00]

- **SSE**. 8 καινούριοι καταχωρητές μεγέθους 128-bit ο καθένας. *XMM0, XMM1, XMM2, XMM3, XMM4, XMM5, XMM6, XMM7*. Οι καταχωρητές SSE είναι πιο ευέλικτοι από αυτούς που χρησιμοποιούνται κατά τις πράξεις MMX, καθώς παρέχουν υποστήριξη για δεκαδικούς αριθμούς.

Η βιβλιοθήκη OpenGL

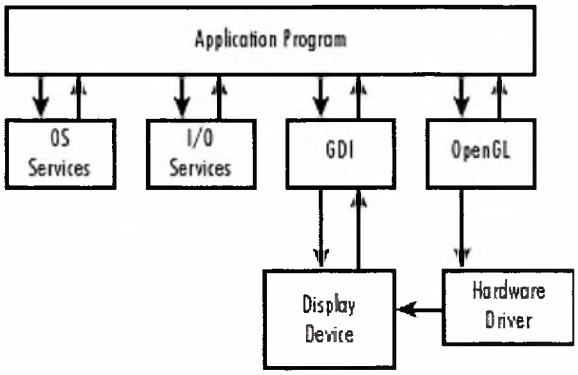
Το OpenGL ορίζεται σαν ένα API (Application Programming Interface) και ο επίσημος ορισμός του είναι "a software interface to graphics hardware" [OGLRe]. Ο κύριος σκοπός του OpenGL είναι η συμβατότητα σε μια ευρεία γκάμα υλικού (hardware) και λειτουργικών συστημάτων (Operating Systems, OS). Υλοποιήσεις του OpenGL υπάρχουν για Mac OS, Microsoft Windows, Linux, Unix. Για να ικανοποιούνται οι παραπάνω απαιτήσεις του compatibility το OpenGL είναι προσανατολισμένο μόνο στην επεξεργασία γραφικών και δεν περιλαμβάνει συναρτήσεις για window management, διαχείριση αρχείων, συσκευών I/O η υποστήριξη ήχου. Αυτό σημαίνει ότι ο προγραμματιστής πρέπει να βασιστεί σε βιβλιοθήκες που εξαρτώνται από το εκάστοτε λειτουργικό σύστημα που χρησιμοποιεί για την δημιουργία του παραθύρου της εφαρμογής ή για το user input. Συγκεκριμένα στην περίπτωση μας στα Microsoft Windows γίνεται χρήση του Win32 API.

Το OpenGL μπορεί να θεωρηθεί σαν μια βιβλιοθήκη που αλληλεπιδρά με την κάρτα γραφικών του συστήματος και υποστηρίζεται από μεγάλο εύρος προγραμματιστικών γλωσσών, όπως C, C++, C#, Java κ.α. Το OpenGL επίσης μπορεί να οριστεί ως ένα διαδικαστικό και όχι περιγραφικό API. Αυτό από την άποψη ότι στο OpenGL δεν περιγράφεις το αποτέλεσμα που θέλεις να επιτύχεις αλλά αντίθετα δίνεις τα βήματα που πρέπει να γίνουν για να παράγεις το επιθυμητό αποτέλεσμα. Για παράδειγμα αν θέλεις να εμφανιστεί στην οθόνη σου μια κόκκινη σφαίρα δεν δίνεις μια μαγική εντολή για να ζωγραφιστεί η σφαίρα κόκκινη και στο σημείο που θες. Πρέπει πρώτα να σχεδιάσεις την σφαίρα , να την μεταφέρεις στις κατάλληλες συντεταγμένες σε σχέση με το viewport σου, να εφαρμόσεις κάποιον φωτισμό ή texture και μετά να δώσεις την εντολή να ζωγραφιστεί στην οθόνη.

Το παραπάνω αιτιολογεί ότι μπορούμε να δούμε το OpenGL σαν μια μηχανή καταστάσεων (state machine). Αυτό μπορεί να εξηγηθεί πρακτικά από το γεγονός ότι κλήση συναρτήσεων OpenGL παραμένουν σε ισχύ μέχρι να επέλθει η αλλαγή τους. Οι καταστάσεις του OpenGL αλλάζουν συνήθως με εντολές όπως glEnable() ή glDisable(). Επίσης συνήθως κάθε μεταβλητή κατάστασης έχει κάποια default τιμή αν δεν έχει αρχικοποιηθεί διαφορετικά. [OGLBI]

Σήμερα πλέον όλες σχεδόν οι κάρτες γραφικών περιλαμβάνουν στους drivers τους κάποια υλοποίηση του OpenGL. Το τμήμα του driver της κάρτας γραφικών δέχεται απευθείας κλίσεις από το OpenGL χωρίς την μεσολάβηση του λειτουργικού συστήματος. Αν κάποια κλίση είναι υλοποιημένη σε επίπεδο hardware (στην κάρτα γραφικών) τότε η

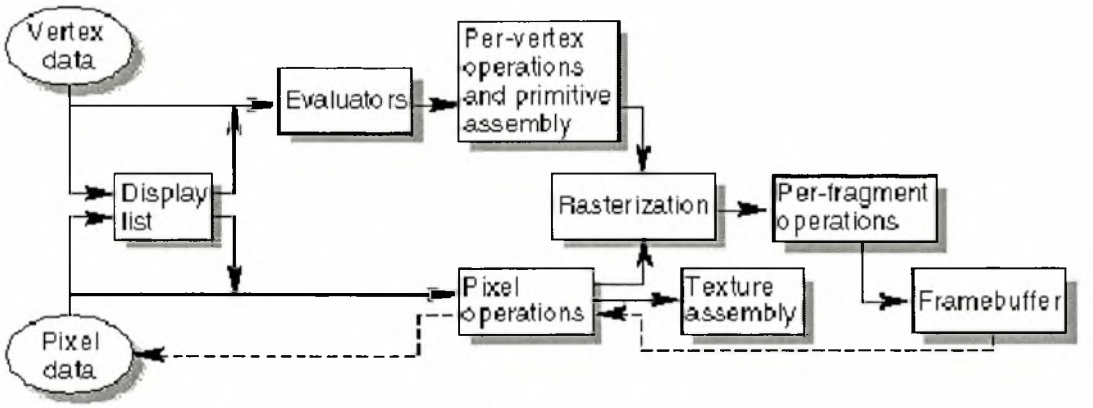
επεξεργασία γίνεται σε επίπεδο GPU (graphics processing unit) αλλιώς γίνεται προσομοίωση στην CPU (central processing unit).



Σχεδιάγραμμα τρόπου αλληλεπίδρασης OpenGL , παρουσία OpenGL driver

Η εικόνα είναι από το βιβλίο [OGLSu]

Γενικά, όλες οι κλήσεις συναρτήσεων OpenGL εκτός από τις αλλαγές κατάστασης



λειτουργίας περνάνε από το OpenGL pipeline για επεξεργασία.

Αφαιρετική προσέγγιση του OpenGL pipeline

Η εικόνα είναι από το βιβλίο [OGLRe]

Πρέπει επίσης να τονίσουμε 2 ακόμα χαρακτηριστικά του OpenGL τα οποία παίζουν καθοριστικό ρόλο στην ανάλυση απαιτήσεων του συστήματος.

3. GLSL. Με την εμφάνιση του OpenGL 2.0 η πιο σημαντική βελτίωση ήταν η GLSL (Open**GL** **S**hading **L**anguage). Μπορούμε να θεωρήσουμε ότι το OpenGL χωρίς να λάβουμε υπ όψιν μας την GLSL μας δίνει ένα fixed functionality. Το GLSL επιτρέπει ουσιαστικά την απευθείας εκτέλεση εντολών στην GPU και οπτικά μοιάζει

με την γλώσσα προγραμματισμού C. Η διαφορά είναι ότι οι πράξεις που γίνονται σε GLSL εκτελούνται στο GPU πολύ πιο γρήγορα από το να εκτελούνταν στο CPU. Οι κάρτες γραφικών των 4 τελευταίων γενεών ουσιαστικά υποστηρίζουν OpenGL 2.0 και μπορούν να θεωρηθούν ως πια ως μιας μορφής επεξεργαστές γενικού σκοπού. Τυπικά αυτοί οι επεξεργαστές μας επιτρέπουν εφαρμογές GPGPU (General-purpose computing on graphics processing units) δηλαδή επεξεργασία δεδομένων γενικού σκοπού, εκμεταλλευόμενοι την μεγάλη παραλληλία που υπάρχει στον επεξεργαστή μιας κάρτας γραφικών.

4. Display Lists. Για πολλές εφαρμογές OpenGL η ταχύτητα δεν είναι το ζητούμενο, αλλά περισσότερο η ποιότητα του αποτελέσματος. Στο Augmented Reality όμως κατά κύριο λόγο μας ενδιαφέρει η όσο το δυνατόν γρηγορότερη εμφάνιση των overlayed objects στην οθόνη. Για παράδειγμα για να ζωγραφίσουμε ένα τρίγωνο στην οθόνη μπορεί να χρειαστούμε μέχρι και 11 κλήσεις συναρτήσεων, όμως στην δική μας περίπτωση που το target application έχει οριστεί να είναι μια εφαρμογή σκακιού κάτι τέτοιο δεν είναι δυνατό. Κάθε τρισδιάστατο μοντέλο υλοποιήθηκε χειροκίνητα στα προγράμματα Maya learning Edition και blender. Κάθε ένα από αυτά περιλαμβάνει αρκετές εκατοντάδες πολύγωνα. Εδώ έρχονται να μας βοηθήσουν οι display lists. Προτού οι εντολές σταλθούν για επεξεργασία στην GPU αποθηκεύονται προσωρινά σε ένα buffer. Μόλις ο buffer γεμίσει οι εντολές στέλνονται για επεξεργασία στο GPU. Πρέπει να τονιστεί ότι οι εντολές που δίνουμε στο OpenGL χρειάζεται να μετατραπούν σε χαμηλού επιπέδου εντολές συμβατές με το εκάστοτε hardware. Τα display lists είναι ουσιαστικά προ-επεξεργασμένες εντολές οι οποίες μας επιτρέπουν την γρηγορότερη εκτέλεση εντολών, δεδομένου ότι δεν αλλάζουμε τις εντολές. (από την στιγμή που μπουν σε ένα display list και γίνει compile) [OGLRe]

Λοιπές Τεχνολογίες

Στα πλαίσια της τελικής εφαρμογής του σκακιού χρησιμοποιήθηκε και ένας αριθμός τρισδιάστατων μοντέλων για να απεικονίσουν τα πιόνια του σκακιού. Τα περισσότερα δημιουργήθηκαν από τον συγγραφέα και μερικά χρησιμοποιούνται υπό την χρήση άδειας GPL. Μιας και δεν αποτέλεσαν τον κυρίαρχο παράγοντα στην εργασία δεν αναλύονται οι μεθοδολογίες κατασκευής τους. Αναφέρουμε απλά ότι χρησιμοποιήθηκαν τα προγράμματα Blender (open source) και Maya (learning edition).

Τα στοιχεία που μας ενδιέφεραν σε σχέση με την διπλωματική ήταν ο αριθμός τριγώνων (triangle count) και ο τύπος του μοντέλου για την εισαγωγή τους στο

πρόγραμμα. Ο αριθμός τριγώνων είναι σημαντικός καθώς όσο πιο μεγάλος είναι ο αριθμός τριγώνων που αποτελείται ένα μοντέλο τόσο περισσότερος χρόνος χρειάζεται για να επεξεργαστεί από την κάρτα γραφικών. Όπου κρίθηκε απαραίτητο δεν χρησιμοποιήθηκαν υφές (textures) ενώ δεν χρησιμοποιήθηκαν και άλλες υπάρχουσες τεχνολογίες (υποστηριζόμενες πια από όλα σχεδόν τα πακέτα σχεδίασης) όπως καμπύλες bezier (bezier curves) και NURBS (Nonuniform rational B-Splines).

Ανάλυση Απαιτήσεων Συστήματος

Σε αυτό το κεφάλαιο θα ακολουθήσει η περιγραφή της αρχιτεκτονικής του συστήματος μας και θα γίνει η ανάλυση απαιτήσεων για τις λειτουργίες του.

Πειραματική ανάλυση δυνατοτήτων του ARtoolkit

Προτού προχωρήσουμε στην αρχιτεκτονική του συστήματος μας πρέπει να αναλύσουμε κάποια πρακτικά συμπεράσματα κατά την μελέτη του ARtoolkit. Όπως προαναφέρθηκε στο κεφάλαιο 3.2.1 το ARtoolkit βασίζεται στις βιβλιοθήκες DSVL και glut. Η συγκεκριμένη έκδοση που μελετήσαμε ήταν η 2.71.2. Τα "gDEBugger version 2.5.0.3785" και "Intel Vtune Performance Analyzer" χρησιμοποιήθηκαν για τις διάφορες μετρήσεις εύρεσης χρόνου.

Το μηχάνημα που πραγματοποιήθηκαν τα ακόλουθα πειράματα είχε το ακόλουθο setup:

Setup μηχανήματος	
Λειτουργικό σύστημα:	**Microsoft Windows XP Proffesional Edition, SP2**
Κάμερα:	**Creative Webcam (WMD)**
Επεξεργαστής:	**Athlon XP 2500+ ,Barton (MMX, SSE, 3DNow!)**
Κάρτα γραφικών:	**Nvidia 6600GT (υποστηρίζει OpenGL 2.0)**

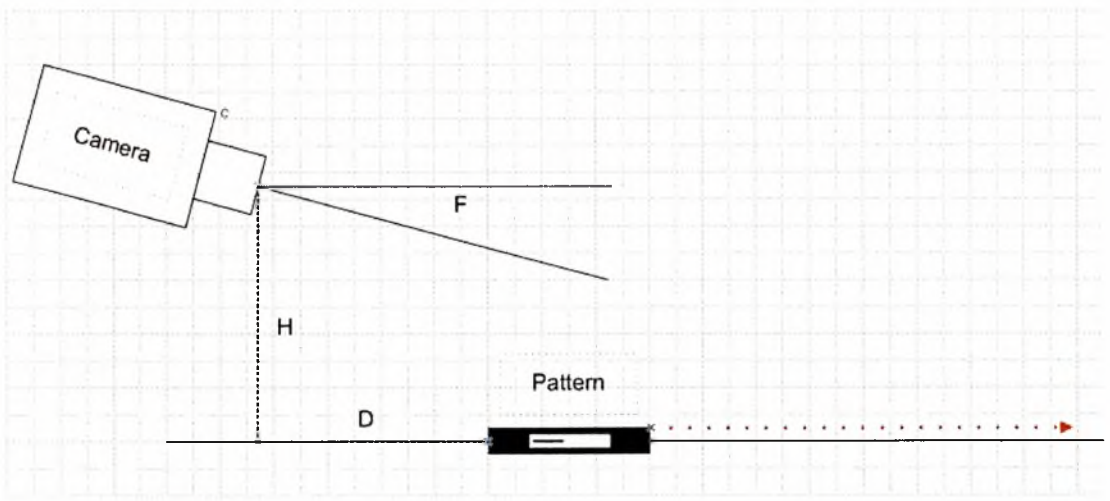
α β γ δ ε

Τα patterns που χρησιμοποιήθηκαν στα πειράματα

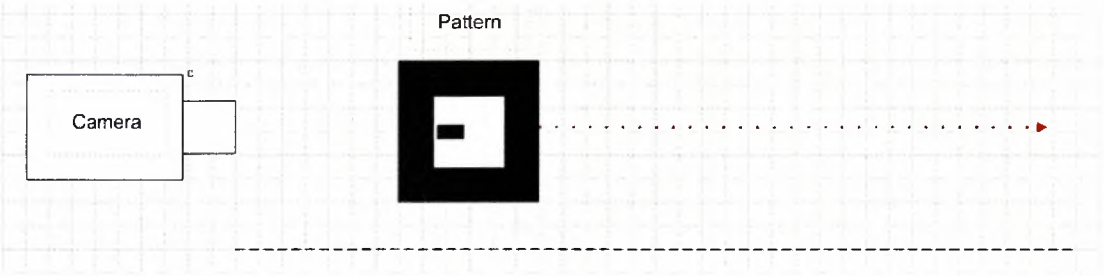
Από τα patterns που χρησιμοποιήθηκαν τα α,β,γ,δ περιέχονταν ήδη στο ARtoolkit ενώ το ε κατασκευάστηκε ξεχωριστά και έγινε train με το αντίστοιχο υποπρόγραμμα του ARtoolkit. Παρακάτω θα δούμε τα σχηματικά από τα διάφορα πειράματα τα οποία βέβαια δεν μπορούν να δώσουν και την ακριβή εικόνα. Ο χρήστης μπορεί να ανατρέξει στην σελίδα www.inf.uth.gr/~soathana και στην επιλογή videos να δει τα videos που τραβήχτηκαν κατά την διάρκεια των πειραμάτων.

Τα πειράματα που πραγματοποιήθηκαν ήταν 4 κατηγοριών:

1. Μέτρηση δυνατότητας του ARtoolkit να αναγνωρίζει patterns κατά την αλλαγή της απόστασης από την κάμερα.. Σε αυτό το βήμα χρησιμοποιήθηκε το pattern γ
    a) Διαστάσεις εκτυπωμένου sample1 Pattern: 8*8cm
    b) Camera angle **F: -45 μοίρες** (σταθερή)
    c) Απόσταση κάμερας από pattern **D: 29.5cm (αρχική)**
    d) Απόσταση κάμερας από οριζόντιο επίπεδο **H: 41.2cm** (σταθερή)
    e) Απόσταση κάμερας από pattern **D: 100.0cm (Τελική)**



Οριζόντια απεικόνιση του πειράματος



Κάθετη απεικόνιση του πειράματος

Αρχικό screenshot

Τελικό screenshot

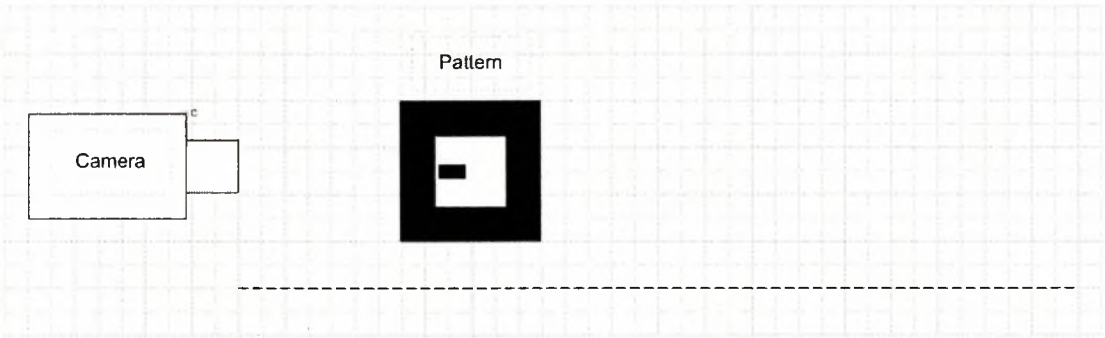
2. Μέτρηση δυνατότητας του ARtoolkit να αναγνωρίζει patterns κατά την εφαρμογή μιας περιστροφής της κάμερας.
    a) Διαστάσεις εκτυπωμένου sample1 Pattern: 8*8cm

    b) Camera angle **F: -45 μοίρες (αρχική)**

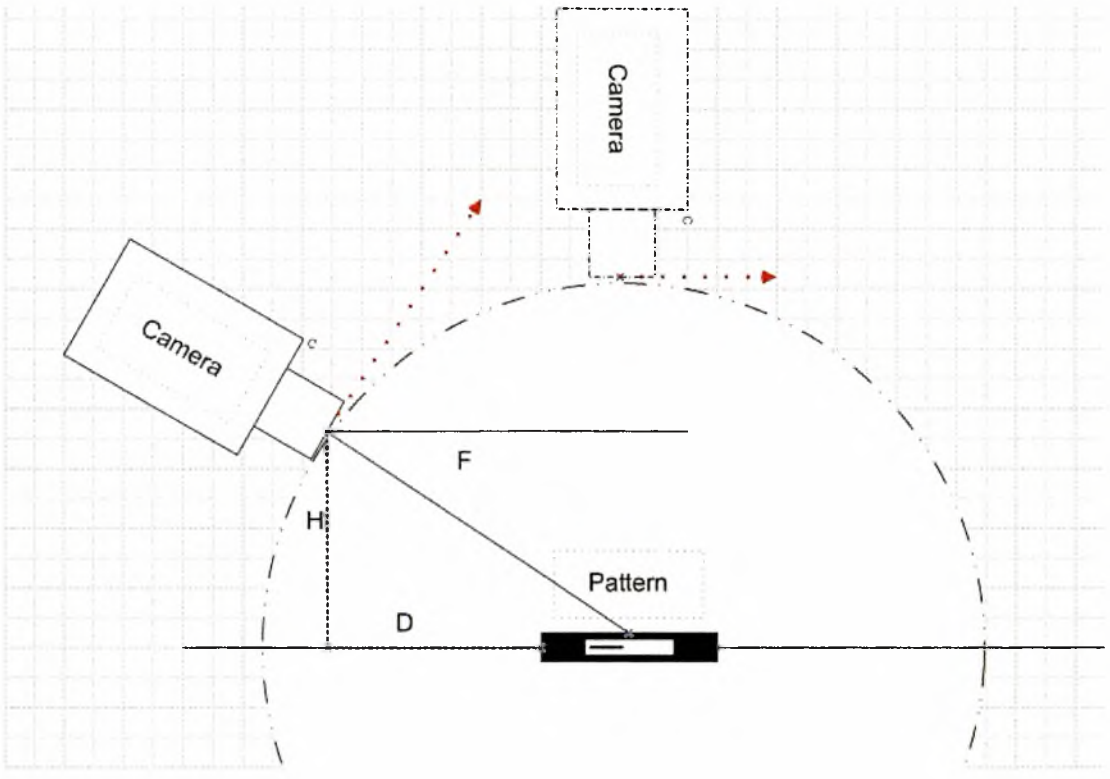
    c) Απόσταση κάμερας από pattern **D: 57.4cm** (σταθερή)

    d) Απόσταση κάμερας από οριζόντιο επίπεδο **H: 41.6cm** (μεταβαλλόμενη)

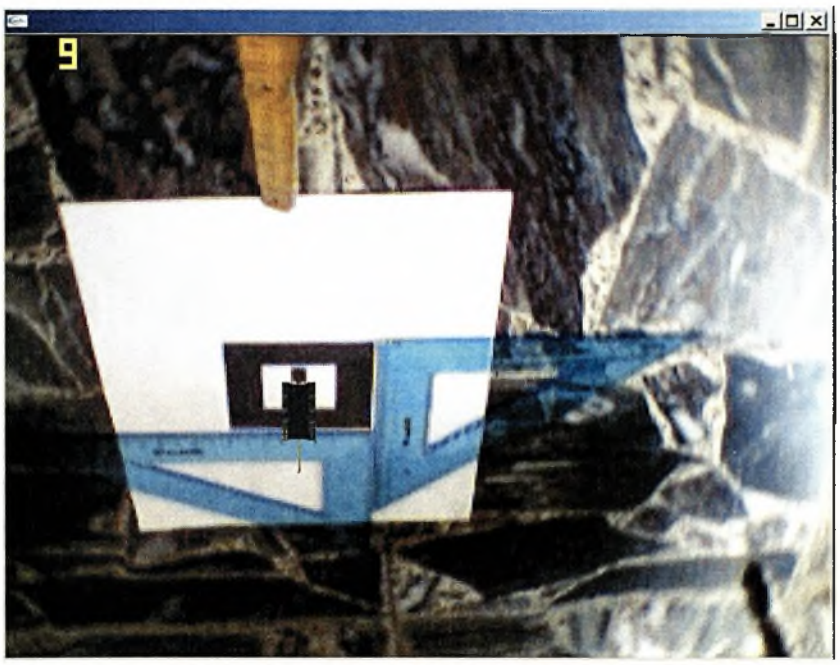
    e) Camera angle **F: -90 μοίρες (τελική)**



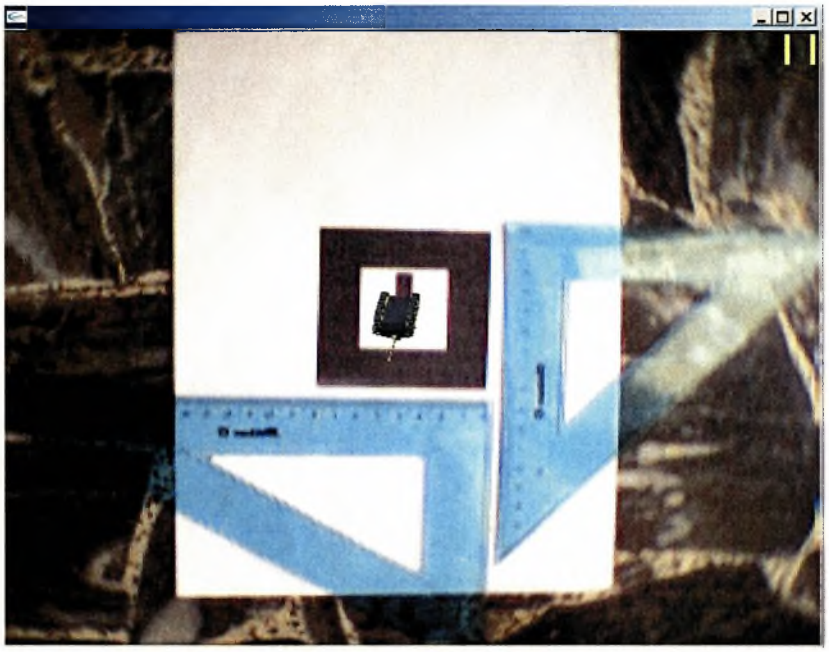
Κάθετη απεικόνιση του πειράματος □



Οριζόντια απεικόνιση του πειράματος



Αρχικό screenshot:



Τελικό screenshot:

3. Μέτρηση χρόνου του ARtoolkit να αναγνωρίζει patterns.

όνομα	διαστάσεις	χρόνοι αναγνώρισης και απεικόνισης pattern
Hiro	8x8cm	**2.576026 ms** **3.010159 ms** **2.790020 ms**
Kanji	8x8cm	**2.186591 ms** **2.249727 ms** **2.387454 ms**
sample1	8x8cm	**2.774654 ms** **2.049423 ms** **2.503391 ms**
uth	12x12cm	**2.661232 ms** **2.943949 ms** **2.859861 ms**

a) Camera angle **F: -85 μοίρες** (σταθερή)

b) Απόσταση κάμερας από pattern **D: 60-70cm** (μικρές αποκλίσεις ανάλογα με το pattern)

c) Απόσταση κάμερας από οριζόντιο επίπεδο **H: 42.0cm** (σταθερή)

Τα time intervals που αναφέρονται παραπάνω είναι οι χρόνοι εκτέλεσης της arDetectMarker αφού το pattern έχει ήδη αναγνωριστεί μια φορά. Για λόγους εγκυρότητας μετρήθηκαν σε κάθε pattern 3 φορές τα time intervals σε ξεχωριστές εκτελέσεις του προγράμματος. Όλα τα patterns όπως και η camera παρέμειναν σταθερά κατά τις εκτελέσεις.

4. Μέτρηση δυνατότητας του ARtoolkit να αναγνωρίζει πολλαπλά patterns.
   a) Camera angle **F: -47 μοίρες** (αρχική)

   b) Απόσταση κάμερας από pattern **D: 60-70cm** (μικρές αποκλίσεις σε κάθε pattern)

   c) Απόσταση κάμερας από οριζόντιο επίπεδο **H: 41cm** (σταθερή)

screenshot

Συμπεράσματα πειραμάτων:

Στα πειράματα αυτά τα αποτελέσματα μπορούν να μας αποδείξουν μερικά από τα ελαττώματα του ARtoolkit, τα περισσότερα από αυτά αναμενόμενα μετά και από την θεωρητική ανάλυση που κάναμε στο προηγούμενο κεφάλαιο. Ειδικά ο φωτισμός παίζει καθοριστικό ρόλο κατά την αναγνώριση των σχημάτων. Τα αποτελέσματα επίσης είναι σε ένα βαθμό ενδεικτικά, καθώς οποιοδήποτε αλλαγή στο υλικό ενδέχεται να τα επηρεάσει. Η μη ύπαρξη πιο εξειδικευμένου εξοπλισμού, για τον έλεγχο της φωτεινότητας η κάποιου μηχανικού συστήματος για να βοηθήσει κατά την μετακίνηση της κάμερας (όλα έγιναν χειροκίνητα) ίσως επηρεάζει αρνητικά τα αποτελέσματα.

Τα πειράματα αυτά επιβεβαιώνονται και από αριθμό δημοσιεύσεων που μας προσφέρουν επιπλέον πληροφορίες. Σύμφωνα με την [Fia05] το πρόβλημα που παρατηρήσαμε με τον φωτισμό οφείλεται ως επί το πλείστον στην διαδικασία του thresholding . Επίσης η διαδικασία του correlation που χρησιμοποιείται για την σύγκριση των διάφορων markers έχει μεγάλη επίπτωση στο **false positive detection (FPD)** δηλαδή στην λανθασμένη ταυτοποίηση ενός pattern που βρέθηκε στο frame με ένα pattern που έχουμε φορτώσει στην μνήμη. Γενικά στην βιβλιογραφία αναφέρεται πως ο καλύτερος τρόπος σύγκρισης μεταξύ 2 markers σε fiducial systems είναι ο υπολογισμός του **hamming distance**. Το hamming distance είναι ουσιαστικά το αποτέλεσμα της σύγκρισης των 2 πινάκων που περιέχουν τις εικόνες (θεωρώντας τις εικόνας σας συμβολοσειρές) και εφαρμόζοντας την πράξη XOR. Βέβαια εδώ πρέπει να παρατηρήσουμε πως στις δημοσιεύσεις [DBD04 , KB99] εμφανίζονται και εκεί αποτελέσματα από πειράματα πάνω στο ARtoolkit μάλιστα πολύ διαφορετικά από τα αποτελέσματα που βρήκαμε εμείς. Αυτό οφείλεται στα εντελώς διαφορετικά setup των μηχανημάτων (για τους χρόνους) όπως επίσης και για τις εντελώς διαφορετικές συνθήκες που επικρατούσαν στα περιβάλλοντα που έγιναν τα πειράματα (φως).

Σημεία ενδιαφέροντος

Μερικά από τα πιο ενδιαφέροντα σημεία δεν τα έχουμε ακόμα αναπτύξει λόγω της ειδικότητας τους, και εξαρτώνται πάντα από την εκάστοτε συγκεκριμένη υλοποίηση του συστήματος καθώς και από τις εκάστοτε περιβαλλοντικές συνθήκες που επικρατούν.

false positive detection (FPD): Μη αναγνώριση ενός marker που υπάρχει στο frame. **false negative detection (FND):** Αναγνώριση ενός marker την στιγμή που δεν υπάρχει όμως στο frame μας. **inter-marker confusion rate (IMCR):** Πιθανό "μπέρδεμα" κατά την αναγνώριση των markers έτσι ώστε ένας marker να αναγνωριστεί ως ένας άλλος. **minimal market size (MMS):** Το ελάχιστο μέγεθος που χρειάζεται να έχει ένας marker (σε pixel) έτσι ώστε να αναγνωριστεί επιτυχώς από το σύστημα. Επιπλέον τα παραπάνω μπορούν να

επηρεαστούν από **θόρυβο** και ενδεχομένως από **μερική επικάλυψη** των markers (partial occlusion).

Τα παραπάνω στην δική μας περίπτωση πρέπει να τα αντιμετωπίσουμε προφανώς με συγκεκριμένους τρόπους που πιθανόν δίνουν διαφορετικές προσεγγίσεις από αυτές των ήδη υπάρχουσων βιβλιοθηκών. Η μερική επικάλυψη για παράδειγμα που μπορεί να έχει ένας marker είναι ζωτικής σημασίας για την εφαρμογή μας. Ο λόγος είναι ότι στην εφαρμογή του σκακιού οι χρήστες - παίχτες χρειάζεται να μετακινούν τον εκάστοτε marker για να κάνουν μια κίνηση. Πάνω στον κάθε marker θα ζωγραφίζεται το μοντέλο του πιονιού του σκακιού. Ας μελετήσουμε λίγο πιο διεξοδικά το θέμα αυτό. Παρακάτω βλέπουμε μια εικόνα σκακιού:

Το ξεκίνημα ενός παιχνιδιού σκακιού

η παραπάνω εικόνα είναι από τον δικτυακό τόπο: wikipedia

Μπορούμε σε αυτό το στάδιο να κάνουμε μερικές πολύ χρήσιμες παρατηρήσεις. Συνολικά στο σκάκι χρειαζόμαστε 32 πιόνια. Θα ήταν όμως σοφό να χρησιμοποιήσουμε άραγε 32 διαφορετικούς markers που ο καθένας θα αντιπροσωπεύει από ένα ξεχωριστό τρισδιάστατο μοντέλο; Φυσικά αυτό δεν είναι σωστό. Πρέπει καταρχάς να δεχτούμε ότι καθώς οι παίχτες παίζουν την εφαρμογή του σκακιού (που θα είναι βασισμένη σε Augmented Reality) όταν κάποιο πιόνι νικιέται από κάποιο άλλο, απλά ο παίχτης θα γυρνάει ανάποδα την κάρτα με το marker, εμποδίζοντας έτσι το μοντέλο να ζωγραφιστεί στα επόμενα frame. Αυτή η προσέγγιση μας βοηθάει πολύ καθώς το ζητούμενο απλοποιείται κάπως καθώς δεν είναι απαραίτητη η εισαγωγή στο πρόγραμμα από τους παίχτες των marker που ανταποκρίνονται στο πιόνι που νικήθηκε η οποιεσδήποτε άλλες παράμετροι.

Έτσι μπορούμε να προχωρήσουμε σε ακόμα μια απλοποίηση σε σχέση με τον αριθμό markers που χρειάζεται να αναγνωρίζει το πρόγραμμα μας. Ο κάθε παίχτης έχει 8 στρατιώτες στην διάθεση του τους οποίους όμως σύμφωνα με τα παραπάνω μπορούμε να τους απεικονίσουμε με ένα μοντέλο όλους, και ένα τύπο marker. Έτσι στο frame μας θα έχουμε

κατά την αρχή του παιχνιδιού 8 ίδιους markers που θα ζωγραφίζουν 8 ίδια μοντέλα. Το ίδιο ισχύει για τον πύργο τον αξιωματικό και το άλογο, που για τον κάθε παίχτη υπάρχουν εις διπλούν και χρησιμοποιώντας ένα τύπο marker για το καθένα ικανοποιούμε τις απαιτήσεις του συστήματος μας. **Έτσι συνολικά και για τους 2 παίχτες μπορούμε να πούμε ότι έχουμε 12 markers που ανταποκρίνονται σε 12 μοντέλα.** Το παραπάνω μπορεί να θεωρηθεί σαν μια από τις απαιτήσεις του συστήματος μας, και μάλιστα από τις πιο σημαντικές αφού με βάση αυτό επηρεάζονται πολλές από τις μεθοδολογίες που χρησιμοποιούμε αργότερα.

Στοχοποιώντας λίγο σε αυτό το σημείο τις παραπάνω προδιαγραφές. Το IMCR αναφέρεται κυρίως ως μέτρο σύγκρισης σε ολοκληρωμένα συστήματα που παρέχουν μεγάλο αριθμό markers όπως για παράδειγμα το ARtag που περιέχει 2046 έτοιμους markers. [Fia04] Το IMCR γενικά είναι άμεσα συνυφασμένο με το hamming distance που αναφέραμε παραπάνω και στην δική μας περίπτωση μπορούμε να εκμεταλλευτούμε τον αριθμό marker που έχουμε για να επιλέξουμε markers με το μεγαλύτερο hamming distance μεταξύ τους.

Το MMS δεν μας ενδιαφέρει επίσης τόσο πολύ καθώς στο σκάκι μας η κάμερα δεν θα είναι και σε μεγάλη απόσταση από τους markers. Στην δημοσίευση [OXM01] αναφέρεται πως το MMS εξαρτάται από τον τρόπο ουσιαστικά που δειγματοληπτείται ο αναγνωρισμένος marker. Αν για παράδειγμα κάνουμε μια δειγματοληψία 6x6 όπως στο ARtag τότε το μέγεθος πρέπει να είναι τουλάχιστον 6x6 pixels. Στην περίπτωση του ARtoolkit που η δειγματοληψία γίνεται σε έναν 16x16 πίνακα το ελάχιστο μέγεθος είναι 16x16 pixels. Βέβαια τα παραπάνω αναφέρονται μόνο στο εσωτερικό των εκάστοτε markers και όχι στο μαύρο περίβλημα τους. Σύμφωνα με την παραπάνω δημοσίευση στην περίπτωση 16x16, το ελάχιστο μέγεθος είναι 21.66 pixels. Αυτά πάντα σε θεωρητικό επίπεδο βέβαια καθώς το πιο ενδιαφέρον σημείο είναι ότι η θεωρία απέχει πάντα από την πράξη και σε ένα ανοιχτό σύστημα όπως αυτό που μελετούμε υπάρχουν πολλές επιπλέον παράμετροι.

Αυτό που μας ενδιαφέρει πάρα πολύ στην περίπτωση μας είναι η μερική επικάλυψη. Αυτή αντιμετωπίζεται με την σωστή επιλογή του εσωτερικού των markers. Ουσιαστικά η επανάληψη της πληροφορίας μπορεί να μας βοηθήσει έτσι ώστε πιθανώς μερική επικάλυψη να μην έχει αποτέλεσμα στην αλλοίωση δεδομένων. Ο θόρυβος επίσης μπορεί να μας επηρεάσει τα δεδομένα εισάγοντας pixel διαφορετικού χρώματος από το πραγματικό επηρεάζοντας την ορθότητα των δεδομένων.

Προγραμματιστικές επιλογές

Στην προηγούμενη ενότητα αναλύσαμε μερικά από τα στοιχεία και τα προβλήματα που παρουσιάζονται σε fiducial systems , ενώ κάναμε και κάποιες προτάσεις σχετικά με το

είδος των marker που θα χρησιμοποιήσουμε. Σε αυτήν την υποενότητα θα εξηγήσουμε κάποιες από τις προγραμματιστικές επιλογές του συγγραφέα, με μορφή παραδειγμάτων, προσπαθώντας να αναδείξουμε κάποιες "ξεχασμένες" από την πλειονότητα των προγραμματιστών επιλογές οι οποίες είναι άμεσα συνδεδεμένες με θέματα βελτιστοποίησης. Το κυρίαρχο θέμα είναι η assembly οι καταχωρητές MMX και SSE.

Στην ενότητα 3.4 κάναμε μια εισαγωγή στην αρχιτεκτονική x86 και τις δυνατότητες της. Υπάρχουν όμως πολλοί τρόποι να γράψεις assembly. Ο συγγραφέας επέλεξε την χρήση inline assembly ένα χαρακτηριστικό που υποστηρίζουν οι περισσότεροι σύγχρονοι compilers. Η inline assembly σου δίνει την δυνατότητα να εισάγεις μέσα σε κώδικα C κομμάτια τα οποία είναι γραμμένα εξ ολοκλήρου σε assembly. Υπάρχουν βέβαια πολλοί τρόποι για να συντάξεις assembly και στην συγκεκριμένη περίπτωση inline assembly. Οι 2 επικρατέστεροι τρόποι είναι το συντακτικό της AT&T και της intel. Προτιμήσαμε τον τρόπο σύνταξης της intel, και κατ επέκταση λόγω χρήσης του visual studio το συντακτικό του MASM (microsoft macro assembler). Ο κώδικας της inline assembly τρέχει φυσικά και σε linux. Ο συγγραφέας το δοκίμασε μέσω του intel C++ compiler περνώντας το όρισμα "-use-msasm" που μας επιτρέπει χρήση συντακτικού του MASM. Άλλες πιθανές επιλογές θα μπορούσαν να είναι ο GAS (ο assembler του gcc) ή ο HLA.

Αρχικός σχεδιασμός της μη βελτιστοποιημένης εφαρμογής noopt. Λειτουργία και χρήση της βιβλιοθήκης

Έχοντας κάνει προηγουμένως μια μικρή εισαγωγή στις δυνατότητες της βιβλιοθήκης ας δοκιμάσουμε λοιπόν τον σχεδιασμό της εφαρμογής **η οποία θα έχει σκοπό να παράγει το επιθυμητό αποτέλεσμα χωρίς όμως καθόλου βελτιστοποιήσεις.**

Setup μηχανήματος	
Λειτουργικό σύστημα:	**Microsoft Windows XP Proffesional Edition, SP2**
Κάμερα:	**Creative Webcam (WMD)**
Επεξεργαστής:	**Athlon XP 2500+ ,Barton (MMX, SSE, 3DNow!)**
Κάρτα γραφικών:	**Nvidia 6600GT (υποστηρίζει OpenGL 2.0)**

Camera calibration

Υπάρχουν 2 τρόποι να γίνει το camera calibration σύμφωνα και με το: Inside ARtoolkit tutorial του Hirokazu Kato. Το 2 step method με χρήση του calib_dist και του calib_cparam (το οποίο δεν παρουσιάζουμε) και αυτό που χρησιμοποιήσαμε, το 1 step method με χρήση του calib_camera2. Ο λόγος που καταφεύγουμε σε αυτό το βήμα και δεν περνάμε απ'ευθείας στο pattern training είναι προφανής για τους οξυδερκείς αναγνώστες. Και φαίνεται καθαρά στην παρακάτω εικόνα:

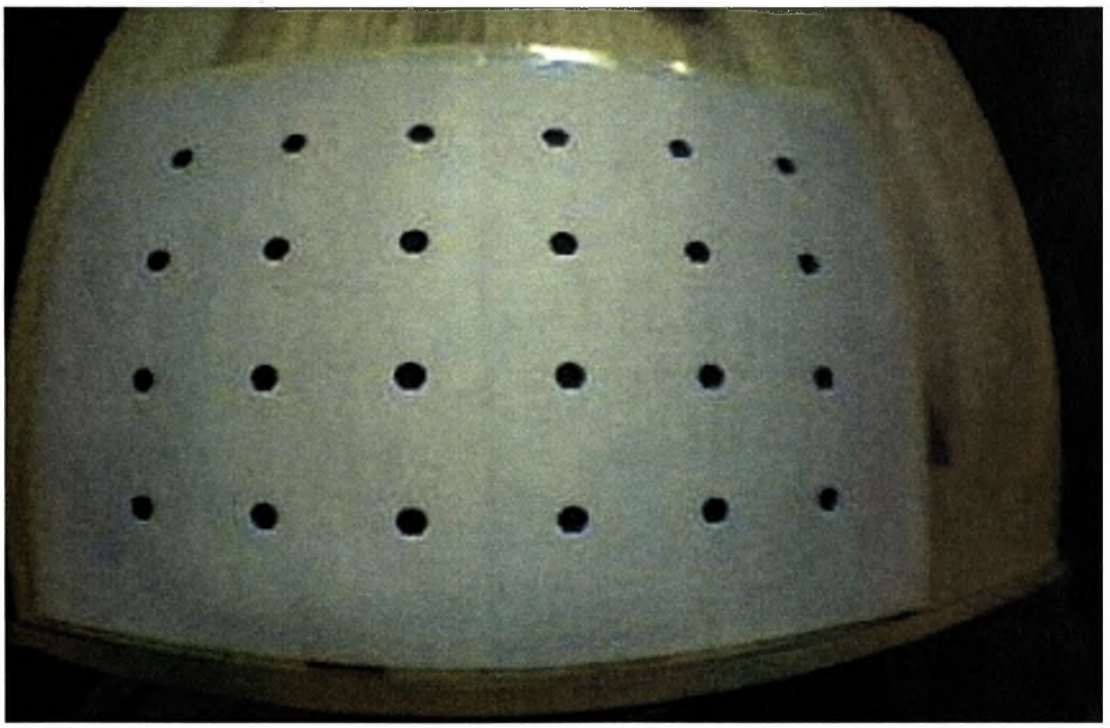


Image distortion: Inside ARtoolkit, Hirokazu Kato

Και οι σχέσεις μεταξύ ideal και observed screen coordinates (distorted) φαίνονται παρακάτω:

$$d^2 = (x_1 - x_0)^2 + (y_1 - y_0)^2$$

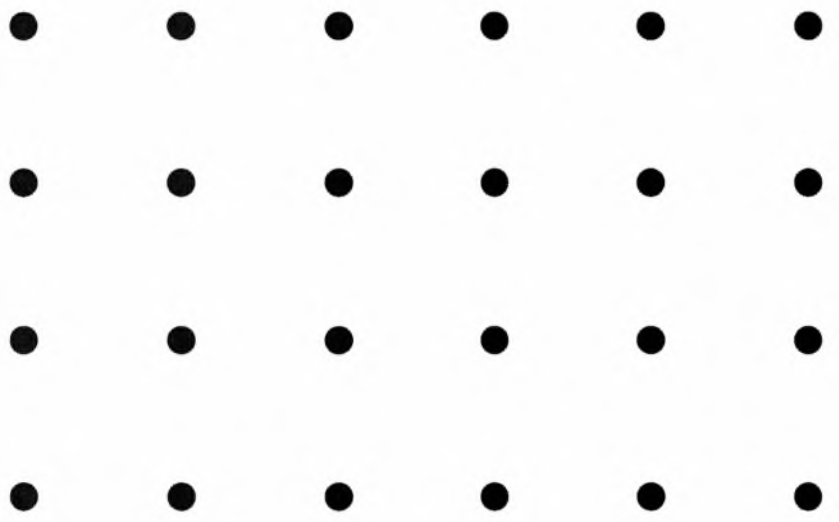
$$p = \{1 - fd^2\}$$

$$x_0 = p(x_1 - x_0) + x_0$$

$$y_0 = p(y_1 - y_0) + y_0$$

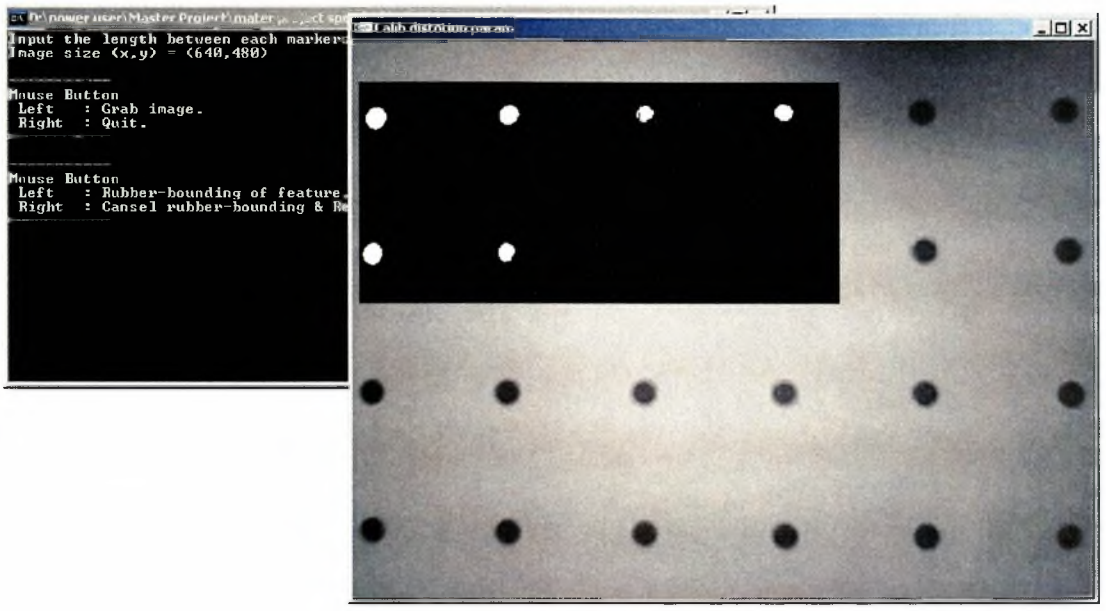
Όπου (x_0, y_0) οι κεντρικές συντεταγμένες της παραμόρφωσης και f ο συντελεστής παραμόρφωσης

Έτσι τελικά τρέχουμε το εκτελέσιμο calib_camera2.exe και αρχικά μπορεί να μας ζητηθεί ανάλογα με την έκδοση του ARtoolkit που έχουμε η εισαγωγή κάποιων παραμέτρων όπως η πραγματική απόσταση μεταξύ των dots του σχήματος (σχήμα 1) η ανάλυση βίντεο που θα κάνει capture η κάμερα η χρωματική απόδοση RGB κτλ.



Σχήμα 1: calib_dist. Η απόσταση που είχαμε ανάμεσα στα dots στην περίπτωση μας ήταν 3,3 cm με εκτύπωση σε χαρτί Α4. Πηγή: ARtoolkit 2.65.

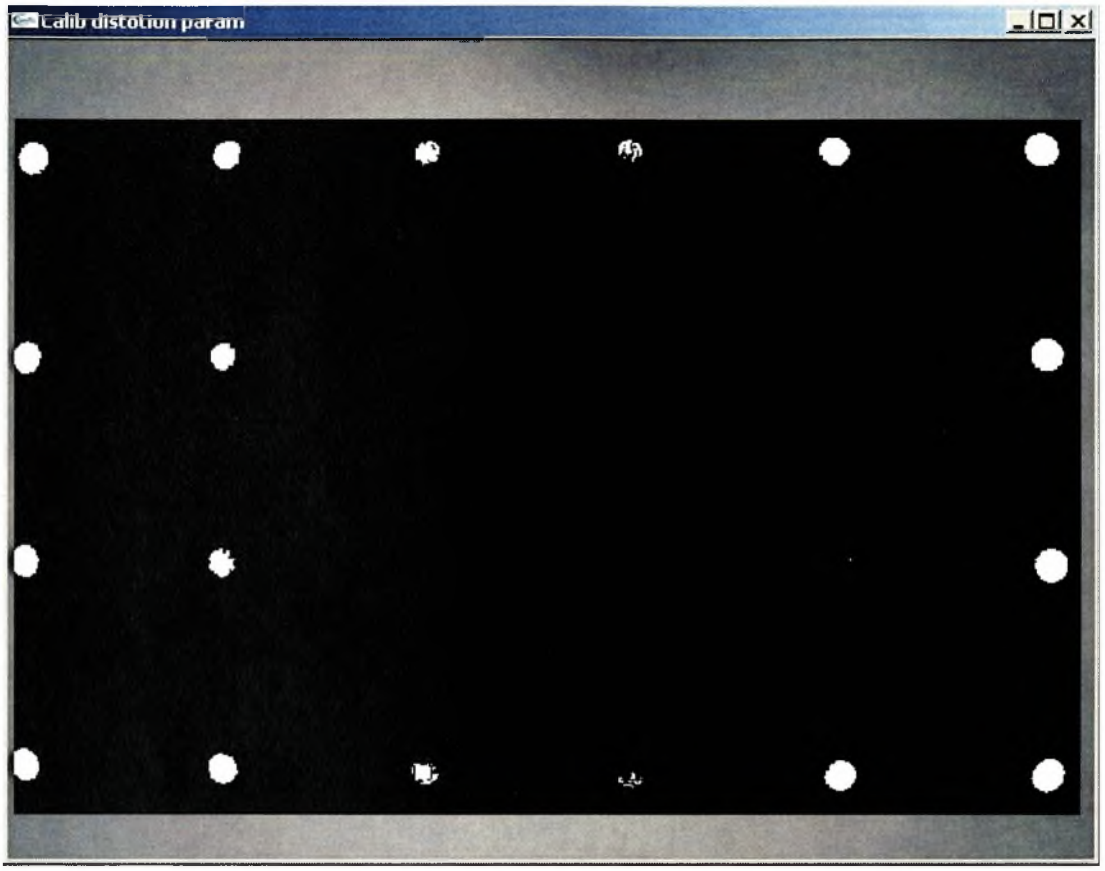
Αφού λοιπόν το σχήμα 1 εκτυπωθεί το βάζουμε μπροστά στην κάμερα και με διάφορες επιλογές που παρέχονται κάνουμε grab ένα frame και επιλέγουμε με το mouse τα dots ένα προς ένα από αριστερά προς τα δεξιά και από πάνω προς τα κάτω. Αξίζει να παρατηρήσουμε το παρακάτω σχήμα όπου το μαύρο παραλληλόγραμμο έχει επιλεχθεί με το mouse. Παρατηρήστε πως απουσιάζουν τα dots στις συντεταγμένες [1,2] [1,3] καθώς λόγω του φωτισμού το ARtoolkit απέτυχε να τα αναγνωρίσει.



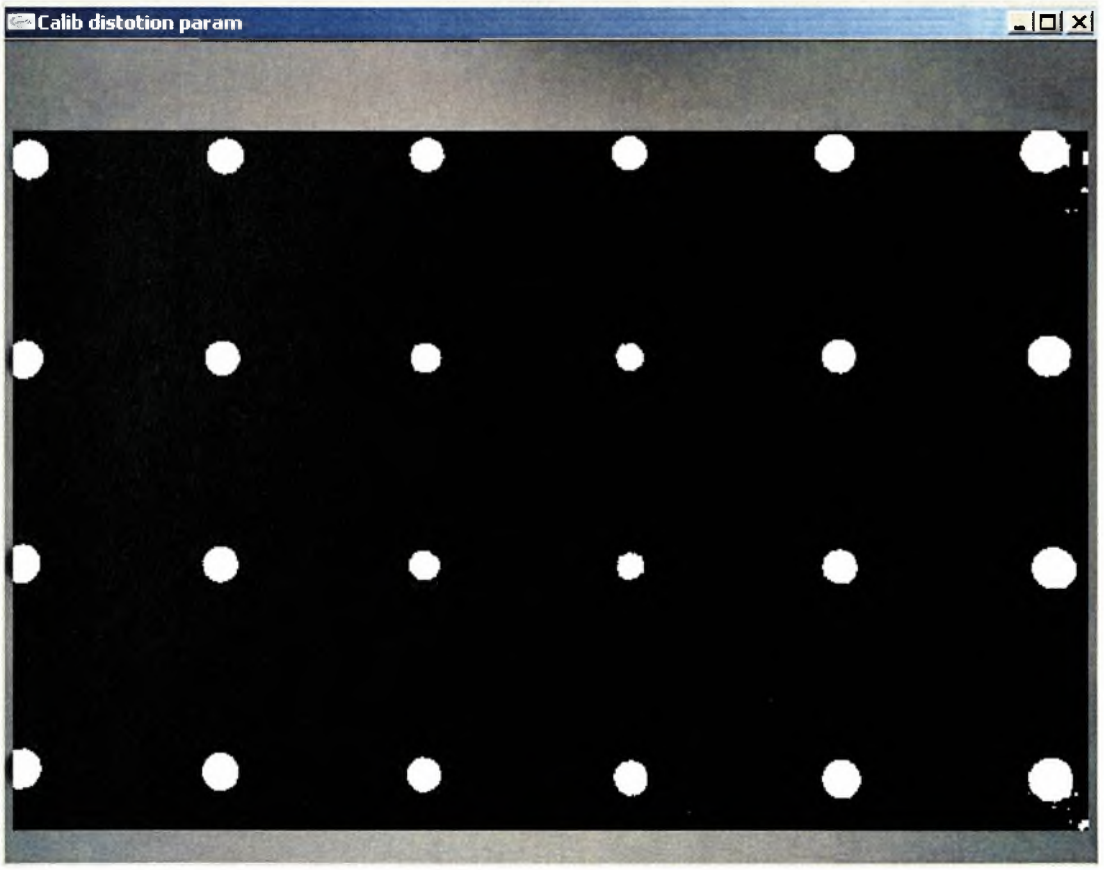
Σχήμα 2: camera calibration 1 step method με παρουσίαση τρόπου επιλογής των dots.

Το παραπάνω σχήμα παρουσιάζει και κάτι που έχει αναφερθεί πρωτύτερα σχετικά με τον τρόπο λειτουργίας της βιβλιοθήκης. **Το ARtoolkit πάντα κάνει thresholding σε κάθε frame που λαμβάνει από την κάμερα και μετατρέπει το τρέχον frame σε binary image**. Μετά την διαδικασία αυτή λοιπόν παράγεται ένα αρχείο που *.dat που περιέχει σε binary μορφή το camera distortion.

Εφόσον προηγουμένως μιλήσαμε για τον ρόλο του Threshold αξίζει να παρουσιάσουμε την σπουδαιότητα του λίγο πιο λεπτομερώς σε σχέση με το camera calibration και κατ επέκταση όλη την βιβλιοθήκη καθώς η συνάρτηση που χρησιμοποιείται στο camera calibration για frame grabbing είναι η ίδια ακριβώς με αυτήν που χρησιμοποιείται και στα τελικά applications. Στα σχήματα 3 και 4 παρουσιάζεται 2 frame με ακριβώς τις ίδιες συνθήκες φωτισμού με μόνη διαφορά την τιμή του threshold. Οι διαφορές είναι εμφανείς καθώς σε περίπτωση που τα dots ήταν patterns στο $1^{ο}$ σχήμα δεν θα αναγνωρίζονταν ενώ στο $2^{ο}$ θα είχαμε ενός είδους «θόρυβο» στα δεδομένα μας. Παρατηρήστε τα dots στις θέσεις [0,5] [1,5] [2,5] [3,5].

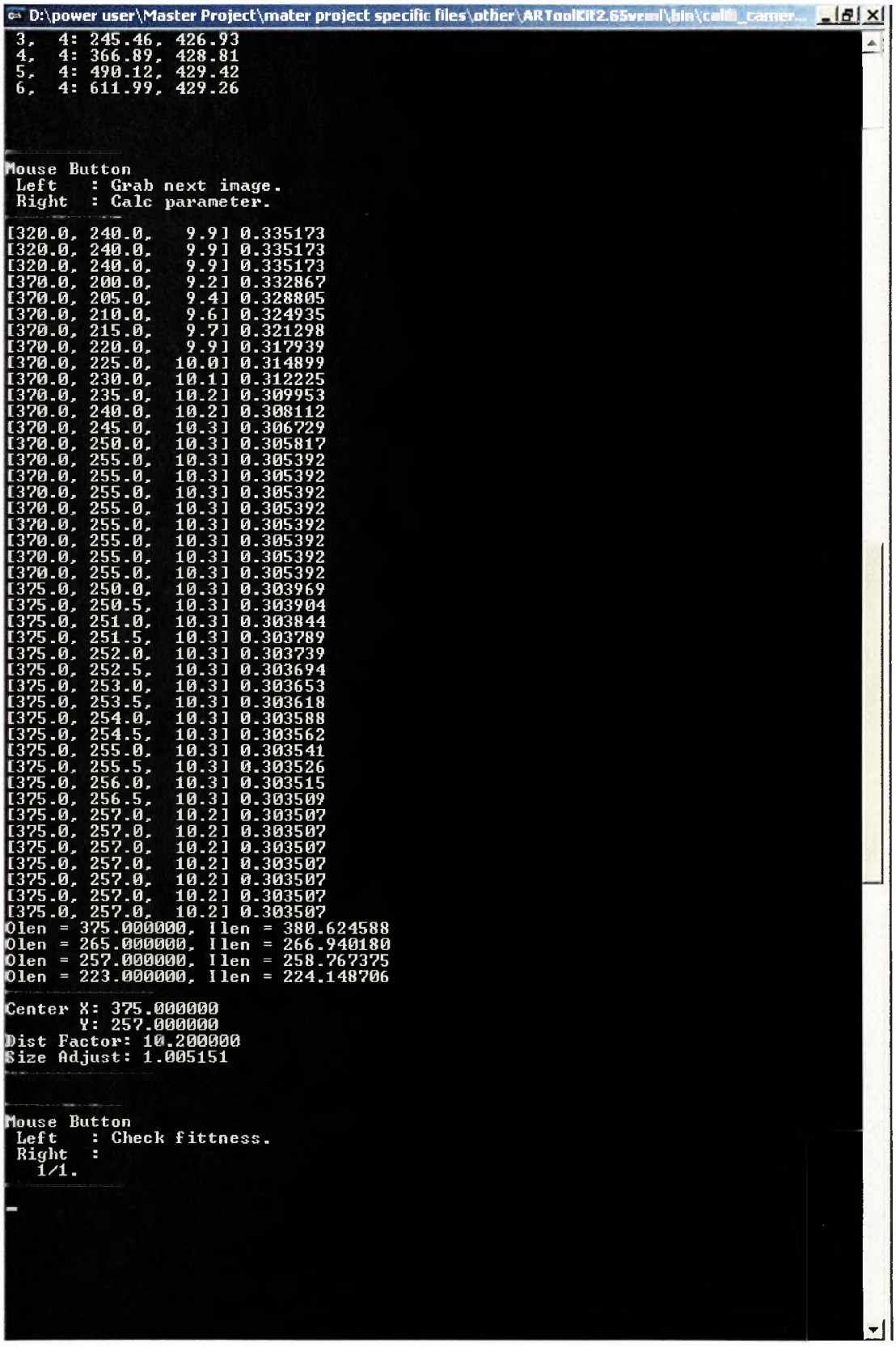


Σχήμα 3: Camera calib Threshold value = 200



Σχήμα 4: Camera calib Threshold value = 150

Τελειώνοντας με το camera calibration στο σχήμα 5 παρουσιάζεται η διαδικασία υπολογισμού των σχέσεων μεταξύ ideal και observed screen coordinates.

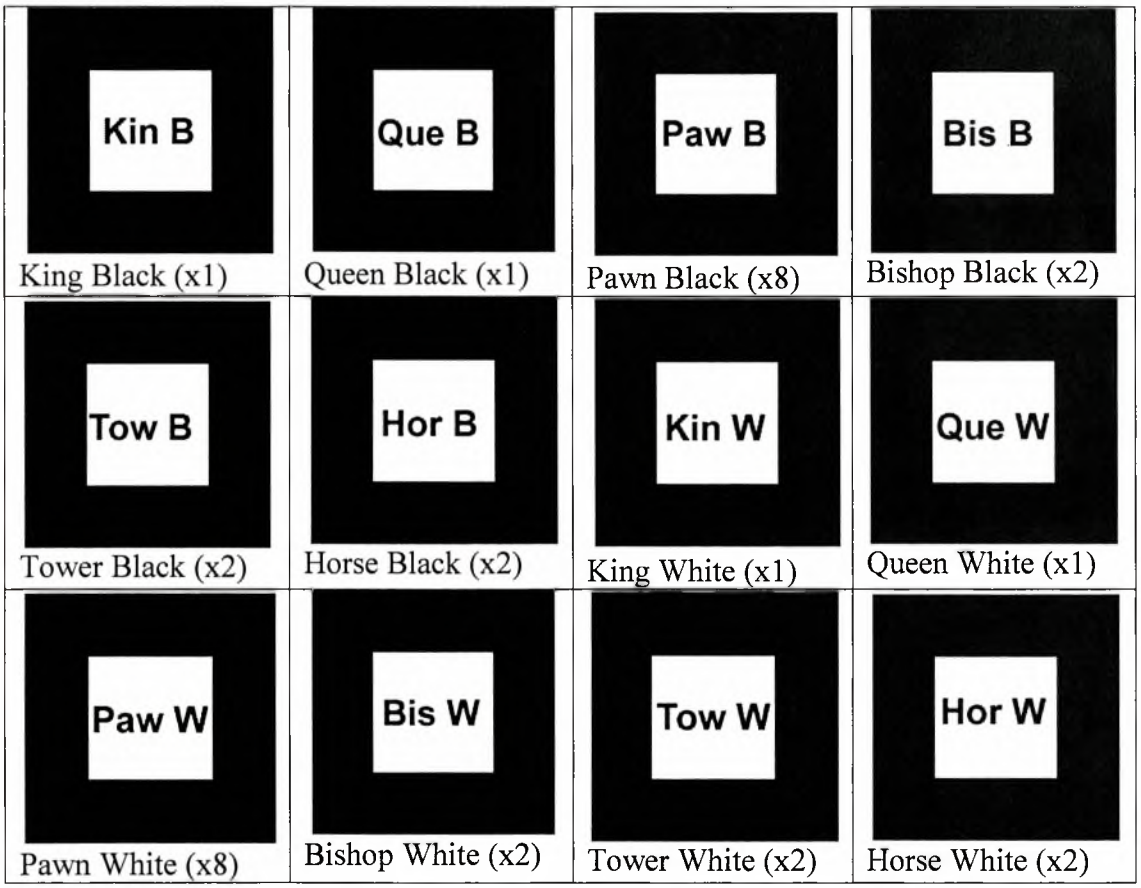


Σχήμα 5: parameter calculation κατά το camera calibration.

Pattern Training

Στο συγκεκριμένο παράδειγμά μας χρειάστηκε να κάνουμε train συνολικά 12 patterns. Τον ρόλο του pattern training τον εξηγήσαμε διεξοδικά σε προηγούμενη παράγραφο από θεωρητικής άποψης οπότε μένει να δούμε πρακτικά τον τρόπο χρήσης του καθώς και τις διάφορες ιδιαιτερότητές του. Στο συγκεκριμένο παράδειγμα χρειαστήκαμε συνολικά 12 **τύπους** patterns:

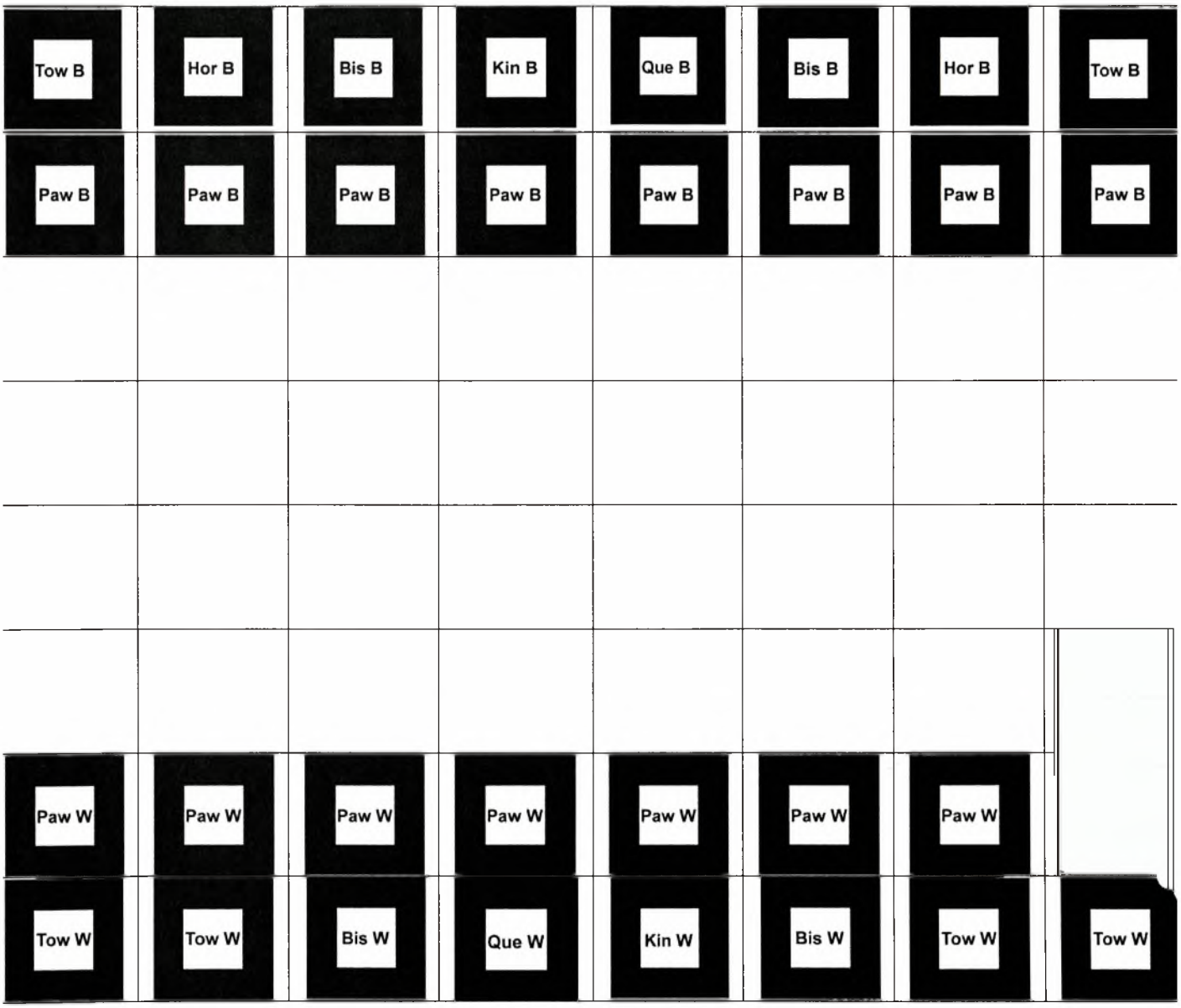
- 2 για τους βασιλιάδες (1 για άσπρα,1 για μαύρα)
- 2 για τις βασίλισσες (1 για άσπρα,1 για μαύρα)
- 2 για τους αξιωματικούς (1 για άσπρα,1 για μαύρα)
- 2 για τα άλογα (1 για άσπρα,1 για μαύρα)
- 2 για τους πύργους (1 για άσπρα,1 για μαύρα)
- 2 για τους στρατιώτες (1 για άσπρα,1 για μαύρα)



Σχήμα 6: Όλα τα patterns που χρησιμοποιήθηκαν σε παρένθεση ο αριθμός αντιγράφων τους.

Όπως όμως είναι γνωστό ένα παιχνίδι σκάκι περιέχει συνολικά 32 πιόνια και προφανώς στην πραγματικότητα θα έχουμε παραπάνω εκτυπωμένα patterns στον «πραγματικό κόσμο» από τα προαναφερόμενα. Τα αρχικά αυτά patterns είναι φτιαγμένα με τον τρόπο που περιγράφει το ARtoolkit δηλαδή δίνουμε απλά σημασία στο μονοσήμαντο προσδιορισμό του orientation κάθε marker μέσω του pattern τους.

Οπότε σύμφωνα με τα παραπάνω στο πρόγραμμά μας η σκακιέρα θα παρουσιάζεται ως εξής:



Σχήμα 7: Η ιδεατή σκακιέρα μας. Παρατηρήστε την πολυπλοκότητα της σε σχέση με όλα τα προηγούμενα παραδείγματα που παρουσιάστηκαν.

Έτσι λοιπόν με την βοήθεια του υποπρογράμματος mk_patt κάνουμε train τα 12 patterns μας. **Εδώ αξίζει να αναφερθεί μια πρωτοτυπία καθώς τα patterns έγιναν train καθώς η κάμερα έκανε capture frames από την οθόνη απ'ευθείας και όχι από το εκτυπωμένο pattern.** Αυτό γιατί τα προβλήματα φωτισμού εμπόδιζαν σε ένα βαθμό ενώ σε αντίθεση η οθόνη ακτινοβολεί φως ομοιόμορφα (το frame rate που έκανε capture η κάμερα στα 15Hz σαφώς μικρότερο από τα 60 της TFT δεν δημιούργησε κανένα πρόβλημα flickering =)).

Αρχικά εισάγουμε το αρχείο που δημιουργήσαμε στο προηγούμενο βήμα το οποίο περιλαμβάνει το camera distortion όπως υπολογίστηκε. Ύστερα ανάλογα αν έχουμε κάνει μεταβολές η όχι στον κώδικα του mk_patt η όχι εισάγουμε διάφορα στοιχεία όπως πριν για το είδος του video feed.

Παρακάτω παρουσιάζεται το περιεχόμενο ενός τέτοιου αρχείου και συγκεκριμένα του patt.hiro που περιλαμβάνεται στην συγκεκριμένη έκδοση του ARtoolkit. Παρουσιάζεται μόνο κομμάτι από το πρώτο orientation του pattern παραλείποντας για ευκολία και μερικές ενδιάμεσες γραμμές.

234	235	240	233	240	234	240	235	240	237	240	238	240	240	240	232
229	240	240	240	240	240	240	240	240	240	240	240	240	240	240	238

.....

225	156	240	240	182	212	225	180	240	240	240	240	240	240	240	240
150	116	238	228	66	205	115	151	238	236	225	240	240	180	226	240
156	84	186	211	47	184	109	170	200	92	30	240	120	50	53	216
147	83	51	73	50	184	106	110	148	17	151	150	45	217	186	85
127	98	219	219	58	179	109	101	128	107	237	125	155	240	163	72
155	86	240	240	76	201	85	108	121	95	232	137	51	118	153	155
149	189	240	240	98	220	141	154	206	178	235	230	152	77	175	209

Σχήμα 6: ανάλυση περιεχομένων ενός trained pattern.

Στις πρώτες 2 γραμμές μπορούμε να δούμε σχηματικά τι απεικονίζει κάθε τιμή του πίνακα μας. Στο πρώτο αυτό orientation μπορούμε να παρατηρήσουμε ότι ανά τρείς οι τιμές παριστάνουν την χρωματική απόδοση ενός pixel με τις τιμές RGB στην σειρά. Επομένως οι αριθμοί 234 235 240 ανταποκρίνονται στο 1^{ο} pixel οι τιμές 233 240 234 στο 2ο pixel κ.ο.κ. Συνολικά κάθε γραμμή έχει 16 τέτοιες τιμές ενώ κάθε orientation έχει 48 τέτοιες στήλες και συνολικά έχουμε: 48*16=768 και 768/3=256 pixelτο οποίο ανταποκρίνεται και στην πραγματικότητα καθώς το εσωτερικό κάθε Pattern δειγματοληπτείται με έναν πίνακα 16*16=256 samples .

Σαν τελευταίο εδώ πρέπει να αναφέρουμε πως επειδή το παράδειγμα μας χρειάζεται να αναγνωρίζει πολλαπλά patterns, φτιάχνουμε ένα αρχείο που περιέχει λίστα με τα trained patterns το οποίο στο παράδειγμα μας το ονομάσαμε multichess.dat .

```
#the number of patterns to be recognized
12

#pattern 1
bisB
patterns/bisB.patt
80.0
0.0 0.0
...
```

Σχήμα 7: Στιγμιότυπο αρχείου multichess.dat.

Με # είναι τα σχόλια. Αρχικά δηλώνουμε τον αριθμό των Patterns που θα περάσουμε στο πρόγραμμα (12) και μετά το όνομα του (bisB) και το σημείο που βρίσκεται το αρχείο σε σχέση με το εκτελέσιμο (patterns/bisB.patt).

Ο κώδικας του noopt

Ο κώδικας του noopt είναι μια απλή χρήση της βιβλιοθήκης ARtoolkit για πολλαπλά patterns που μοιάζει εν μέρει με το παράδειγμα που περιλαμβάνεται στο ARtoolkit

"LoadMultiple" ο κώδικας κάνει χρήση της βιβλιοθήκης gsub και όχι gsub_lite ωστόσο υπάρχουν αρκετές διαφορές σε σχέση με τα ήδη υπάρχοντα παραδείγματα.

Α)Γίνεται χρήση της βιβλιοθήκης glm για την φόρτωση των μοντέλων του σκακιού που έχουμε δημιουργήσει ΒΛΕΠΕ ΤΑΔΕ. Η βιβλιοθήκη είναι υπεύθυνη για την δημιουργία display lists που βοηθάνε στην μείωση του χρόνου απεικόνισης των μοντέλων. Για να αποφύγουμε να φορτώσουμε 2 εκδόσεις του ίδιου μοντέλου μια άσπρη και μια μαύρη, όπως επίσης και να αποφύγουμε φόρτωση materials/textures κάνουμε ένα απλό τρικ με τον φωτισμό έτσι ώστε να φαίνεται μια διαφορά ανάμεσα στα άσπρα και στα μαύρα πιόνια.

Β)Γίνεται χρήση των συναρτήσεων που είναι στο timer.h για τον υπολογισμό των χρόνων. Το timer.h περιλαμβάνει υλοποιήσεις συναρτήσεων με την βοήθεια της QueryPerformanceCounter που είναι πολύ πιο ακριβής από την _ftime που χρησιμοποιεί το ARtoolkit. Η QueryPerformanceCounter είναι βασισμένη στο μέτρημα του ρολογιού του επεξεργαστή. Μικρά προβλήματα ενδέχεται να υπάρξουν σε περίπτωση που ο επεξεργαστής μπορεί να αλλάζει συχνότητα για εξοικονόμηση ενέργειας σύμφωνα με την intel. Οι μέθοδοι στο times.h παράγουν σε κάθε εκτέλεση το αρχείο times.txt που περιλαμβάνει πληροφορίες σχετικά με την εκτέλεση του προγράμματος. Θα σταθούμε μόνο σε αυτά που μας ενδιαφέρουν για το noopt καθώς λεπτομερέστερη ανάλυση δίνεται στο allopt. Στην περίπτωση του noopt το πιο ενδιαφέρον χαρακτηριστικό είναι το «median inter marker confusion rate». Στις σχετικές δημοσιεύσεις υπάρχουν αρκετοί τρόποι μέτρησης αυτού του λάθους αλλά οι περισσότεροι προσανατολίζονται σε γενικές μεθόδους. Εμείς στην δική μας περίπτωση και κατεπέκταση στην δεύτερη εκτέλεση του noopt το υπολογίζουμε σαν τον αριθμό των marker που βρέθηκαν παραπάνω από μία φορά σε ένα frame. Αυτό γιατί εκ των προτέρων έχουμε την γνώση ότι στο $2^{ο}$ παράδειγμα υπάρχει μόνο ένας marker από κάθε είδος, οπότε διπλές εμφανίσεις κάποιου marker ανάγονται αυτόματα σε inter marker confusion.

Ο κώδικας του noopt περιλαμβάνει κλήσεις των συναρτήσεων (του timer.h) timer_start και timer_end με τις εξής παραδοχές:

Initialization: Μετράται από την στιγμή που επιλέξει ο χρήστης τις ρυθμίσεις για την κάμερα μέχρι το τελείωμα του φορτώματος των τρισδιάστατων μοντέλων.

Περιλαμβάνει επίσης :την εύρεση του μεγέθους του παραθύρου, το φόρτωμα των παραμέτρων της κάμερας και το φόρτωμα των patterns στη μνήμη

Marker_coords_Tranformation: περιλαμβάνει την κλήση της arDetectMarker για την εύρεση όλων των marker στο frame

Match_Patterns: περιλαμβάνει την σύγκριση με όλους τους marker που έχουμε βρει στο frame με ένα από τα pre trained patterns. (κάθε φορά επιλέγεται να κάνουμε sampling ψευδοτυχαία σε ένα).

Περιλαμβάνει επίσης : Το ζωγράφισμα ενός πράσινου τετραγώνου στους marker που ταυτοποιήθηκαν.

Marker_coords_Tranformation: Περιλαμβάνει την κλήση είτε της arGetTransMat είτε της arGetTransMatCont για ένα object του οποίου η ύπαρξη έχει ταυτοποιηθεί. Οι 2 αυτές συναρτήσεις βοηθούν στο pose και position estimation του τρέχοντος marker στο σύστημα συντεταγμένων.

3d_Object_Rendering: Περιλαμβάνει το ζωγράφισμα των αντικειμένων με τις αντίστοιχες αλλαγές στο φωτισμό και στο opengl state.

Αποτελέσματα – feedback

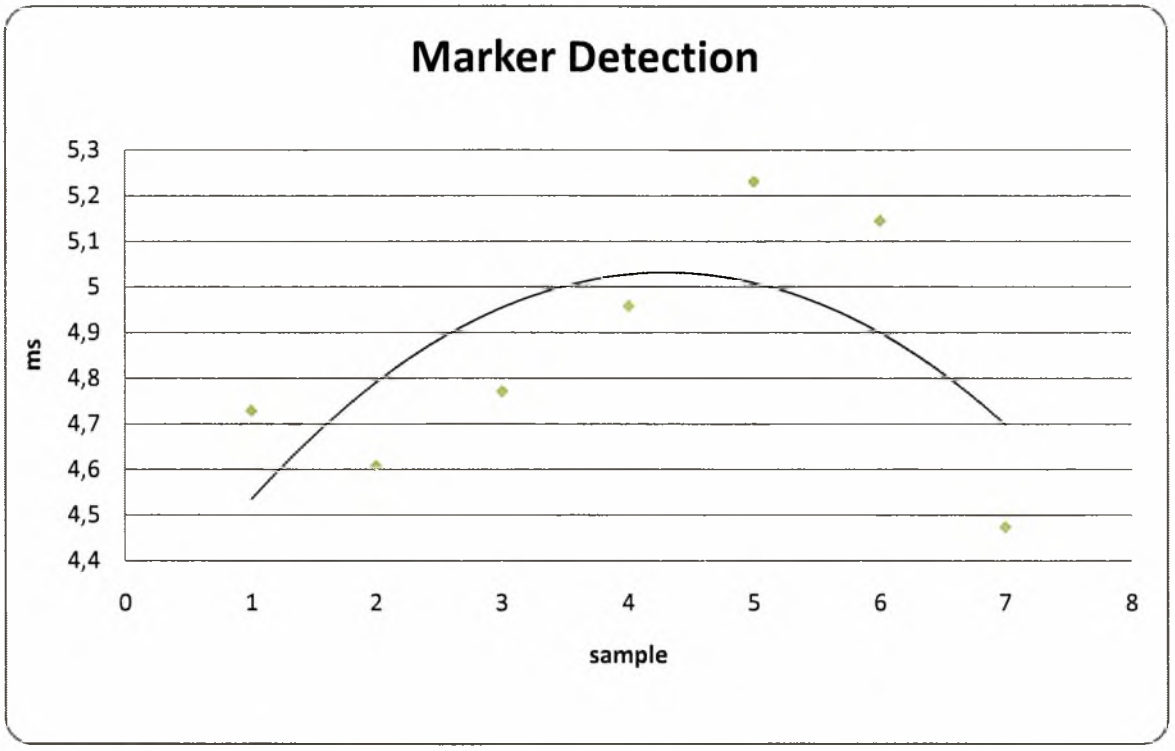
Παρακάτω βλέπουμε τα πειραματικά αποτελέσματα σε 3 διαφορετικές περιπτώσεις εκτέλεσης του προγράμματος:

noopt test-run 1 15sec

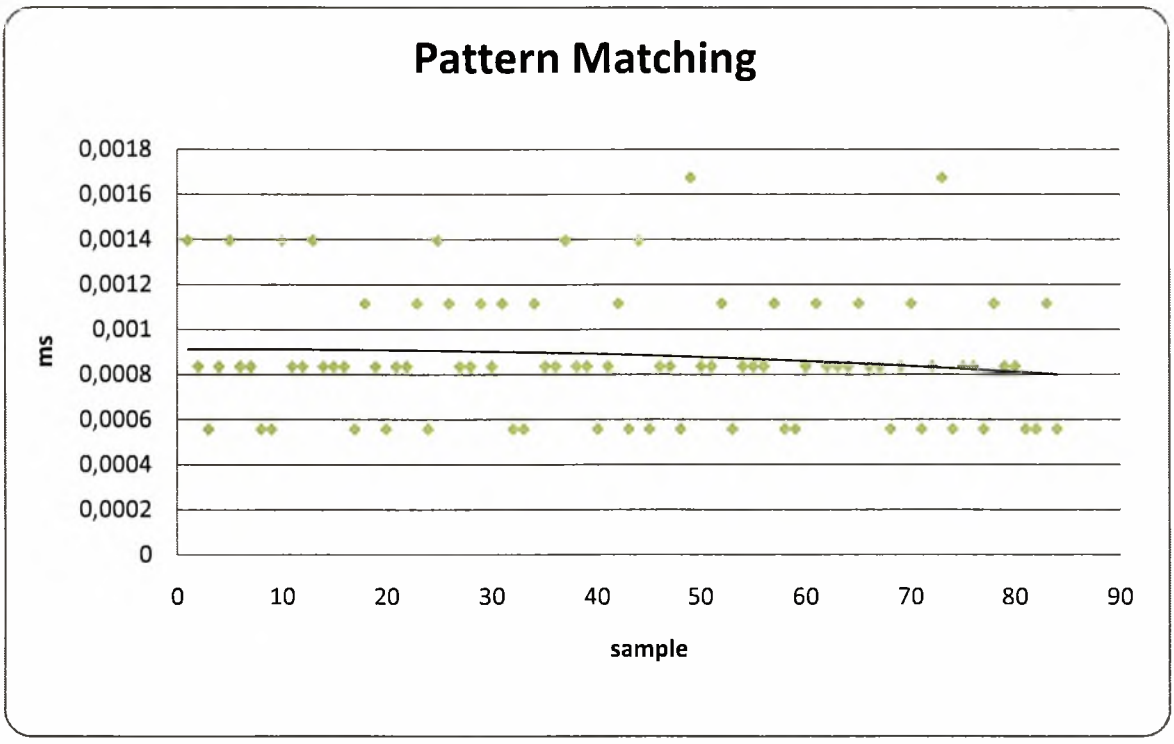
Ρυθμίσεις	
χρωματική απόχρωση	RGB
ανάλυση	640x480
αριθμός marker στο frame	0

Initialization time: 3122,044922 ms

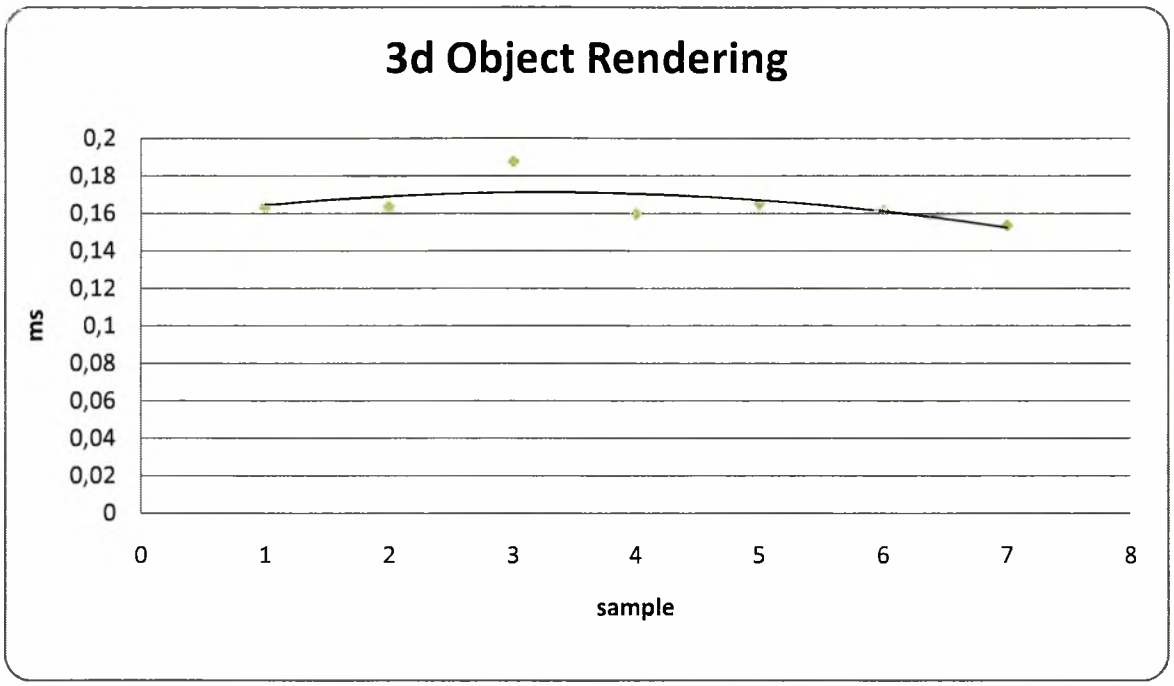
Marker Detection Times:



Match Patterns To Trained Patterns:



3D Object Rendering:



----------Collective_Stats----------		
Marker_Detection_median_time=	**4,845588**	**in_7_times**
Match_Patterns_median_**time=**	**0,000878**	**in_84_times**
Marker_**Coords_Transformation_median_time_cannot_be _computed_because_no_execution_time_given**		
3d_object_Rendering_median_time=	**0,165025**	**in_7_times**
No_**markers_detected**		
No_**markers_matched**		

Επεξηγήσεις

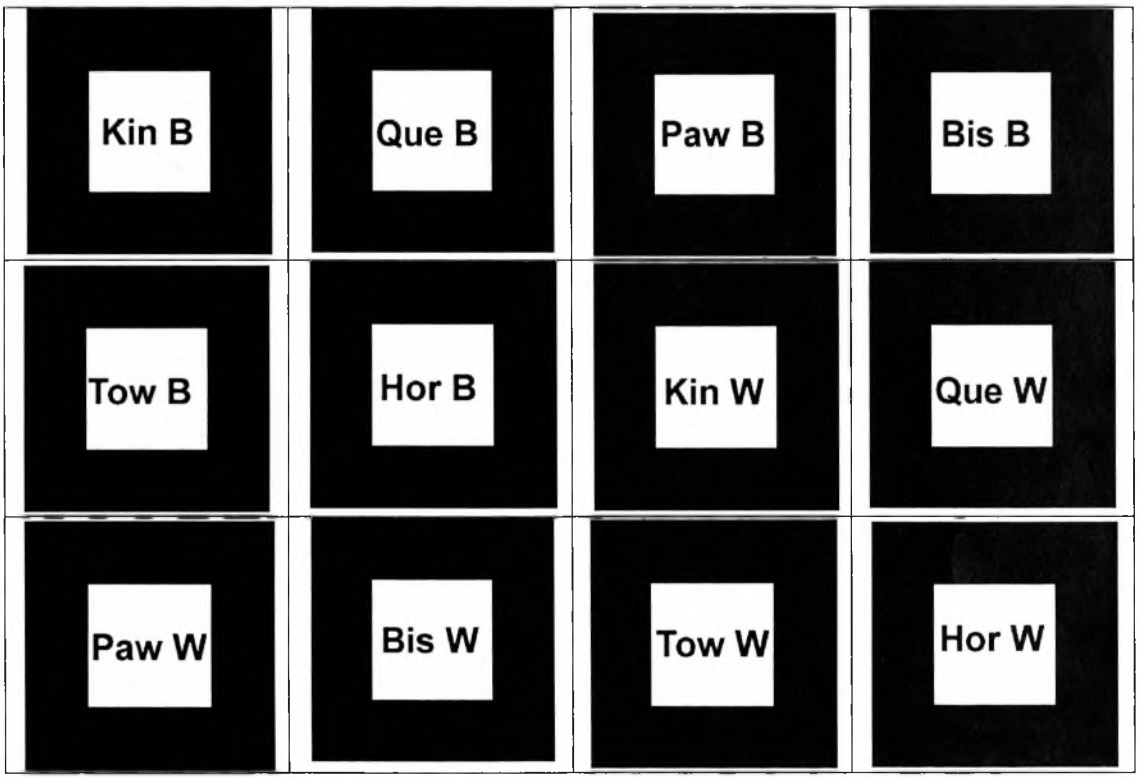
Οι γραμμές που φαίνονται στα σχήματα είναι πολυωνυμικές προσεγγίσεις με πολυώνυμα $2^{ου}$ βαθμού. Οι συναρτήσεις για μετατροπή συντεταγμένων δεν εκτελέστηκαν καμία φορά γι΄αυτό και δεν επιστρέφουν αποτελέσματα. Δεν βρέθηκαν καθόλου markers οπότε δεν ταιριάχτηκαν και στα pre trained patterns. Μπορεί επίσης να φαίνεται περίεργο που έχουμε τιμές στο match patterns with trained patterns και 3d object rendering αλλά αυτό είναι εύκολα αιτιολογήσιμο καθώς και στα 2 σημεία το πρόγραμμα κάνει ελέγχους είτε βρει είτε δεν βρει Patterns στο τρέχον frame καθώς

είναι αδύνατον να ξέρει εκ των προτέρων τις αλλαγές στο περιβάλλον. Αυτό θα μας απασχολήσει λεπτομερώς αργότερα.

noopt test-run 2 15sec

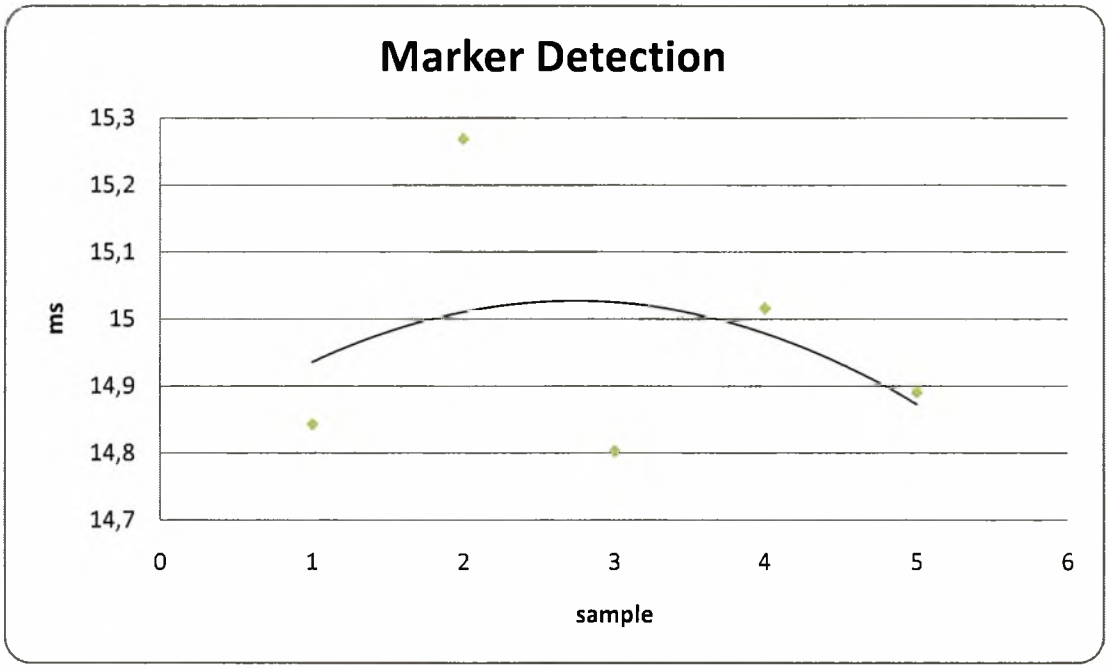
Ρυθμίσεις	
χρωματική απόχρωση	RGB
ανάλυση	640x480
αριθμός marker στο frame	12

Σχηματική αναπαράσταση frame:

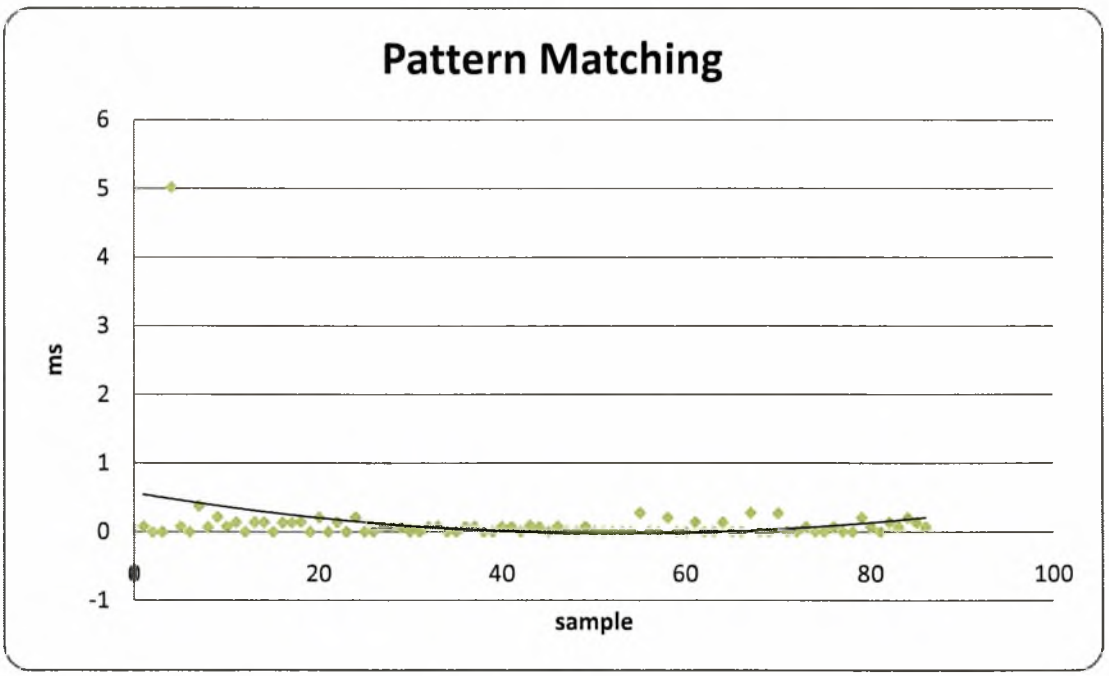


Initialization time: 3107,421143 ms

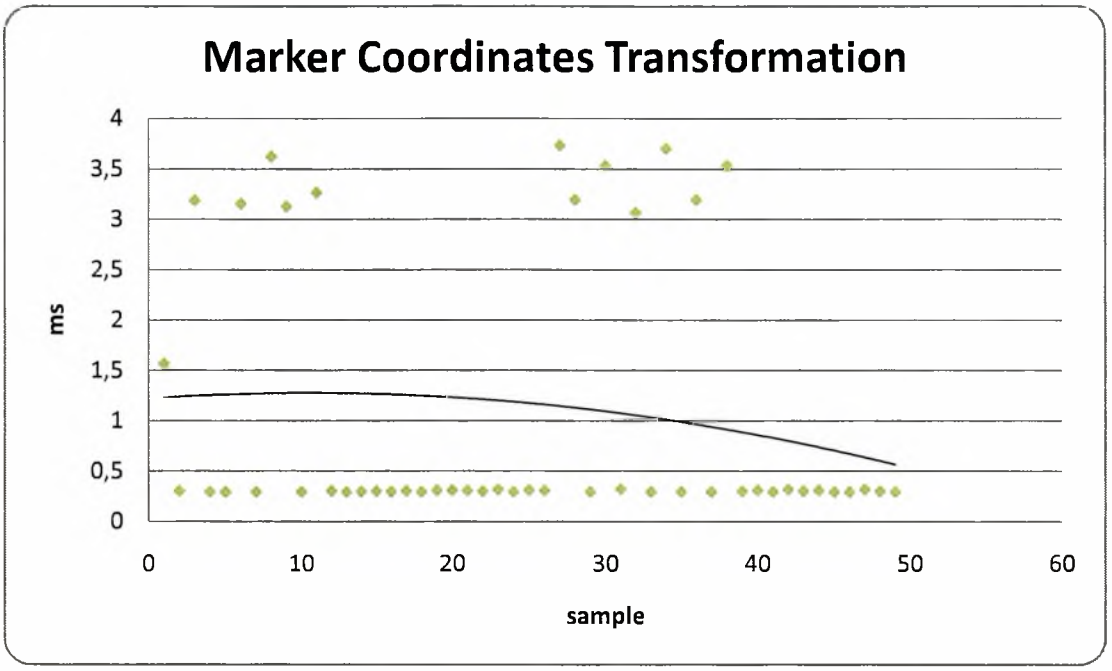
Marker Detection Times:



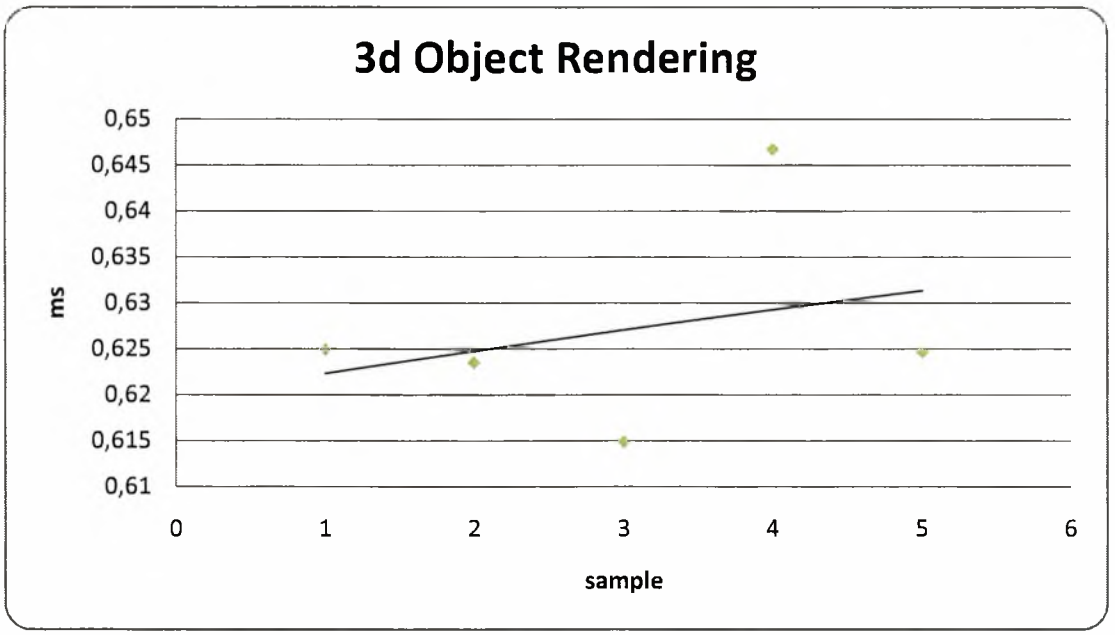
Match Patterns To Trained Patterns:



Marker Coordinates Transformation:



3D Object Rendering:



----------Collective **Stats**----------		
Marker_Detection_median_time=	**14,964639**	**in_5_times**
Match_Patterns_median_time=	**0,125123**	**in_86_times**
Marker_Coords_Transformation_median_time=	**1,079820**	**in_49_times**
3d_object_Rendering_median_time=	**0,626951**	**in_5_times**
Median_marker_num=	**10,107143**	
Median_marker_matched=	**5,821429**	
Median inter_marker_confusion_rate=	**4,285714**	

noopt test-run 3 15sec

-not run

Η Τρίτη εκτέλεση του noopt που προορίζονταν να γίνει με παρόντα όλα τα 24 patterns δεν εκτελέστηκε λόγω των προβλημάτων που παρουσιάστηκαν στο noopt test-run 2.

author: Sotiris Athanasiou

organisation: UTH-CCED

version: final

Info: Samples taken approx every 15 frames in case no marker is present to frame

----------Time Iterations----------

Performance Counter Frequency(ticks per second)=3579545

0 Initialization= 3122.044922 ms

1 Marker_Detection_(arDetectMarker)= 4.729651 ms

2 Match_Patterns_(inside_detected_Markers)_to_trained_patterns= 0.001397 ms

3 Match_Patterns_(inside_detected_Markers)_to_trained_patterns= 0.000838 ms

4 Match_Patterns_(inside_detected_Markers)_to_trained_patterns= 0.000559 ms

5 Match_Patterns_(inside_detected_Markers)_to_trained_patterns= 0.000838 ms

6 Match_Patterns_(inside_detected_Markers)_to_trained_patterns= 0.001397 ms

7 Match_Patterns_(inside_detected_Markers)_to_trained_patterns= 0.000838 ms

8 Match_Patterns_(inside_detected_Markers)_to_trained_patterns= 0.000838 ms

9 Match_Patterns_(inside_detected_Markers)_to_trained_patterns= 0.000559 ms

10 Match_Patterns_(inside_detected_Markers)_to_trained_patterns= 0.000559 ms

11 Match_Patterns_(inside_detected_Markers)_to_trained_patterns= 0.001397 ms

12 Match_Patterns_(inside_detected_Markers)_to_trained_patterns= 0.000838 ms

13 Match_Patterns_(inside_detected_Markers)_to_trained_patterns= 0.000838 ms

14 3d_Object_Rendering= 0.163429 ms

.....

92 Match_Patterns_(inside_detected_Markers)_to_trained_patterns= 0.000838 ms

93 Match_Patterns_(inside_detected_Markers)_to_trained_patterns= 0.000838 ms

94 Match_Patterns_(inside_detected_Markers)_to_trained_patterns= 0.000559 ms

95 Match_Patterns_(inside_detected_Markers)_to_trained_patterns= 0.000559 ms

96 Match_Patterns_(inside_detected_Markers)_to_trained_patterns= 0.001117 ms

97 Match_Patterns_(inside_detected_Markers)_to_trained_patterns= 0.000559 ms

98 3d_Object_Rendering= 0.153651 ms

----------Collective Stats----------

Marker Detection median time=4.845588 in 7 times

Match Patterns median time=0.000878 in 84 times

Marker Coords Transformation median time cannot be computed because no execution time given

3d object Rendering median time=0.165025 in 7 times

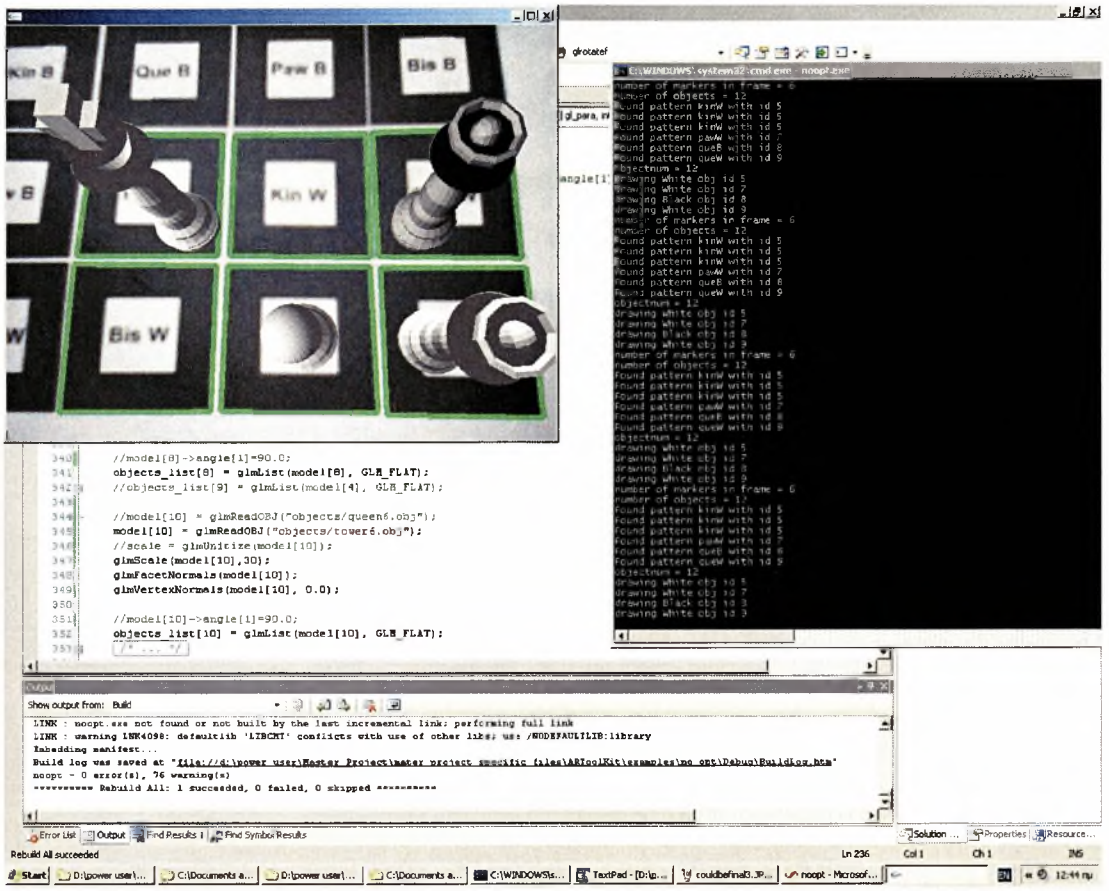
no markers detected

no markers matched

File generated Wed Jul 09 19:51:30.265 2008

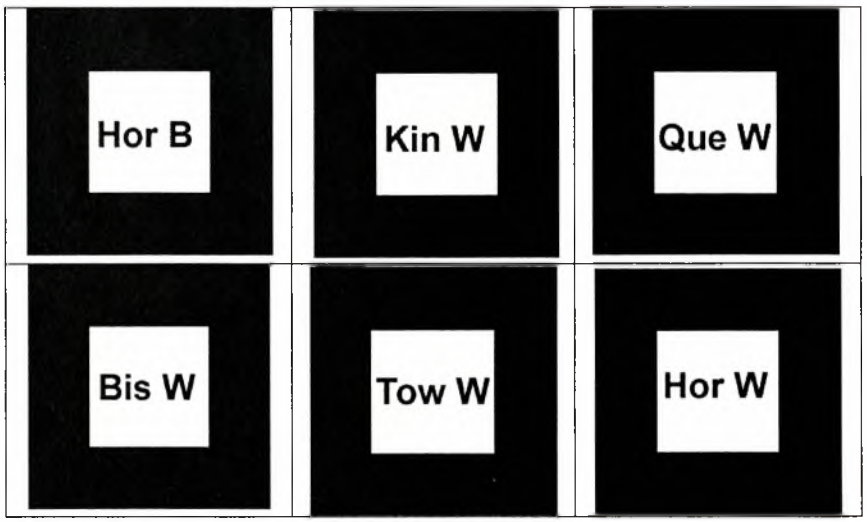
File Generated by timer.c Advanced Time measurement based on QueryPerformanceCounter function

Τμήμα αρχείου εξόδου παραγόμενου από το times.c



Screenshot κατά την διάρκεια διαφορετικής εκτέλεσης με ζοομ σε 6 από τα 12 patterns

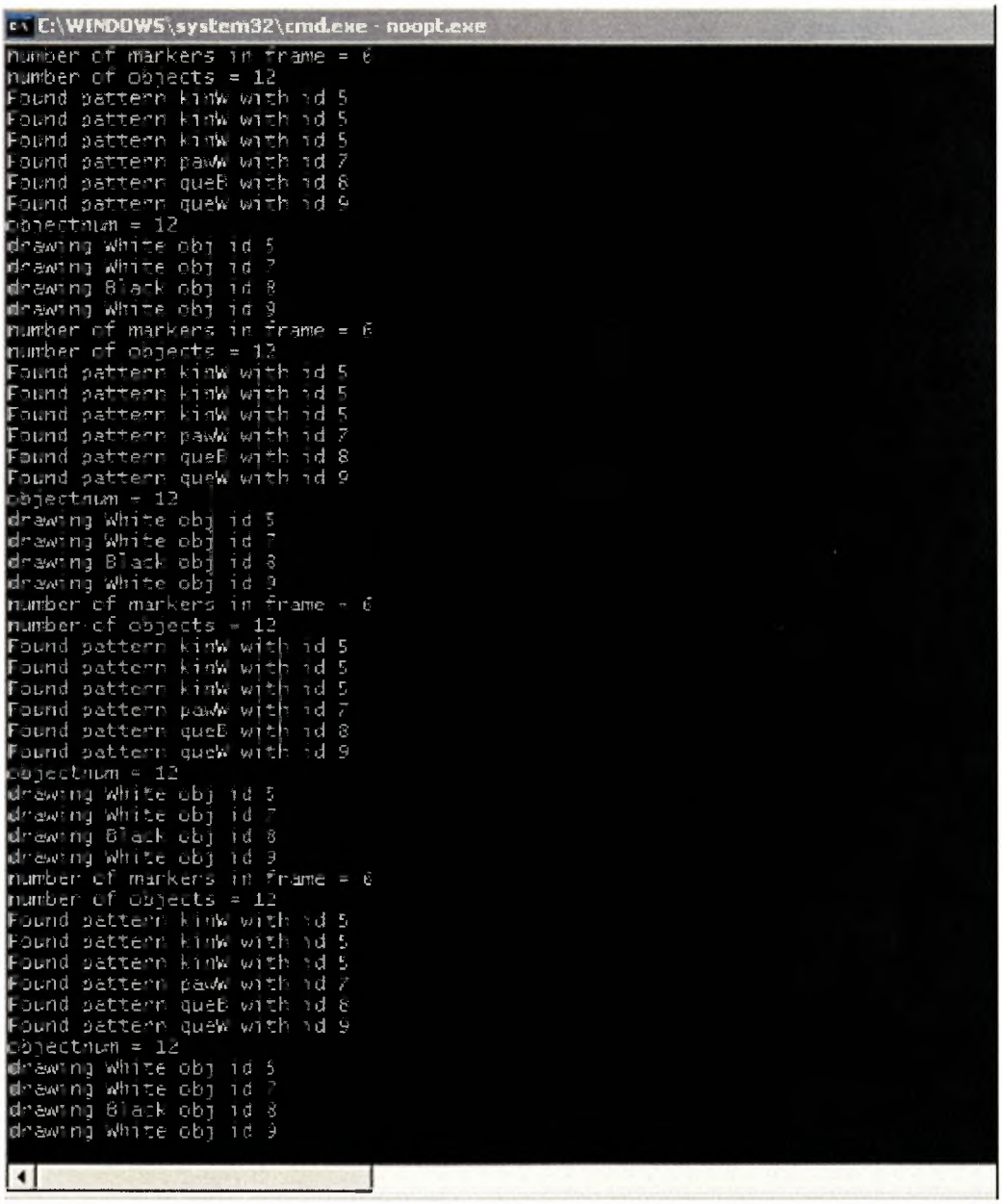
Παρατηρήστε τι συμβαίνει παραπάνω. Έχουν αναγνωριστεί και τα 6 Patterns αλλά με IMCR. Αυτό που κανονικά θα φαίνονταν είναι:



Όπότε θεωρητικά θα έπρεπε να ζωγραφίζονται:

Μαύρο άλογο	Άσπρος βασιλιάς	Άσπρη βασίλισσα
Άσπρος αξιωματικός	Άσπρος πύργος	Άσπρο άλογο

Το ICMR γίνεται πιο κατανοητό αν παρατηρήσουμε στο παράθυρο εκτέλεσης τα διαγνωστικά που έχουμε βάλει να εμφανίζονται.



```
number of markers in frame = 6
number of objects = 12
Found pattern kinW with id 5
Found pattern kinW with id 5
Found pattern kinW with id 5
Found pattern pawW with id 7
Found pattern queB with id 8
Found pattern queW with id 9
objectnum = 12
drawing White obj id 5
drawing White obj id 7
drawing Black obj id 8
drawing White obj id 9
number of markers in frame = 6
number of objects = 12
Found pattern kinW with id 5
Found pattern kinW with id 5
Found pattern kinW with id 5
Found pattern pawW with id 7
Found pattern queB with id 8
Found pattern queW with id 9
objectnum = 12
drawing White obj id 5
drawing White obj id 7
drawing Black obj id 8
drawing White obj id 9
number of markers in frame = 6
number of objects = 12
Found pattern kinW with id 5
Found pattern kinW with id 5
Found pattern kinW with id 5
Found pattern pawW with id 7
Found pattern queB with id 8
Found pattern queW with id 9
objectnum = 12
drawing White obj id 5
drawing White obj id 7
drawing Black obj id 8
drawing White obj id 9
number of markers in frame = 6
number of objects = 12
Found pattern kinW with id 5
Found pattern kinW with id 5
Found pattern kinW with id 5
Found pattern pawW with id 7
Found pattern queB with id 8
Found pattern queW with id 9
objectnum = 12
drawing White obj id 5
drawing White obj id 7
drawing Black obj id 8
drawing White obj id 9
```

Παράθυρο διαγνωστικών στη γραμμή εντολών

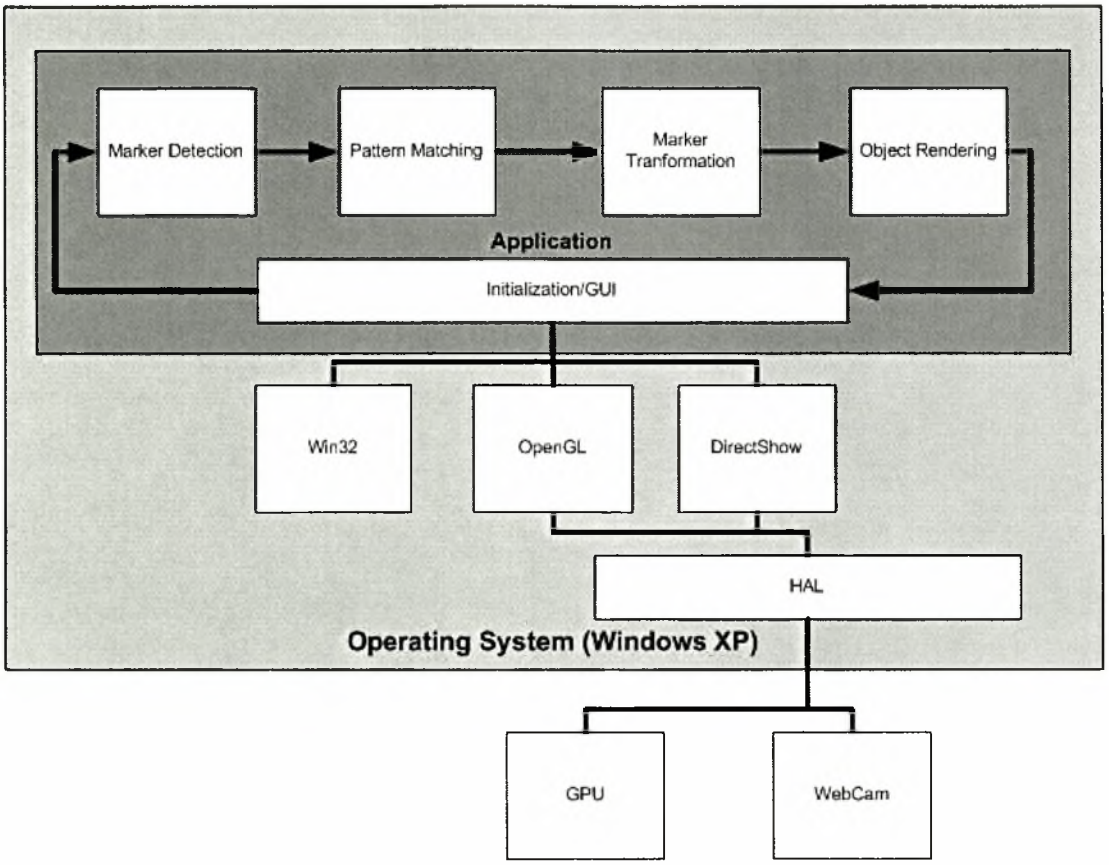
Στο παράθυρο φαίνεται πως αντί για τα προαναφερθέντα patterns το πρόγραμμα βρίσκει 1 φορά σωστά τα queW, queB, pawW αλλά λανθασμένα το kinW 3 φορές. Επειδή η draw έχει ρυθμιστεί να ψάχνει μία φορά για να ζωγραφίσει το κάθε 3d object (εφόσον έχουμε την γνώση ότι κάθε ένα στο πείραμα εμφανίζεται μια φορά) ζωγραφίζεται μόνο το ένα από τα 3. Σημειώνουμε πάλι πως τα patterns δημιουργήθηκαν όπως ακριβώς το hiro και το kanji που περιλαμβάνονται ήδη στο πακέτο του ARtoolkit. Το μόνο που διευκρινίζεται στο ARtoolkit είναι τα patterns να έχουν μονοσήμαντο orientation.

Αρχιτεκτονική συστήματος

Θα χωρίσουμε το σύστημα μας σε 4 διακριτά υποσυστήματα:

- Υποσύστημα αρχικοποίησης και διεπαφής με τον χρήστη. Το υποσύστημα αυτό είναι υπεύθυνο για το initialization της εφαρμογής, το παραθυρικό περιβάλλον, καθώς και διάφορες άλλες λειτουργίες. (**initialization/GUI subsystem**).
- Υποσύστημα εξαγωγής markers (**marker detection subsystem**) .Το υποσύστημα αυτό είναι υπεύθυνο για τον εντοπισμό markers στο τρέχον frame.
- Υποσύστημα ταιριάσματος patterns που βρέθηκαν στο frame, με patterns που υπάρχουν στην βιβλιοθήκη. (**pattern matching subsystem**).Δεχόμαστε σαν είσοδο την έξοδο του προηγούμενου υποσυστήματος και εντοπίζουμε αν στο εσωτερικό κάποιου marker υπάρχει ένα pattern που μας ενδιαφέρει.
- Υποσύστημα μετατροπής του marker matrix (**marker matrix transformation subsystem**).
- Υποσύστημα rendering των μοντέλων στις σωστές συντεταγμένες. (**object rendering subsystem**). Όπως και στο προ-προηγούμενο στάδιο δεχόμαστε σαν είσοδο την έξοδο του προηγούμενου υποσυστήματος και ζωγραφίζουμε στο frame αυτό που μας ενδιαφέρει.

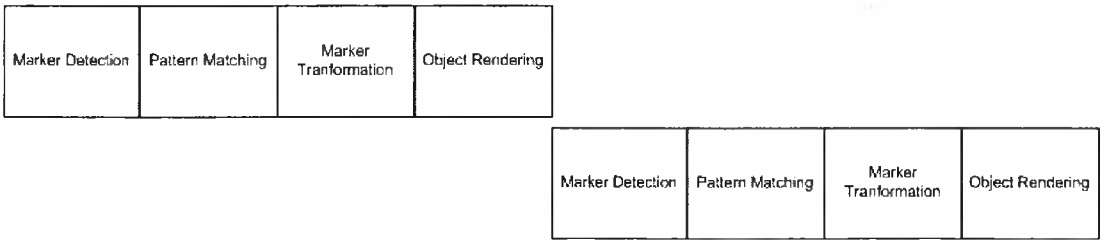
Εδώ παρουσιάζουμε ένα block diagram του προγραμματιστικού κομματιού του συστήματος μας. Πρέπει να τονίσουμε όμως ότι ακόμα δεν έχουμε αναφερθεί στο πώς ακριβώς θα είναι τα fiducials και πως θα είναι εκτυπωμένα και θα χρησιμοποιούνται για να βοηθήσουν στην υλοποίηση του παιχνιδιού σκακιού.



Block Diagram της εφαρμογής μας.

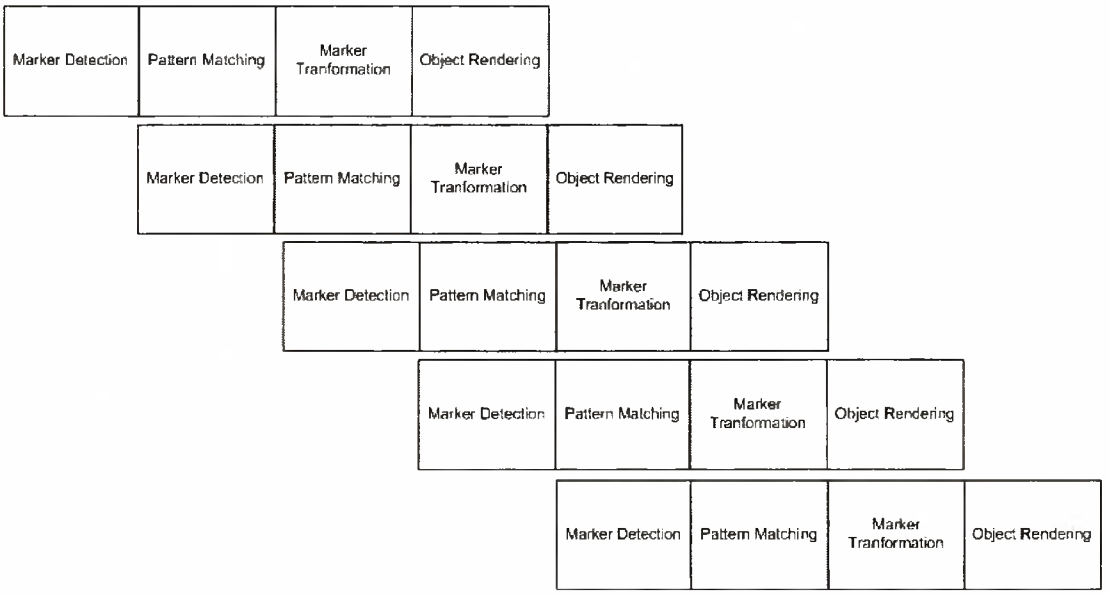
δίνουμε βάση στον τρόπο επικοινωνίας της εφαρμογής με τις βιβλιοθήκες που χρησιμοποιήθηκαν καθώς και με το απαιτούμενο hardware.

Σε αυτό το στάδιο μπορούμε παρατηρώντας προσεκτικά να διαπιστώσουμε ότι το πρόγραμμα μας παρουσιάζει συμπεριφορά παρόμοια με ενός pipeline. Μετά από διακριτά στάδια τα οποία εκτελούνται σειριακά τα δεδομένα μεταφέρονται από στάδιο σε στάδιο ώσπου να παραχθεί το επιθυμητό αποτέλεσμα. Η εκτέλεση λοιπόν εκ πρώτης όψεως θα μπορούσε να μοιάζει με την παρακάτω (δεδομένου ότι έχουμε ένα thread):



Σχεδιάγραμμα εκτέλεσης προγράμματος

Άρα το ερώτημα που δημιουργείται είναι αν θα μπορούσαμε να πετύχουμε με χρήση threads μερική (η πλήρη επικάλυψη) σε επίπεδο υποσυστημάτων έτσι ώστε σε 8 χρονικές στιγμές το πρόγραμμα να εκτελείται 5 φορές αντί για 2, πάντα από αφαιρετικό επίπεδο.



Πλήρης επικάλυψη κατά την εκτέλεση του προγράμματος

Πρέπει λοιπόν να λάβουμε υπόψη τις αλληλοεξαρτήσεις μεταξύ των υποσυστημάτων του προγράμματος (και κατ επέκταση της βιβλιοθήκης καθώς κατά βάση έτσι λειτουργεί). Η ανάλυση ξεκινάει σε αυτό το κεφάλαιο και συνεχίζεται στο επόμενο.

Περιγραφή Λειτουργιών

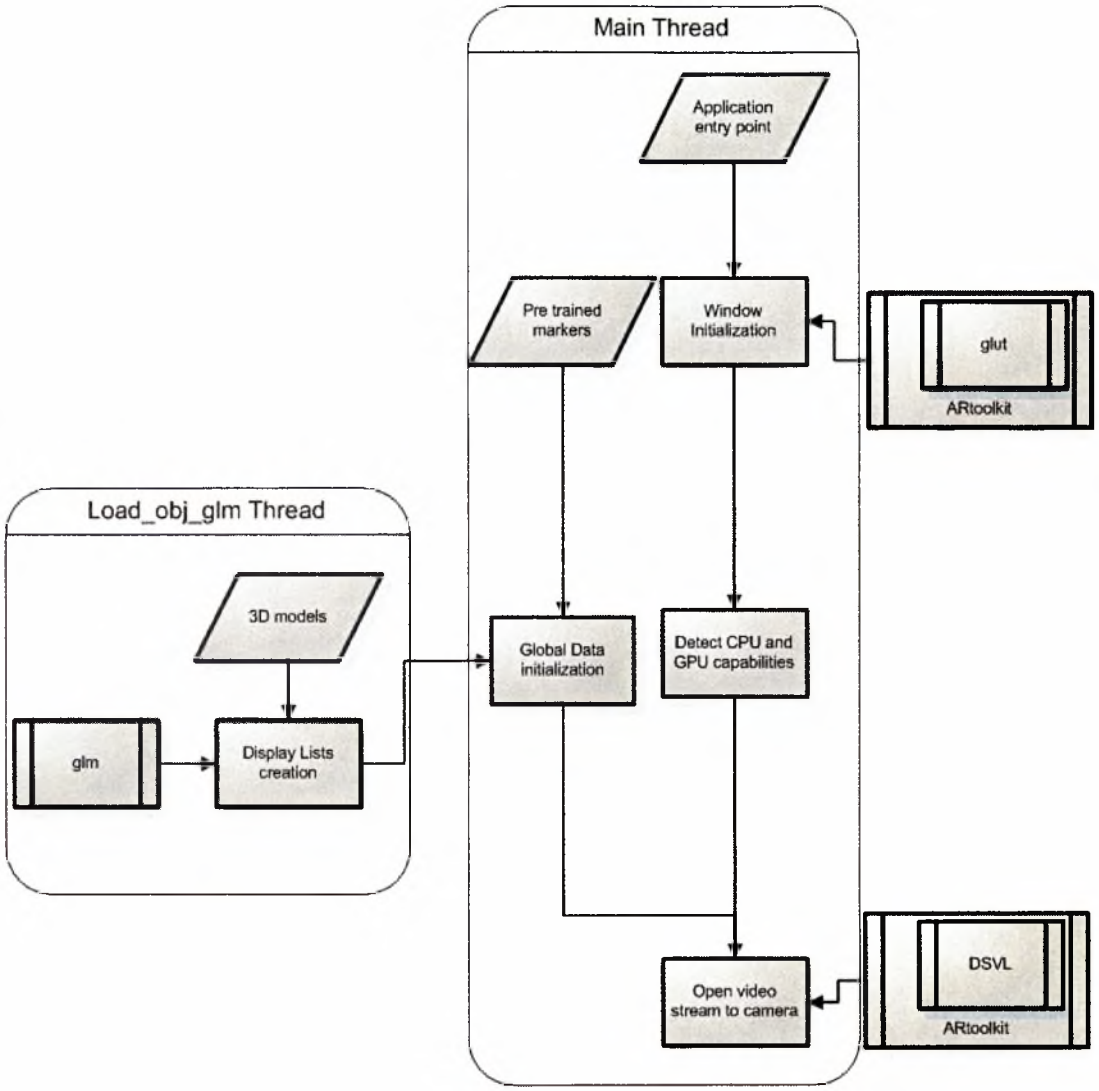
Εδώ περιγράφουμε τις λειτουργίες που απαιτείται να εκτελεί το σύστημα. Προηγουμένως χωρίσαμε ήδη το σύστημά μας σε 4 γενικά υποσυστήματα, τώρα θα χρειαστεί να περιγράψουμε το κάθε υποσύστημα ξεχωριστά. Θα δώσουμε μια μικρή περιγραφή του τρόπου λειτουργίας του κάθε υποσυστήματος καθώς και ένα διάγραμμα ροής το οποίο βέβαια είναι και αυτό σε κάποιο αφαιρετικό επίπεδο. Αξίζει να αναφέρουμε πως αν και χρησιμοποιήσαμε στην ανάλυση μας προηγουμένως την βιβλιοθήκη του ARtoolkit, η εφαρμογή μας διατηρεί την ίδια γενική αρχιτεκτονική με μικρές διαφορές. Οι διαφορές έχουν να κάνουν κυρίως με τον τρόπο που δουλεύουν τα επιμέρους τμήματα και όχι με ριζικές

αλλαγές στην όλη φιλοσοφία των εφαρμογών Augmented Reality που βασίζονται σε fiducial systems.

Initialization/GUI Subsystem

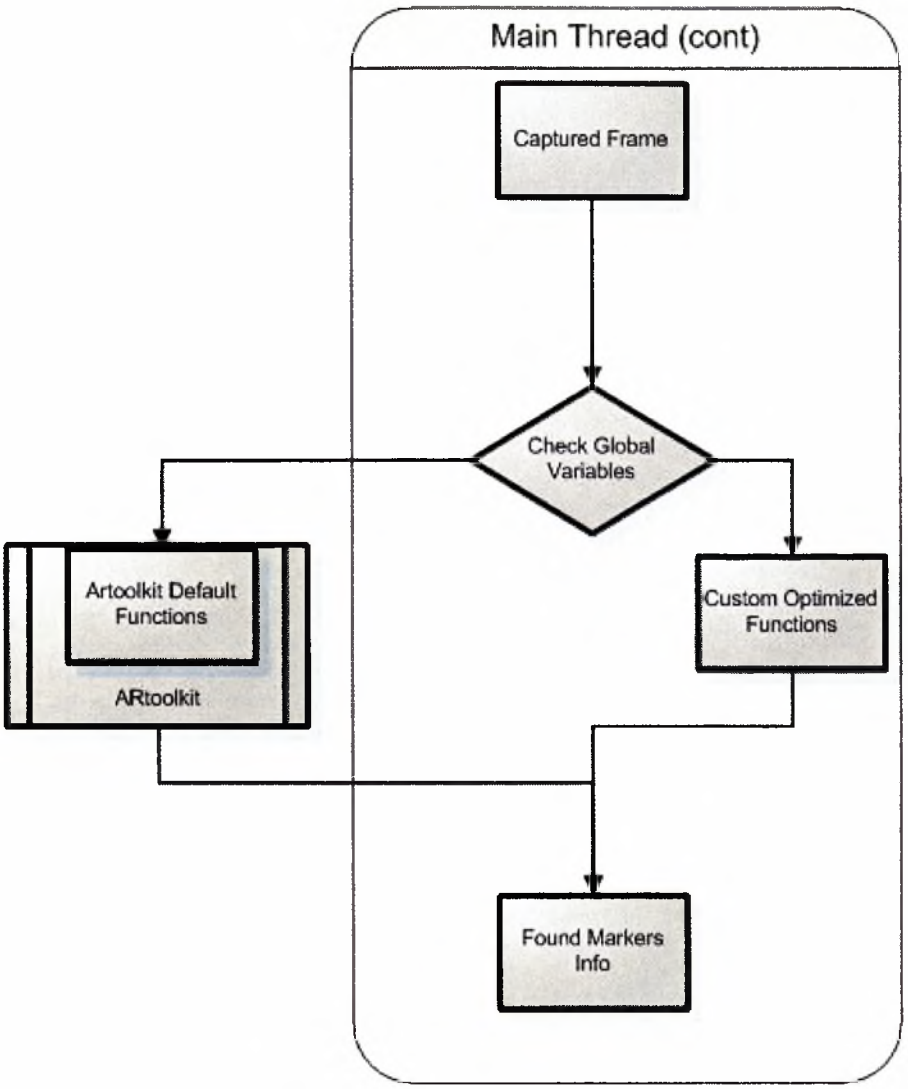
Το υποσύστημα αυτό είναι υπεύθυνο για τις αρχικοποιήσεις που χρειάζεται να γίνουν καθώς και για το παραθυρικό περιβάλλον. Πιο λεπτομερειακά, συμβαίνουν τα εξής:

- αρχικοποίηση του παραθύρου εφαρμογής μέσω της βιβλιοθήκης Win32
- έλεγχος για υποστήριξη διάφορων τεχνολογιών από το CPU
- έλεγχος έκδοσης OpenGL καθώς και υποστήριξης από την κάρτα γραφικών
- άνοιγμα video stream της κάμερας μέσω directshow (η καλύτερα DSVL που είναι ο wrapper)
- αρχικοποίηση δομών δεδομένων
- αρχικοποίηση OpenGL και των display lists (φορτώνουμε και τα τρισδιάστατα μοντέλα μέσω glm library)



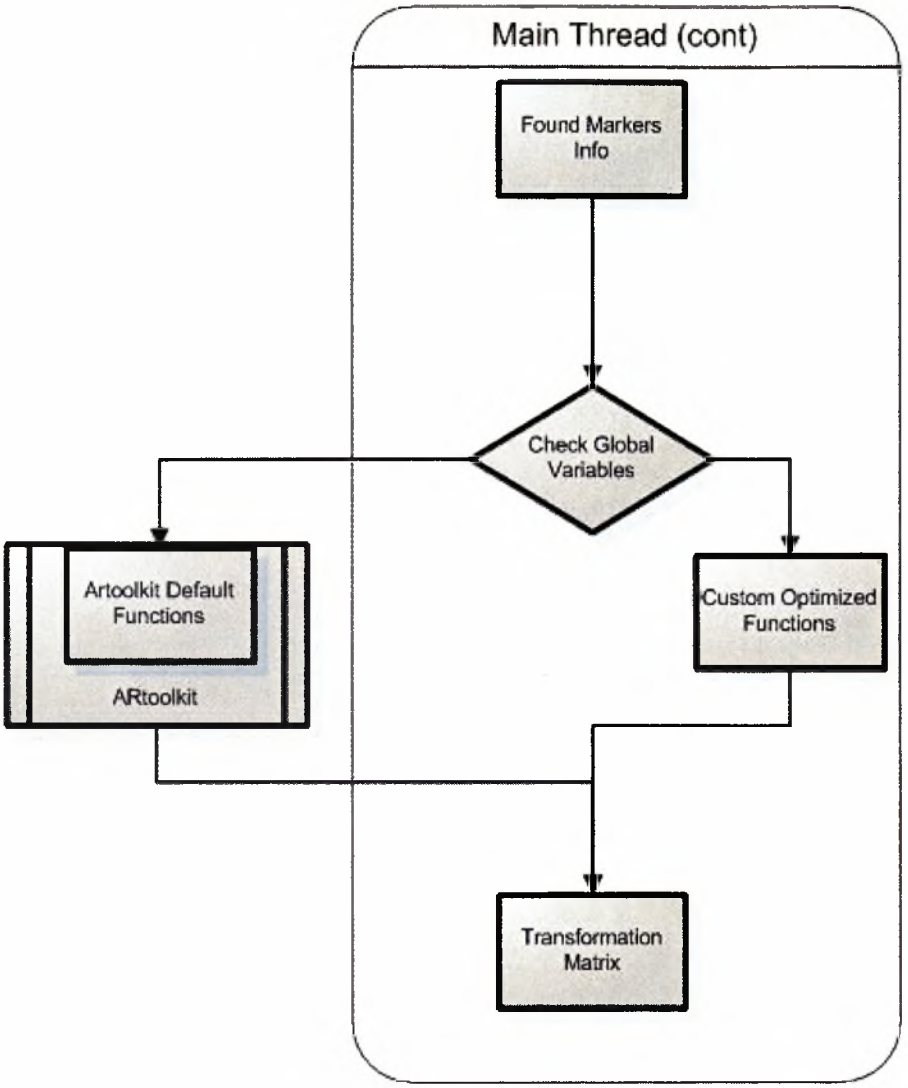
Διάγραμμα Ροής Δεδομένων (Data Flow Diagram) του πρώτου υποσυστήματος.

Marker Detection Subsystem

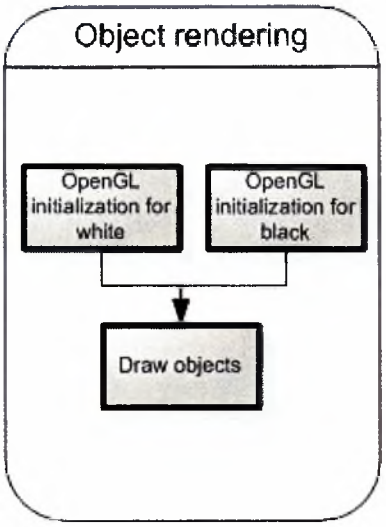


Διάγραμμα Ροής Δεδομένων (Data Flow Diagram) του δεύτερου υποσυστήματος.

Marker Transformation Calculations



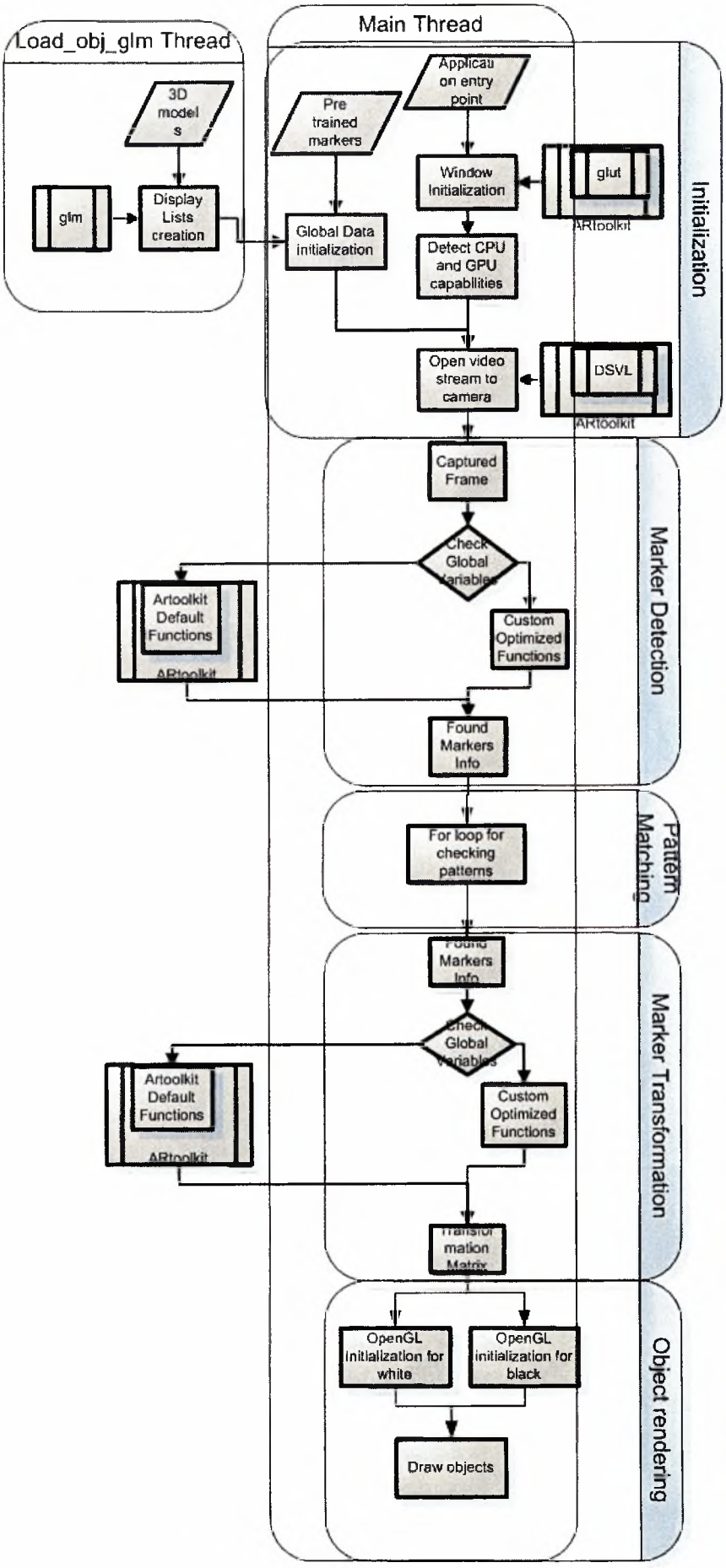
Object Rendering Subsystem



Διάγραμμα Ροής Δεδομένων (Data Flow Diagram) του τέταρτου υποσυστήματος.

Γενική εικόνα

Δεδομένου τώρα ότι έχουμε αναλύσει την αρχιτεκτονική κάθε υποσυστήματος παρουσιάζουμε την συνολική εικόνα του όλου συστήματος όπου εμφανίζονται τα threads καθώς και το βήμα του virtual pipeline που αντιστοιχεί κάθε υποσύστημα.


Load_obj_glm Thread
3D model s
glm
Display Lists creation
Main Thread
Applicat on entry point
Pre trained markers
Window Initialization
glut
ARtoolkit
Global Data initialization
Detect CPU and GPU capabilities
Initialization
Open video stream to camera
DSVL
ARtoolkit
Captured Frame
Check Global Variables
Artoolkit Default Functions
ARtoolkit
Custom Optimized Functions
Found Markers Info
Marker Detection
For loop for checking patterns
Pattern Matching
Found Markers Info
Check Global Variables
Artoolkit Default Functions
ARtoolkit
Custom Optimized Functions
Transfor mation Matrix
Marker Transformation
OpenGL Initialization for white
OpenGL initialization for black
Draw objects
Object rendering

Διάγραμμα Ροής Δεδομένων (Data Flow Diagram) ολόκληρου του συστήματος.

Αν έχουμε κάποια σχετική εμπειρία με threads μπορούμε ήδη να φανταστούμε τον τρόπο που θα γίνεται η επικάλυψη. Το load_obj_glm thread αρχικοποιείται στην αρχή του Initialization και λόγω του ότι ο σκοπός του είναι να διαβάζει αρχεία από τον δίσκο εκτελείται ψευδοπαράλληλα μειώνοντας τον χρόνο του initialization.Το draw thread επίσης αρχικοποιείται κατά το initialization αλλά αντιθέτως δεν καταστρέφεται παρά μόνο μόλις το πρόγραμμα τερματίσει. Περισσότερες λεπτομέρειες μπορούμε να δούμε στα επόμενα κεφάλαια.

Σχεδίαση Συστήματος

Εδώ ακολουθεί η σχεδίαση του συστήματος μας, όπου σε κάθε υποσύστημα γίνεται λεπτομερής αναφορά στις λειτουργίες του. Σε αυτό το κεφάλαιο παρουσιάζονται τμήματα κώδικα η/και ψευδοκώδικα. Όπου «//» θα βάζουμε στα απλά σχόλια ενώ «/* ...*/» σε σχόλια όπου τμήμα(τα) του κώδικα παραλείπονται.

Αρχιτεκτονική βελτιστοποιημένου συστήματος

Σε αυτό το σημείο παρουσιάζουμε τα υποσυστήματα που διαιρείται το πρόγραμμα μας. Τα χωρίζουμε ανάλογα με την λειτουργικότητα τους χρησιμοποιώντας σαν ελάχιστη διαμέριση τον τύπο της κλάσης. Στο τέλος του κεφαλαίου, δίνουμε ένα block diagram κλάσεων που παρουσιάζεται τυχόν κληρονομικότητα και αλληλεπιδράσεις μεταξύ των κλάσεων.

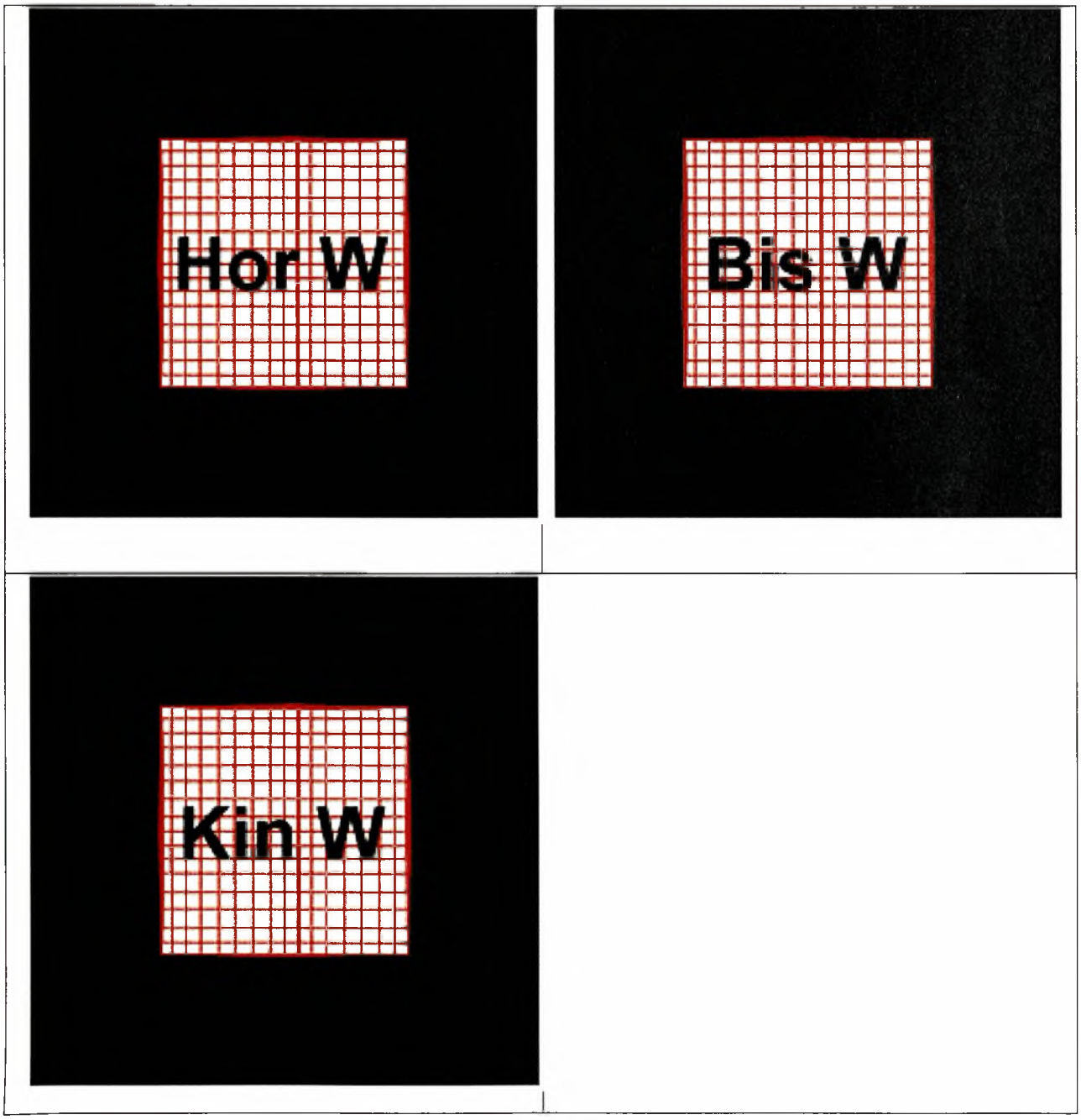
Εδώ παρουσιάζουμε τα επιμέρους κομμάτια από τα οποία θεωρούμε ότι έχει κτιστεί ο κώδικάς μας. Συνήθως, επειδή οι περισσότεροι γράφετε σε αντικειμενοστρεφή γλώσσα προγραμματισμού (C++/JAVA), τα κομμάτια είναι στην ουσία οι κλάσεις της εφαρμογής. Για κάθε κλάση γράψτε μια σύντομη περιγραφή (η αναλυτική περιγραφή των μεθόδων/λειτουργιών/συναρτήσεων του θα ακολουθήσει στην επόμενη ενότητα). Τέλος, δίνουμε ένα γενικό σχήμα που δείχνει τις κλάσεις και πώς αυτές επικοινωνούν μεταξύ τους. Το σχήμα αυτό π.χ. αρκεί να είναι ένα απλό block diagram κλάσεων, όπου θα φαίνεται η κληρονομικότητα και οι συνδέσεις μεταξύ των κλάσεων. Μια κλάση συνδέεται με μια άλλη αν μια μέθοδός της χρησιμοποιεί αντικείμενο από την άλλη ως παράμετρο.

Αντιμετώπιση του IMCR

Όπως προηγουμένως είδαμε το ICMR μας δημιουργεί πολλά προβλήματα. Θα αντλήσουμε ιδέες για την αντιμετώπιση του από τις διάφορες δημοσιεύσεις στον τομέα και κυρίως από τα [OXM01], [Fia05] και [Fia04]

Ας αναλύσουμε όμως πρώτα προτού «αποκαλύψουμε» την λύση μας τι έφταιγε στην προηγούμενη περίπτωση. Ξεκινάμε από τα συγκεκριμένα patterns τα οποία μπερδεύτηκαν μεταξύ τους δηλαδή τα horW, kinW, bisW.

Έχουμε προαναφέρει πως το ARtoolkit κάνει δειγματοληψία στο εσωτερικό ενός marker προτού προχωρήσει σε σύγκριση των patterns που είναι στο frame με τα pre trained patterns. Τα προαναφερθέντα Patterns φαίνονται ως εξής:



Τα 3 patterns παρουσία πλέγματος

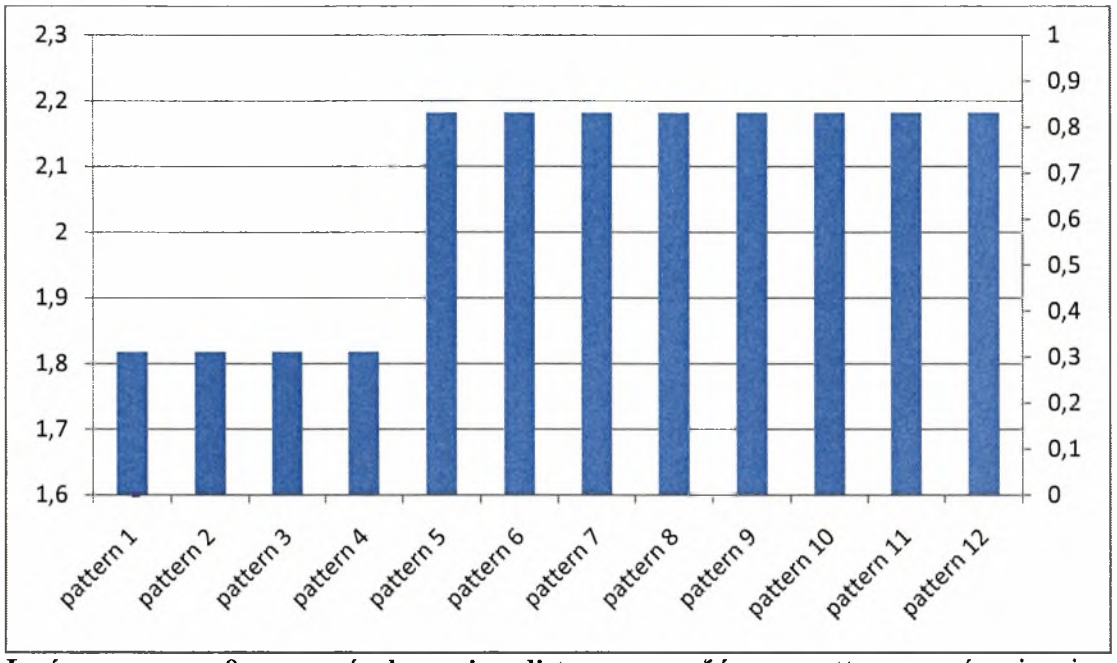
Κάθε pattern μπορεί να θεωρηθεί σαν ένα string μεγέθους 16x16=256 αν κοιτάξουμε τώρα προσεχτικά μπορούμε να δούμε πως λιγότερα από 28 bit είναι η διαφορά μεταξύ των 3 παραπάνω καθώς το W είναι κοινό σε όλα. Το 28 είναι 4x7 το εμβαδόν που καταλαμβάνουν τα "Hor" ,"Bis", "Kin" . Στην πραγματικότητα είναι περίπου τα μισά αν σκεφτούμε πως λόγω δειγματοληψίας είναι πολύ εύκολο το άσπρο να θεωρηθεί μαύρο η/ και το αντίστροφο . Για να συνοψίσουμε αν τα 28 από τα 256 bit διαφέρουν (η καλύτερη περίπτωση) μόνο το (hamming distance/total information size) = 28/256 = 0,109375 της πληροφορίας είναι διαφορετική. Όλα τα παραπάνω είναι «υπερ-βέλτιστες» περιπτώσεις γιατί στην πραγματικότητα λόγω data-corruption φωτισμού και γωνίας του pattern με την κάμερα οι διαφορές μπορεί να είναι πολύ μικρότερες.

Ας παρουσιάσουμε τα βελτιωμένα patterns τώρα. Έχουμε 12 unique patterns και θέλουμε να βρούμε καλύτερο τρόπο να τα κωδικοποιήσουμε ποιός όμως είναι ο βέλτιστος τρόπος ;

Δυαδική μετατροπή:

Pattern_1 horse white	**0000**
Pattern_2 tower white	0001
Pattern_3 bishop white	0010
Pattern_4 pawn white	0011
Pattern_5 queen white	0100
Pattern_6 king white	0101
Pattern_7 horse black	0110
Pattern_8 tower black	0111
Pattern_9 bishop black	1000
Pattern_10 pawn black	1001
Pattern_11 queen black	1010
Pattern_12 king black	1011

Σε αυτό το σημείο δημιουργούμε το πρόγραμμα thrHamDst (theoretical Hamming Distance) στο οποίο περνάμε σαν ορίσματα τις δυαδικές τιμές παραπάνω και δημιουργείται ένα αρχείο που περιέχει υπολογισμένες όλες τις τιμές μεταξύ των patterns.



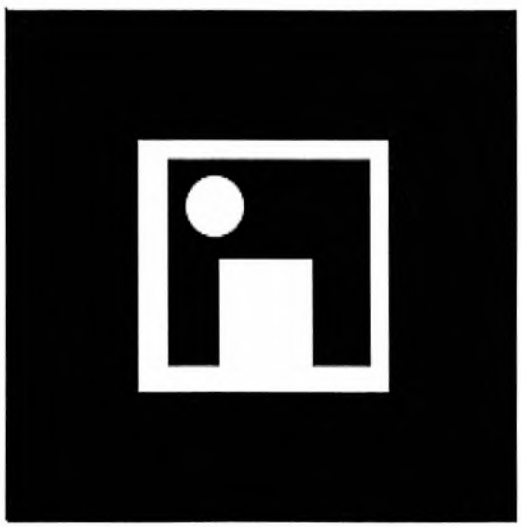
Ιστόγραμμα του θεωρητικών hamming distances μεταξύ των patterns χωρίς mirroring και διαφορετικά orientations.

Το συνολικό hamming distance είναι : 2.060606

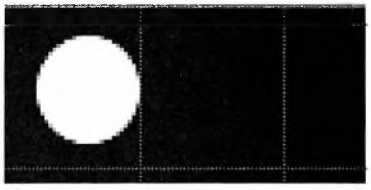
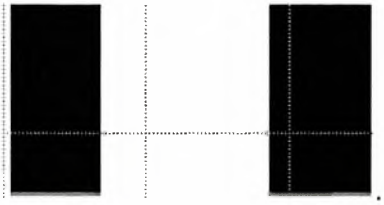
Και ο λόγος (hamming distance/total information size) =2.060606/4 =0,5151515. Πολύ καλύτερος από πριν.

Τα προηγούμενα θα συνέβαιναν αν κάθε σχήμα δειγματοληπτούνταν με 4 bit πληροφορίας , πράγμα όμως που θα καθιστούσε την πληροφορία πολύ ευάλωτη καθώς φωτισμός θέση του pattern κτλ θα επηρέαζαν σε πολύ μεγάλο βαθμό το αποτέλεσμα. Το πραγματικό hamming distance που θα έχουμε μπορεί να υπολογιστεί αν λάβουμε υπόψη μας ότι δειγματοληπτούμε με έναν 16x16 πίνακα.

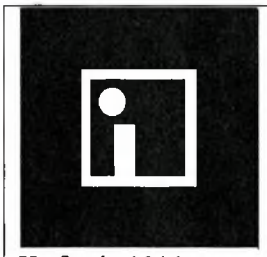
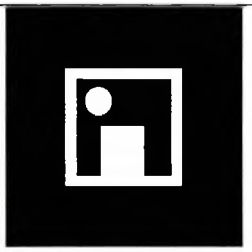
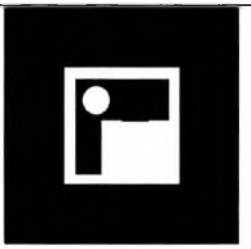
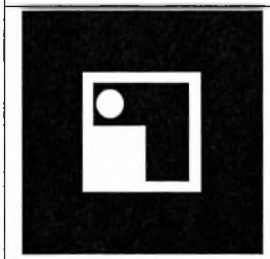
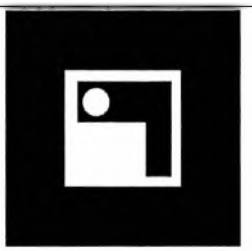
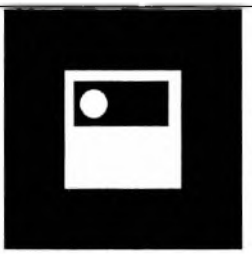
Το κάθε pattern αποφασίζουμε να το αναπαραστήσουμε ως εξής:



Παράδειγμα νέων βελτιστοποιημένων patterns.

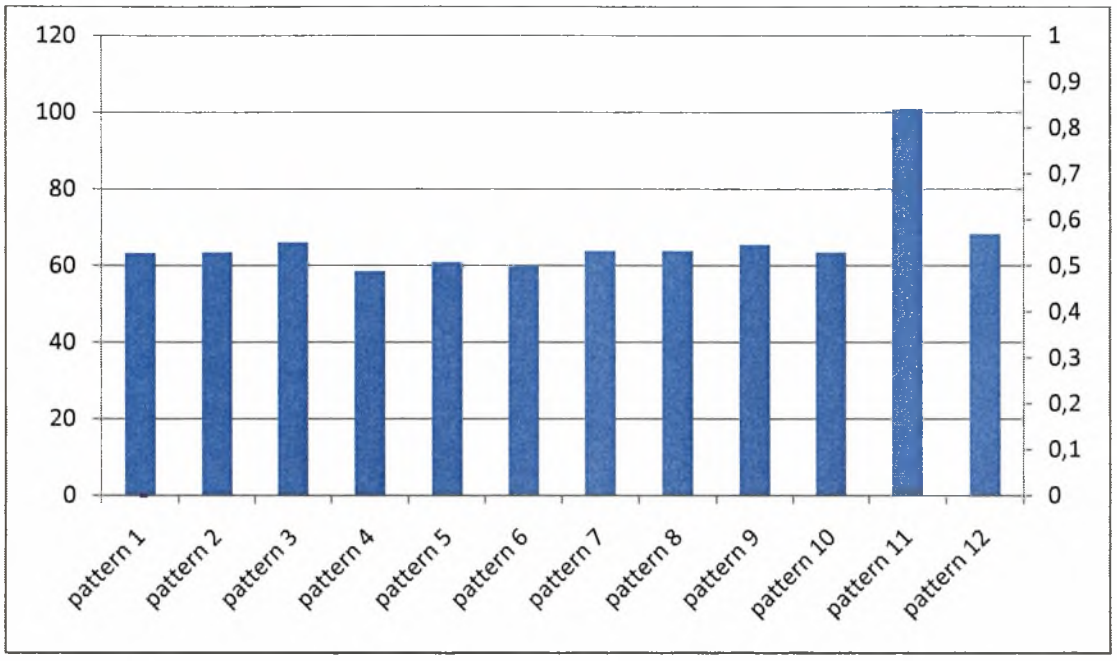
Όπου το σχήμα  χρησιμοποιείται για το orientation του pattern ενώ ακριβώς από κάτω υπάρχει ο κωδικός του pattern όπως περιγράψαμε προηγουμένως στον πίνακα «δυαδική μετατροπή»  .

Το συγκεκριμένο σχήμα ανταποκρίνεται στον κωδικό «1001» που αντιστοιχεί στο "pawn black".

 Κωδικός 1011 KinB	 Κωδικός 1010 QueB	 Κωδικός 1001 PawB	 Κωδικός 1000 BisB
 Κωδικός 0111 TowB	 Κωδικός 0110 HorB	 Κωδικός 0101 KinW	 Κωδικός 0100 QueW
 Κωδικός 0011 PawW	 Κωδικός 0010 BisW	 Κωδικός 0001 TowW	 Κωδικός 0000 HorW

Τα νέα βελτιστοποιημένα patterns

Το μόνο αρνητικό με τα νέα patterns είναι ότι δεν μπορούμε να αναγνωρίσουμε απ'ευθείας σε ποιό μοντέλο ανταποκρίνονται αλλά ας ελπίσουμε ότι το ARtoolkit θα τα καταφέρει καλύτερα ☺.



Ιστόγραμμα πραγματικών hamming distances των βελτιστοποιημένων Patterns χωρίς mirroring και διαφορετικά orientations.

Το συνολικό hamming distance είναι : 66,560603

Και ο λόγος (hamming distance/total information size) = 66,560603/256 =0,2578125

Πρέπει να σημειώσουμε πως και στα 2 παραπάνω ιστογράμματα για το hamming distance κάθε pattern υπολογίζονται PATERNUM-1 ζευγάρια (δηλ 11) ενώ συνολικά (PATERNUM-1)*PATTERNUM ζευγάρια για το συνολικό hamming distance (132). Τα νέα βελτιστοποιημένα patterns αποδεικνύονται σαφώς ανώτερα καθώς στην χειρότερη περίπτωση του pattern 2 έχουμε λόγο 66,090912/256= 0,258167625 (χρησιμοποιώντας το μέσο hamming distance 11 ζευγαριών) σε αντίθεση με τα αρχικά που είχαμε 0,109375.

Marker Detection Optimization

Η όλη λειτουργία του marker detection στο ARtoolkit συνοψίζεται στην συνάρτηση arDetectMarker. Το marker detection είναι ουσιαστικά το στάδιο εξαγωγής περιοχών συνεχόμενου μαύρου χρώματος από το τρέχον frame (που πιθανόν να είναι marker) καθώς και διάφορων χαρακτηριστικών της περιοχής. Σύμφωνα με το [Fia04] πρόκειται και για την μεγαλύτερη συμβολή του ARtoolkit στον τομέα του Augmented Reality βασιζομένου σε patterns. Λίγα μπορούν να βελτιωθούν σχετικά με την λογική που ακολουθείται στην

επεξεργασία του frame καθώς και σε αλγοριθμικά σημεία που γίνεται error checking και παράγεται ο βαθμός βεβαιότητας για το κάθε pattern.

Πολλές όμως αλλαγές μπορούν να γίνουν χρησιμοποιώντας δυνατότητες του hardware καθώς και προγραμματίζοντας σε χαμηλό επίπεδο. Αρχικά δημιουργήσαμε ένα source και ένα header αρχείο με τις βελτιωμένες συναρτήσεις έτσι ώστε να κάνουμε πολύ εύκολα override τις αρχικές συναρτήσεις του ARtoolkit.H ardetectmarker αντικαθίστανται από την ardetectmarkerO όπου "O" Optimized. Επιλέξαμε καταρχάς να βελτιστοποιήσουμε τμήματα κώδικα στο εσωτερικό βρόγχων που εκτελούνται αρκετές φορές έτσι να είναι ορατά τα αποτελέσματα. Οι καταχωρητές SSE είναι φτιαγμένοι με σκοπό την πράξη μεταξύ αριθμών τύπου float με τον κάθε έναν από τους καταχωρητές να μπορεί να αποθηκεύσει 4 floats. Έτσι έχουμε 4*32=128bit που είναι και το μέγεθος κάθε καταχωρητή. Η γενική αλγοριθμική μορφή που θα αντιμετωπίσουμε σε όλες τις περιπτώσεις είναι:

```
for(i=0; i<something ;i++){
        (πιθανώς εμφωλευμένα for ή while loops)
        Πράξεις πάνω σε δεδομένα τύπου float
        }
```

Το οποίο θα αντιμετωπίζεται ως εξής

```
for(i=0; i<something ;i++){
        (πιθανώς εμφωλευμένα for ή while loops)
        Πέρασμα δεδομένων στους καταχωρητές SSE
        Πράξεις μεταξύ των καταχωρητών
        Εξαγωγή των αποτελεσμάτων από τους καταχωρητές
        }
```

Στην επόμενη παράγραφο αναλύουμε διεξοδικά τις 2 μεθοδολογίες που ακολουθήθηκαν ωστόσο εδώ θα αναλύσουμε την γενική χρήση τους:

Αλγοριθμικά τα SSE intrinstics χρησιμοποιούνται ως εξής:

```
__m128 tmp;
__declspec(align(16)) float p[4];
...
for(i=0; i<something ;i++){
        (πιθανώς εμφωλευμένα for ή while loops)
        ....
      tmp = _mm_set_ps(float z , float y , float x , float w ); /*...passing data to registers*/

       _mm_add_ps(__m128 tmp2,__m128 tmp3); /*...various combinations of operations*/
      _mm_mul_ps(__m128 tmp2,__m128 tmp3);
      _mm_sub_ps(__m128 tmp2,__m128 tmp3);

      _mm_store_ps(float *p, __m128 a); /*...storing the result*/
       }
...
```

Σε αντίθεση η χρήση inline assembly με καταχωρητές SSE γίνεται ως εξής:

```
typedef __declspec(align(32)) struct     //aligning data according to their size in this case 32
      {
            float v00,v01,   v10,v11,        v20,v21,        v30,v31;
      }input;                                          //input struct for the SSE registers

typedef __declspec(align(16)) struct
      {
            float output1, output2, output3, output4;
      }output;                                         //output struct for the SSE registers

...
input Mem[30];
output out;

for(i=0; i<something ;i++){
```

```
        (πιθανώς εμφωλευμένα for ή while loops)
        ....
        __asm{
                push eax          //eax or whatever register changes value inside inline
assembly
                ...
                  mov eax, j              //getting the correct index number in eax
            imul eax, 16
                ...
                movaps xmm0, Mem[eax] //tranfering data from Mem[eax] to SSE
register
                ...
                  subps   xmm0, xmm1          /*...various combinations of
simultaneous operations*/
            mulps   xmm0, xmm0
                ...
                shufps xmm1, xmm1,0xAA /*...possible data shuffling*/
                ...
                pop eax              //restoring register(s) state
                }
        }
...
```

Pattern Matching Optimization και Marker Coordinates Transformation Optimization

Το pattern matching είναι η διαδικασία κατά την οποία συγκρίνουμε τα marker id και αποφασίζουμε αν κάποιο από τα συγκεκριμένα patterns που βρίσκεται στην μνήμη αντιστοιχεί σε pattern που εντοπίστηκε στο τρέχον frame. Στα παραδείγματα του ARtoolkit που περιλαμβάνονται ακολουθείται μια συγκεκριμένη μεθοδολογία η οποία παρουσιάζεται παρακάτω:

```
...
for( i = 0; i < objectnum; i++ ) {                    //0 ws 12 stin periptwsi mas
                k = -1;
        for( j = 0; j < marker_num; j++ ) {          //0 ws arithmo marker pou vrethikan
                  if( object[i].id == marker_info[j].id) {
                              //sto object einai apothikeymena ta pre trained patterns
                              // enw sto marker info ta eswterika twn marker pou vrethikan sto frame


                              /*... you've found a pattern and drawing green rectangle*/
                                  if( k == -1 ) k = j;
                        else // make sure you have the best pattern (highest confidence factor)
                                  if( marker_info[k].cf < marker_info[j].cf ) k = j;
                          }
                        }
                        if( k == -1 ) {
                                  //an kammia fora den itan iso to object me to marker vale oti den einai visible
                                                          object[i].visible = 0;
                                                      continue;
                              //feygei apo ton vrogxo an to antik den einai orato
                        }

                        marker_matched++;
                        /* ...calculate the transform for each marker */
                        ...
                        object[i].visible = 1;
        }
...
```

Object Rendering Optimization (ακόμα και κάποια αποτυχία υλοποίησης είναι σημαντικό να αναφερθεί)

Το object rendering είναι ένα πολύ κρίσιμο σημείο της εφαρμογής μας. Σύμφωνα με τα παραδείγματα του ARtoolkit ο τρόπος που παρουσιάζεται είναι ο εξής:

```
...
for( i = 0; i < objectnum; i++ ) {
        if( object[i].visible == 0 ) continue;
        argConvGlpara(object[i].trans, gl_para);
        draw_object( object[i].id, gl_para);
    }
...
```

```
static int    draw_object( int obj_id, double gl_para[16])
{

    ...
    argDrawMode3D();
    argDraw3dCamera( 0, 0 );
    glMatrixMode(GL_MODELVIEW);
    glLoadMatrixd( gl_para );


            ...
        if(obj_id == 0){

        }
        else {

        }
    argDrawMode2D();
    return 0;
}
```

Μελετήσαμε 2 διαφορετικούς τρόπους βελτιστοποίησης αυτού του σταδίου του προγράμματος:

1 thread ανά τρισδιάστατο αντικείμενο :

```
HANDLE Handle_Of_Thread_0 = 0;
HANDLE Handle_Of_Thread_1 = 0;
/* ..all thread HANDLES declarations*/
HANDLE Array_Of_Thread_Handles[numberofobjects];
...
```

```
static void draw( ObjectData_T *object, int objectnum ) {
for( i = 0; i < objectnum; i++ ) {
        if( object[i].visible == 0 ) continue;
        argConvGlpara(object[i].trans, gl_para);
        switch(object[i].id){

    case 0: {
                    Array_Of_Thread_Handles [0]= CreateThread( NULL, 0,
                                                    Thread_no_0, gl_para, 0, NULL);
                                        }
    }
    case 1: {
    }

  WaitForMultipleObjects(
   2,
   Array_Of_Thread_Handles,
   FALSE,
   INFINITE);
   ...
   }
...
}
```

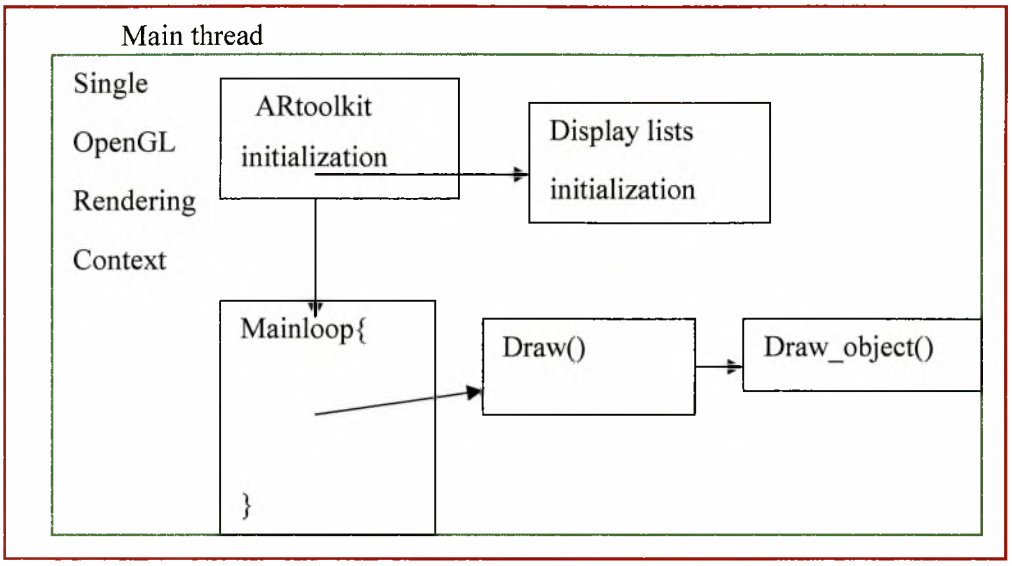
1 drawing thread

```
static void init( void )
{
...
hRenderThread = (HANDLE)_beginthreadex(NULL, 0, draw_thread, data, 0, &nRenderThreadID);
}
```

```
static void draw( ObjectData_T *object, int objectnum )
{
...
for( i = 0; i < objectnum; i++ ) {
        if( object[i].visible == 0 ) continue;
        if(PostThreadMessage(nRenderThreadID, UWM_READ, object[i].id, gl_para)==0)
            {


            }

...
}
...
}
```

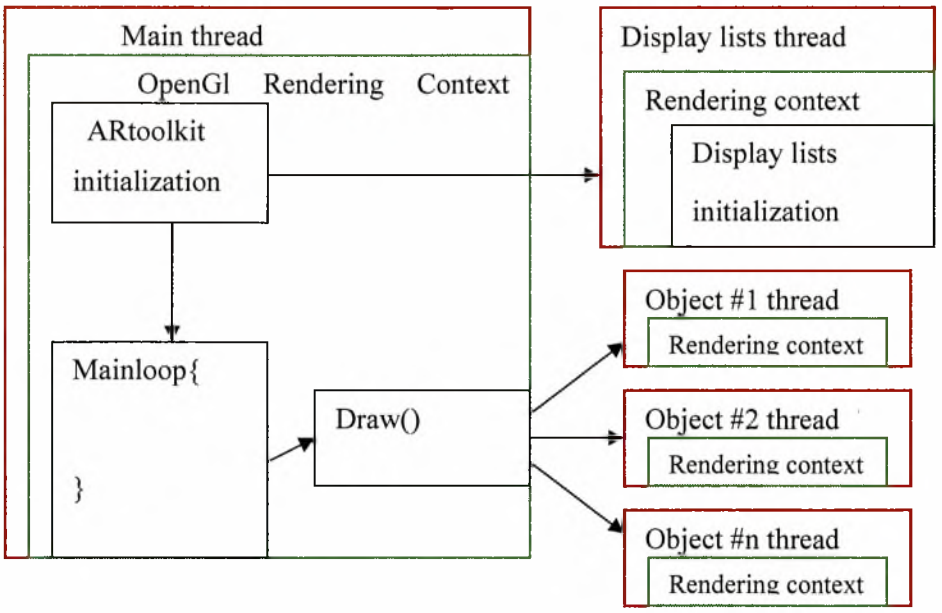
Σε αντίθεση με την 1η λύση για το πρόβλημα βελτιστοποίησης του ARdetectmarker που την απόρριψη της αιτιολογούμε στην επόμενη παράγραφο (καθώς χρειάζεται να αναλύσουμε κώδικα σε βάθος) η αιτιολόγηση απόρριψης της λύσης βελτιστοποίησης του object rendering με πολλαπλά threads μπορεί να γίνει σε πιο αφηρημένο επίπεδο. Θα αναλύσουμε την συμπεριφορά των threads στην συγκεκριμένη υλοποίηση καθώς και τμήματα αρχιτεκτονικής του OpenGL και του συστήματος σχεδιασμού παραθύρων των windows, WIN32. (η ανάλυση μπορεί να επεκταθεί και για το σύστημα παραθύρων του linux καθώς έχουν παρόμοια συμπεριφορά).

Έστω ότι δεν χρησιμοποιούσαμε κανένα thread στο πρόγραμμα. Σχηματικά θα είχαμε μια εξής αφαιρετική διάταξη αρχιτεκτονικής του προγράμματος:



Όπου ουσιαστικά έχουμε ένα OpenGL context και ένα window HANDLER. Το σύστημα δημιουργεί context και κάνει το απαραίτητο initialization μέσω της συνάρτησης arginit(); Η οποία με την σειρά της καλεί την glutinitwindow(); .Αυτή είναι οπτικά η αρχιτεκτονική του noopt.

Στην περίπτωση τώρα που μελετάμε έχουμε το εξής:



Κάθε τύπος αντικειμένου (6 συνολικά) αντιστοιχεί σε 2 pattern ένα για μαύρο και ένα για άσπρο. **Κάθε** τύπος αντικειμένου έχει ένα δικό του thread το οποίο είναι υπεύθυνο για διάφορες διεργασίες που πρέπει να γίνουν για την ομαλή εμφάνιση του στην οθόνη περιλαμβάνοντας κλήσεις συναρτήσεων του OpenGL. **Έτσι** για παράδειγμα αν αναγνωριστούν 3 αντικείμενα στην οθόνη **θα δημιουργηθούν μόνο 3 thread** (το καθένα με

το context του) κατά τον έλεγχο του switch-case και μετά θα καταστραφούν μαζί με το rendering context τους. **Αν** στο επόμενο loop δεν αναγνωριστεί τίποτα **δεν δημιουργείται κανένα thread**. Το μόνο το οποίο μένει στη μνήμη και δεν διαγράφεται είναι το rendering context των display lists το οποίο επαναχρησιμοποιείται από κάθε thread που δημιουργείται.

Και εδώ μπορούν να παρατηρηθούν τα προβλήματα. **Το OpenGL καθώς και το σύστημα παραθύρων δεν υποστηρίζουν ζωγράφισμα στην οθόνη από πολλαπλά threads**. Κάθε thread για να ζωγραφίσει στο παράθυρο χρειάζεται να κλειδώσει το window HANDLER να δημιουργήσει rendering context με βάση το ήδη υπάρχον context στη μνήμη και μετά να ζωγραφίσει. Αυτό προσθέτει μεγάλο overhead και τελικά το όλο πρόγραμμα πάει πιο αργά αντί να πηγαίνει πιο γρήγορα. **Θα μπορούσαμε άραγε να δημιουργούμε κατά το initialization τα threads και απλά να τους περνάμε κάποιο μήνυμα για το αν το αντικείμενο πρέπει να ζωγραφιστεί η όχι;** Η λύση αυτή μελετήθηκε επίσης αλλά το πρόβλημα συνεχίζει να υπάρχει σε κάθε thread χρειάζεται να περάσουμε το gl_para και να το φορτώσουμε στο OpenGL. Το gl_para είναι ένας πίνακας που περιέχει το marker transformation του αντικειμένου σχετικά με το camera viewport. Οπότε θα γλιτώναμε κάποιες πράξεις (υπό το πρίσμα ότι το πρόγραμμα ίσως εκτελούνταν σε multi-core επεξεργαστή) αλλά πάλι θα έπρεπε να περάσουμε το window handler το gl_para και μετά να ζωγραφίσουμε στην οθόνη. Οι 2 παραπάνω λύσεις καθώς και διάφορες παραλλαγές τους απορρίφθηκαν ως μη-βέλτιστες.

Δοκιμάστηκε μια ακόμα λύση πιο μεγάλης πολυπλοκότητας. Ένα thread να δημιουργείται κατά την εκκίνηση του προγράμματος με σκοπό να περιμένει είσοδο από το main thread για να ζωγραφίσει κάτι. Κάναμε τις απαραίτητες αλλαγές στην βιβλιοθήκη δημιουργώντας συναρτήσεις που να μας επιστρέφουν κάποιες από τις μεταβλητές που χρειαζόμαστε στο thread (returnxsize, returnysize, zoom κτλ) Το thread αφαιρετικά παρουσιάζεται ακριβώς παρακάτω:

```
unsigned int __stdcall draw_thread(threaddata *data){

//opengl declarations

while(runstatus){
if(PeekMessage(&intmsg, NULL, DRAW_OBJ, DISP_FRAME, PM_REMOVE)){
                        switch(intmsg.message)
                        {

                                case DRAW_OBJ:


                                        enter = 1;
                                        objid=intmsg.wParam;
                                        gl_para3=intmsg.lParam;
```

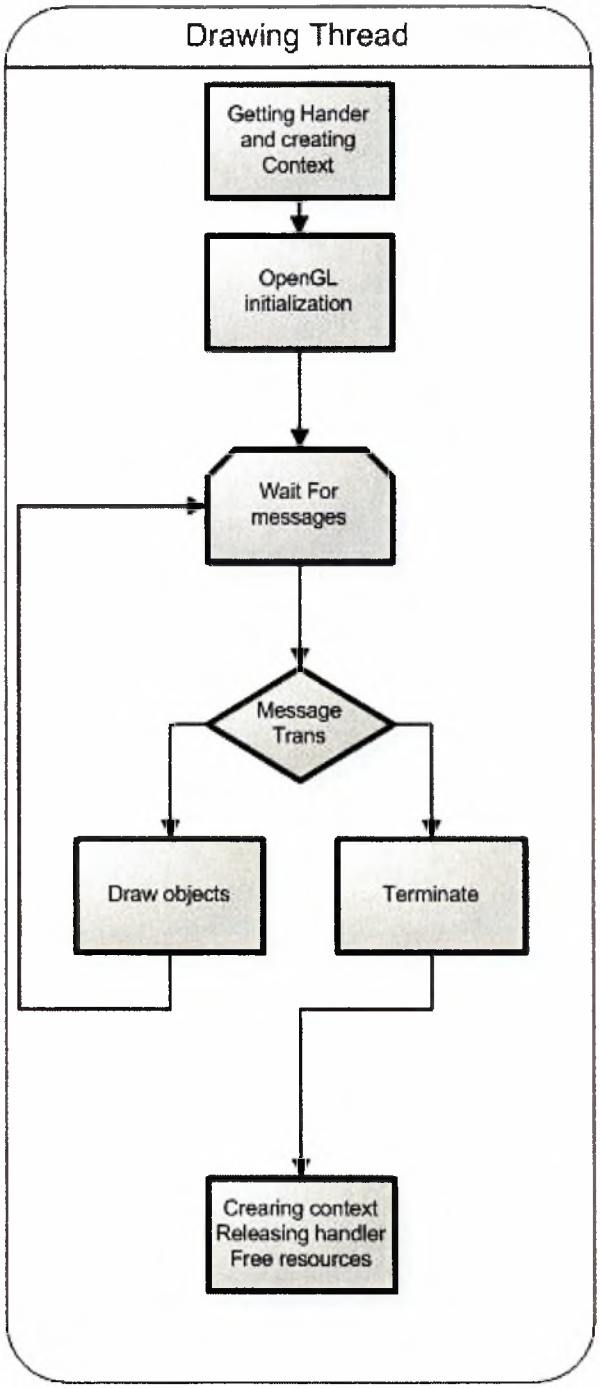
```
                    break;

                case STOP:
                    runstatus = 0;
                    printf("terminating thread");
                    break;

                //draw the object

            }
}
```

Το πρώτο πράγμα που παρατηρούμε είναι ότι thread επιστρέφει διαφορετικό τύπο δεδομένων από ότι πριν. Αυτό γιατί το thread το δημιουργούμε με διαφορετικό τρόπο χρησιμοποιώντας την _beginthreadex ως εξής:

hRenderThread = (HANDLE)_beginthreadex(NULL, 0, draw_thread, data, 0, &nRenderThreadID);

Όπου hRenderThread ο handler του thread το data είναι διάφορα δεδομένα που περνάμε σχετικά με το μέγεθος του παραθύρου η παράμετρος zoom κτλ. Μπορούμε να παρατηρήσουμε τώρα στο παραπάνω πλαίσιο ότι έχουμε δημιουργήσει ένα message queue όπου δέχεται συγκεκριμένο range μυνημάτων ένώ με το PM_REMOVE λέμε στο queue να μυνήματα να διαγράφονται μετά την λήψη τους. Το msg είναι ένας τύπος δεδομένων όπου μπορούμε να χρησιμοποιήσουμε για να στείλουμε μυνήματα από και σε thread. Το main thread στέλνει μυνήματα ως εξής:

```
if(PostThreadMessage(nRenderThreadID, DRAW_OBJ, object[i].id, gl_para)==0)
                {
                    printf("thread error %d",GetLastError());
                }
```



Το σχεδιάγραμμα των μεταβολών της drawobject (δεν χρησιμοποιήθηκε τελικά)

Όπου nRenderThreadID το id του thread που στέλνουμε το μύνημα. Προσοχή γιατί αν δεν υπάρχει message queue θα λάβουμε ένα system error σε επίπεδο λειτουργικού. **Η λύση αυτή είναι από τεχνικής άποψης άψογη και πάρα πολύ χρήσιμη αλλά όχι για εφαρμογές έχουν input από κάμερα ή από άλλη οπτική συσκευή. Το πρόβλημα για μια ακόμα φορά εντοπίζεται στο OpenGL.** Εκτός από την συνάρτηση ζωγραφίσματος η συνάρτηση που παίρνει δεδομένα από την κάμερα, και τα αποθηκεύει στο buffer πρόσβαση επίσης στο

παράθυρο χρειάζεται και η argDispImage. Η drawobjects λαμβάνει κάθε φορά δεδομένα μέσω της δομής msg και ζωγραφίζει ανάλογα τα αντικείμενα που χρειάζεται. Πρέπει όμως κάθε φορά πριν καλέσουμε την PostThreadMessage να γεμίσουμε την δομή data με το window handler το OpenGl context να συσχετίσουμε το data με το msg wParamm και μέσω του thread με τις εντολές GetDC και wglMakeCurrent να ξανακάνουμε τις διάφορες αρχικοποιήσεις. **Η λύση αυτή δουλεύει αλλά είναι αργή**.

Η τελική λύση που χρησιμοποιούμε δημιουργεί ένα μόνο ένα thread κατά την αρχικοποίηση για να φορτώνονται τα διάφορα αντικείμενα.

Περιγραφή Συναρτήσεων

Βασιζόμαστε στις παρατηρήσεις του προηγουμένου κεφαλαίου κατά τον διαχωρισμό των υποσυστημάτων, έτσι ώστε να χρησιμοποιήσουμε τις ίδιες παραδοχές για την γενική λειτουργία του συστήματος μας, και τον ίδιο λογικό διαχωρισμό των λειτουργιών του.

Εδώ περιγράφουμε συνοπτικά τις λειτουργίες/μεθόδους/συναρτήσεις των κλάσεων. Καλό είναι οι περιγραφές να δίνονται σύντομα, περιεκτικά και αριθμημένα, π.χ. 1. αυτό, 2. το άλλο, κ.λ.π. Δεν βάζουμε κώδικα καθόλου!

IMCR Specific

thrHammingD

Η συνάρτηση αυτή δημιουργεί έναν πίνακα 4x12 για να προσομοιώσει την συμπεριφορά και να βρει το hamming distance δεδομένου ότι δεχόμαστε ότι έχουμε για δειγματοληψία μεγέθους 4 για κάθε pattern. Στο τέλος καλεί την hammingD.

realHammingD

Η συνάρτηση αυτή διαβάζει τα αρχεία *.patt που έχει δημιουργήσει το ARtoolkit κατά το στάδιο του pattern training. Αποθηκεύει σε έναν πίνακα 256x12 μόνο το πρώτο orientation κάθε pattern . Στο τέλος καλεί την hammingD.

hammingD

Η hammingD δέχεται ως ορίσματα έναν πίνακα με προκαθορισμένο αριθμό στηλών αλλά μεταβλητό αριθμό γραμμών και στην περίπτωση μας οι γραμμές είναι 12 (για 12 Patterns) ενώ οι στήλες είναι ουσιαστικά το μέγεθος δειγματοληψίας. Περνιέται επίσης ένα string με το όνομα αρχείου που θα αποθηκευτούν τα αποτελέσματα (ανάλογα με το αν καλείται από την thrHammingD η την realHammingD).

Check Hardware Functions (initcheck, checkCPU, checkGPU)

Αυτού του τύπου οι συναρτήσεις εκτελούνται κατά το initialization time και επιτελούν 3 κυρίως λειτουργίες:

1) Έλεγχο για τυχόν υποστήριξη της τεχνολογίας MMX από τον επεξεργαστή.
2) Έλεγχο για τυχόν υποστήριξη της τεχνολογίας SSE από τον επεξεργαστή.
3) Ελέγχει την έκδοση OpenGL που υποστηρίζει το σύστημα καθώς και διάφορα extemsions του OpenGL.

Η λειτουργικότητα είναι πολύ σημαντική οι 2 πρώτες λειτουργίες επιτελούνται με την χρήση της εντολής σε assembly CPUID. Αυτή η εντολή είναι απαραίτητη σε όλα τα προγράμματα που κάνουν χρήση συγκεκριμένων καταχωρητών SIMD αλλιώς σε περίπτωση που πάμε να εκτελέσουμε τέτοιες πράξεις και δεν υποστηρίζονται από τον επεξεργαστή τότε έχουμε exceptions και το πρόγραμμα μας τερματίζει. Η εντολή περιγράφεται στο [Int99-2] (σελίδα 3-111), ενώ στο [Int99-1] (σελίδα 8-10) αναφέρεται πως είναι απαραίτητο να γίνεται ο έλεγχος αυτός. Για την τρίτη λειτουργία ελέγχουμε την έκδοση του OpenGL που είναι εγκατεστημένη στο μηχάνημα εκτέλεσης του προγράμματος. Αυτό το βήμα είναι εξίσου σημαντικό και γίνεται για να αποφευχθούν τυχόν exceptions που μπορούν να συμβούν στην περίπτωση που ο χρήστης έχει παλαιότερη έκδοση του OpenGL από την ζητούμενη.

Display List Functions (load_glm_obj)

Η κλάση μας αυτή είναι υπεύθυνη για την διαχείριση των τρισδιάστατων μοντέλων τα οποία ζωγραφίζονται στον εκάστοτε marker:

1) Φορτώνει 6 τρισδιάστατα μοντέλα τύπου *.obj.
2) Κάνει τις απαραίτητες πράξεις πάνω στα μοντέλα μέσω της βιβλιοθήκης glm η οποία είναι υπεύθυνη για την διαχείριση τους.
3) Δημιουργεί ένα display list για το κάθε μοντέλο το οποίο αποθηκεύεται σε έναν global εμβέλειας πίνακα.

Η κλάση αυτή είναι σχετικά εύκολα κατανοητή αν και πολύ σημαντική για την αποδοτική λειτουργία του όλου προγράμματος. Μέσω αυτής διαβάζουμε τα τρισδιάστατα μοντέλα .obj και τα φορτώνουμε στην μνήμη κάνοντας και κάποιες απαραίτητες μεταβολές.

Detect Marker Optimized Functions (arDetectMarkerO, arDetectMarker2O...)

Περιλαμβάνει βελτιστοποιημένες εκδόσεις ήδη υπαρχόντων συναρτήσεων του ARtoolkit χωρίς όμως να αλλάζεται η φιλοσοφία και η λειτουργικότητα των συναρτήσεων. Κατά κύριο λόγο οι βελτιστοποιήσεις έχουν να κάνουν με χρήση inline assembly σε συνδυασμό με πράξεις SSE και MMX και βελτιστοποιήσεις υψηλού επιπέδου. Λεπτομέρειες στην επόμενη παράγραφο.

Marker Transformation Functions (arGetTransMatO, arGetTransMatContO)

Περιλαμβάνει βελτιστοποιημένες εκδόσεις ήδη υπαρχόντων συναρτήσεων του ARtoolkit χωρίς όμως να αλλάζεται η φιλοσοφία και η λειτουργικότητα των συναρτήσεων. Κατά κύριο λόγο οι βελτιστοποιήσεις έχουν να κάνουν με χρήση inline assembly σε συνδυασμό με πράξεις SSE και βελτιστοποιήσεις υψηλού επιπέδου. Λεπτομέρειες στην επόμενη παράγραφο.

Monitoring Functions (init_time_file, timer_start, timer_end, timer_print, grab_info)

Βοηθητικές συναρτήσεις για την λειτουργία του προγράμματος:

1. Εκτύπωση σε αρχείο διαφόρων μεταβλητών π.χ. αριθμός marker που αναγνωρίζονται στο τρέχον frame ICMR κτλ.
2. Μέτρηση χρόνου που χρειάζεται για να εκτελεστούν συγκεκριμένες μέθοδοι του προγράμματος.
3. Εκτύπωση χαρακτήρων στην οθόνη βοηθητικού περιεχομένου
4. Εκτύπωση μέσων όρων των διάφορων μεταβλητών.

Rendering Functions (draw_thread, datamalloc, fetch)

Συλλογή συναρτήσεων υπεύθυνων για το τελικό στάδιο του optimization.

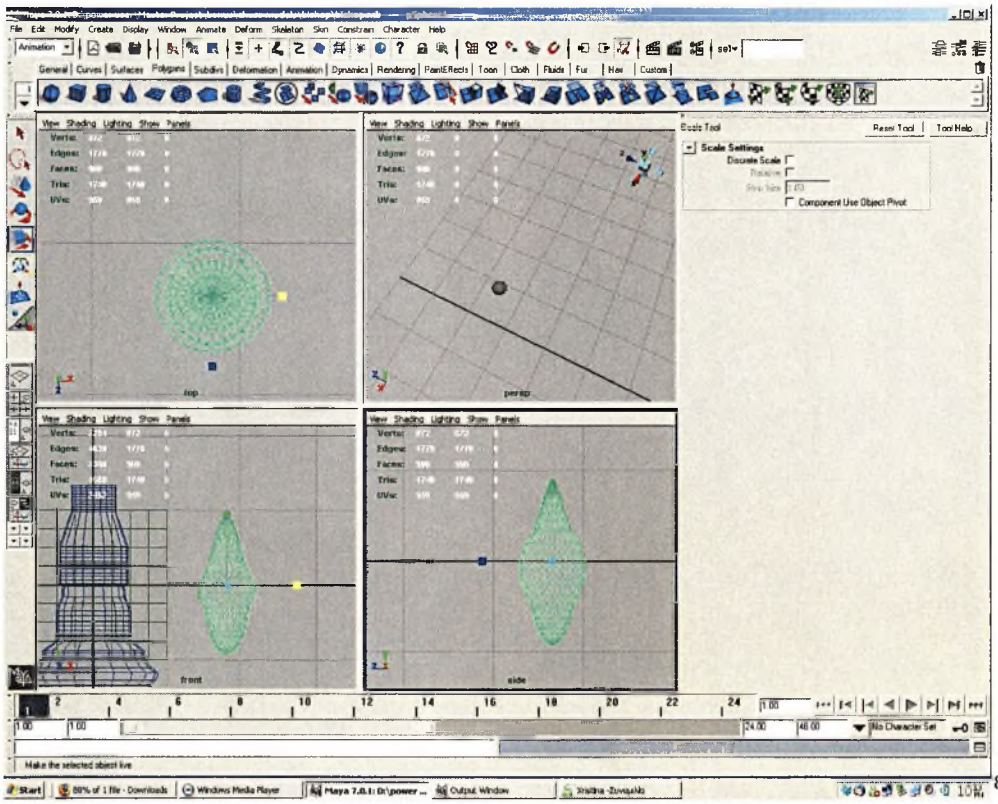
1. Η datamalloc αρχικοποιεί κάποια από τα δεδομένα που περνάνε στο thread που ως επί το πλείστον δεν αλλάζουν κατά την διάρκεια εκτέλεσης του προγράμματος.
2. Η fetch εκτελείται πάντα προτού στείλουμε μήνυμα στην drawthread (για να κάνει rendering ένα αντικείμενο) αρχικοποιώντας τα μέλη μιας δομής η οποία μετά περνιέται στο thread σαν όρισμα.

3. Η drawthread είναι η κύρια συνάρτηση που μας ενδιαφέρει σε αυτό το στάδιο και εκτελείται μέσα από ένα thread. Είναι υπεύθυνη για αρχικοποιήσεις του OpenGL διαχείριση του φωτισμού και φυσικά το object rendering.

Αυτό είναι το τελικό στάδιο που εκτελείται στο εκάστοτε frame. Περνάμε στην drawthread τις συντεταγμένες που πρέπει να ζωγραφιστούν τα αντικείμενα τις οποίες έχουμε πάρει μέσω της προηγούμενης ανάλυσης του frame. Η διαδικασία πραγματοποιείται για κάθε έναν από τους marker που έχουμε εντοπίσει στο frame.

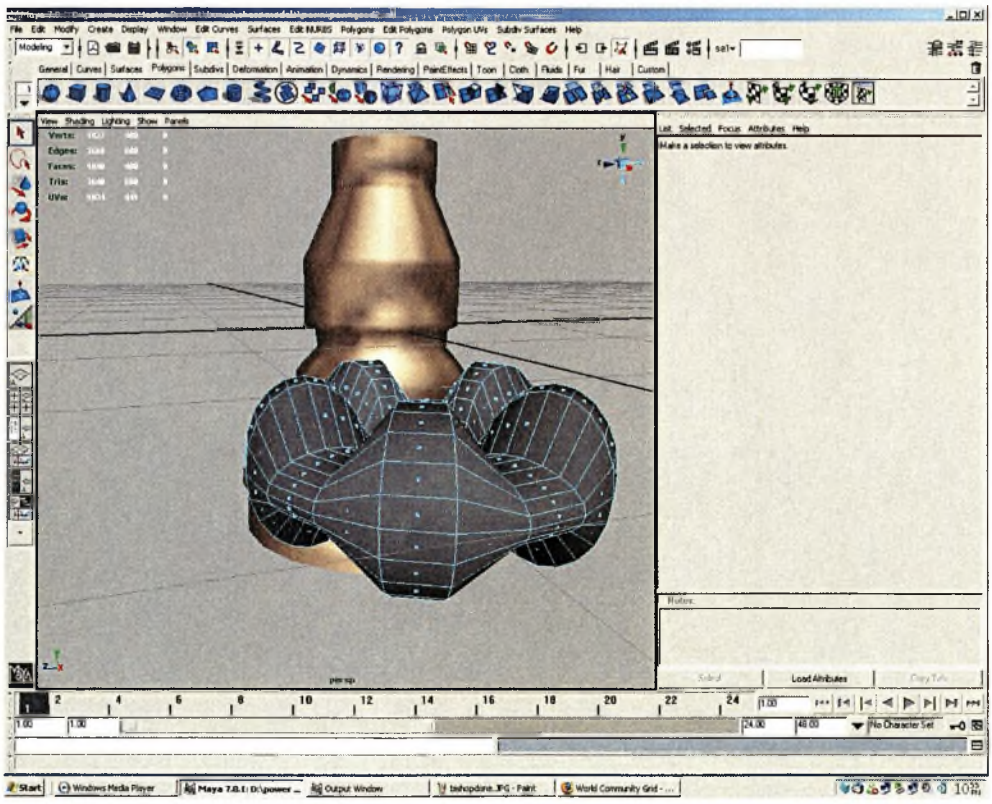
Κωδικοποίηση αρχείων και δημιουργία τρισδιάστατων μοντέλων

Τα αρχεία που χρησιμοποιήθηκαν στην εφαρμογή μας ήταν κατά κύριο λόγο τρισδιάστατα μοντέλα τύπου *.obj. Τα μοντέλα αυτά όπως προαναφέρθηκε σε προηγούμενο κεφάλαιο δημιουργήθηκαν από τον συγγραφέα στο πρόγραμμα Maya Learning Edition και το Blender 3D. Παραθέτουμε μερικά screenshots κατά την δημιουργία τους:

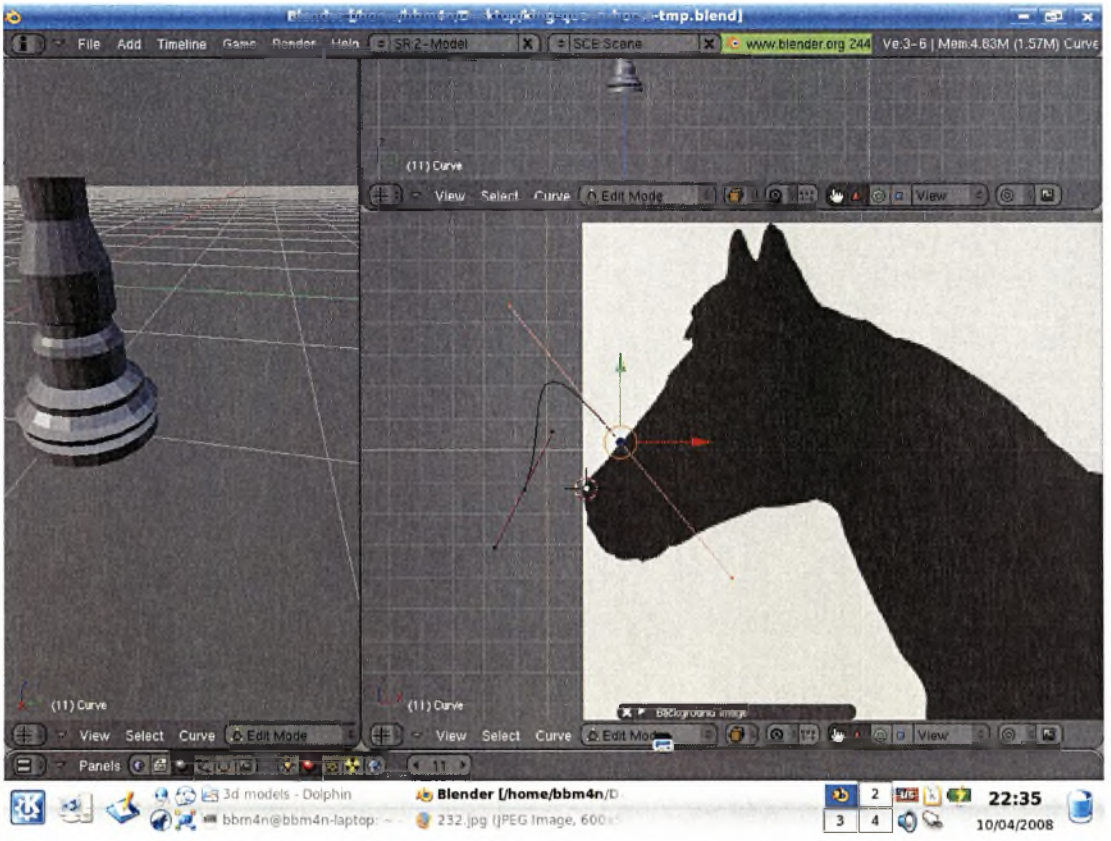


Σχεδίαση ενός από τα τρισδιάστατα μοντέλα (συγκεκριμένα του αξιωματικού)

Σχεδίαση του πιονιού του πύργου.



Ο σχεδιασμός του αλόγου, στο blender.



Να υπενθυμίσουμε πως τα τρισδιάστατα μοντέλα φτιάχτηκαν μόνο με meshes και όχι με άλλους τρόπους, όπως επίσης δεν χρησιμοποιήθηκαν και textures. Βέβαια πρέπει να απαντήσουμε στο κρίσιμο ερώτημα του γιατί χρειάζεται να φτιάξουμε εμείς τα μοντέλα και όχι πολύ απλά να τα πάρουμε έτοιμα, αφού υπάρχει μεγάλος αριθμός από ελεύθερα μοντέλα που ο καθένας μπορεί να τα χρησιμοποιήσει, χωρίς να χρειάζεται να τα αγοράσει. Για να απαντηθεί αυτή η ερώτηση χρειάζεται να κάνουμε μια ανάλυση του πώς ακριβώς λειτουργεί ένα μοντέλο, και τι ακριβώς αποθηκεύεται στο εκάστοτε αρχείο *.obj.

Οποιοδήποτε πρόγραμμα σχεδίασης τρισδιάστατων μοντέλων δεν είναι παρά απλά ένα GUI το οποίο κρύβει την πολυπλοκότητα του μέσω ενός ωραίου γραφικού περιβάλλοντος. Ουσιαστικά παρέχει έτοιμα εργαλεία και μεθόδους για ενέργειες οι οποίες γίνονται συχνά από τους 3d modelers. Μετά τα μοντέλα μπορούμε να τα σώσουμε σε τρισδιάστατα format, με format της επιλογής μας σε αυτήν την εργασία το WAVEFRONT OBJ. Το format αυτό είναι από τα πιο απλά και περιλαμβάνει μια λίστα από vertex coordinates, normal coordinates κ.τ.λ.

απόσπασμα αρχείου WAVEFRONT OBJ

//

.....

v -0.016483 1.094048 -0.010545

v -0.019377 1.094048 -0.004516

v -0.020374 1.094048 0.002168

v -0.019377 1.094048 0.008852

v -0.016483 1.094048 0.014881

v -0.011976 1.094048 0.019666

v -0.006296 1.094048 0.022739

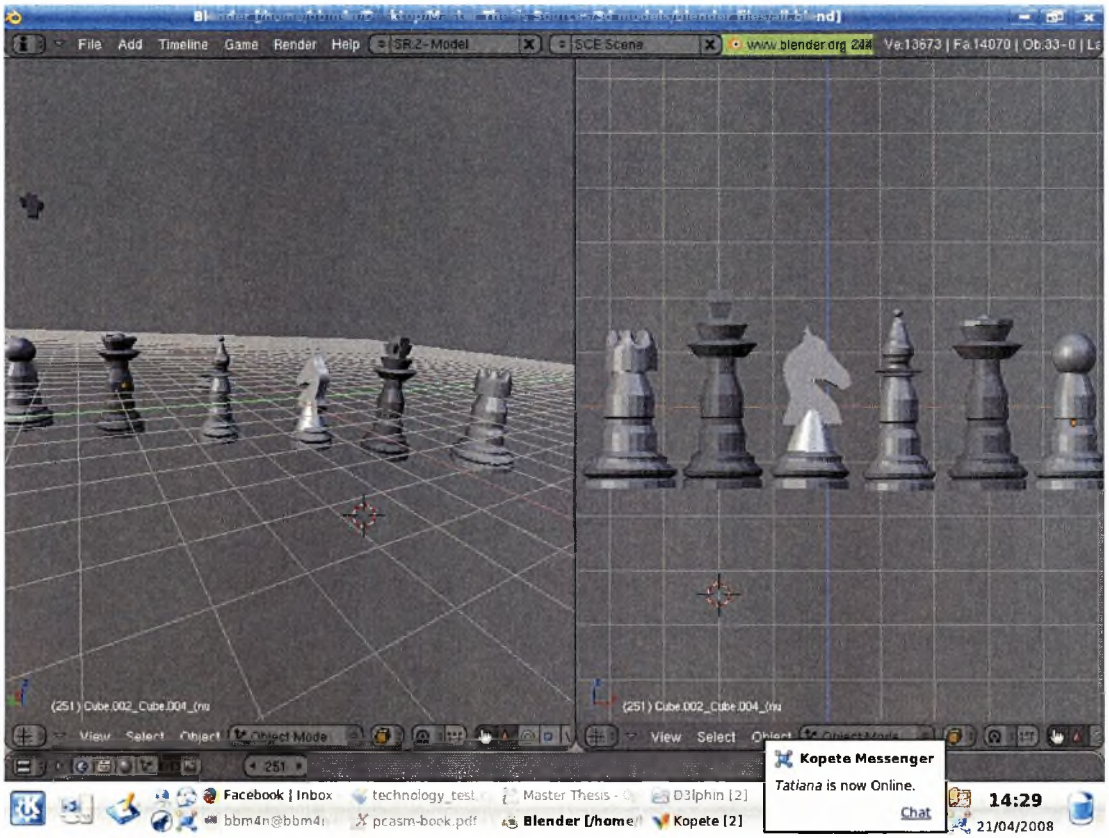
.....

//

Τα αρχεία των μοντέλων λοιπόν γίνονται export σε OBJ format και σε μορφή ASCII, έτσι ώστε να διαβάζονται από την βιβλιοθήκη μας όπως τα αρχεία κειμένου. Θα μπορούσαμε βέβαια για ταχύτητα και μέγεθος να τα διαβάζουμε και σε binary format αλλά αυτό κρίθηκε εκτός του κυρίως θέματος της εργασίας. Η λίστα αυτή με τις συντεταγμένες του μοντέλου διαβάζεται από την βιβλιοθήκη που χρησιμοποιούμε (glm) και προτού μετατραπεί σε display list περνάει μέσα από το OpenGL **το οποίο μετά αναλαμβάνει να την ζωγραφίσει ακμή-ακμή**. Γι΄αυτό στην περίπτωση μας, που θέλουμε μέγιστη ταχύτητα είναι απολύτως σημαντικός το vertex count, facet count, κτλ. Αυτό αιτιολογεί το ότι ο συγγραφέας δημιούργησε ο ίδιος τα μοντέλα του, και δεν κατέφυγε σε έτοιμα, με σκοπό να διατηρήσει χαμηλά τους παραπάνω αριθμούς προσπαθώντας όμως παράλληλα και σε μία ποιοτική απεικόνιση των μοντέλων του σκακιού.

Το vertex, facet, object count όπως παρουσιάζεται στο blender

Όλα τα μοντέλα που δημιουργήθηκαν σε ένα γενικό screen shot.



Πρέπει επίσης να επιστήσουμε την προσοχή στο σύστημα αξόνων που χρησιμοποιούν τα προγράμματα τρισδιάστατης σχεδίασης σε σχέση με αυτό που χρησιμοποιούμε στη απεικόνιση. Σε περίπτωση λανθασμένου συστήματος συντεταγμένων ενδέχεται να τα μοντέλα να απεικονίζονται πλάγια λόγω τού άξονα z.

Υλοποίηση

Εδώ λέμε ότι θα συζητήσουμε λεπτομερώς θέματα υλοποίησης του συστήματος.

Λεπτομέρειες υλοποίησης

Εδώ περιγράφουμε λεπτομερώς θέματα της διπλωματικής που έχουν τεχνικό ή αλγοριθμικό ενδιαφέρον. Για παράδειγμα, κάποια μέθοδος σε κάποια κλάση ενδεχομένως να υλοποιεί έναν πολύπλοκο αλγόριθμο. Εδώ είναι το κατάλληλο σημείο για την περιγραφή του.

Προσδιορίστε επομένως τα θέματα αυτά, βάλτε μια ενότητα για κάθε ένα και περιγράψτε τα αναλυτικά. Η περιγραφή μπορεί να γίνει βάζοντας κομμάτια κώδικα ή ψευδοκώδικα, και περιγράφοντάς τα με λόγια. Μην ξεχνάτε να δίνετε πάντα παραδείγματα για το πώς τρέχει ένα κομμάτι κώδικα π.χ. για έναν αλγόριθμο.

Βελτιστοποιήσεις SSE

Σε αυτό το σημείο θα κληθούμε να εξηγήσουμε ένα από τα σημεία που έγινε βελτιστοποίηση καθώς όλα τα υπόλοιπα έγιναν με παρόμοιο τρόπο. Μια λίστα όλων των βελτιστοποιήσεων για το στάδιο του marker detection δίνεται στο τέλος της παραγράφου.

Αρχικός κώδικας σε ένα βρόγχο.

Έχουμε τον εξής κώδικα αρχικά:

```
for( i = 0; i < prev_num; i++ ) {
    for( j = 0; j < wmarker_num; j++ ) {
// Line 1022
rarea = (double)prev_info[i].marker.area/double)wmarker_info[j].area;
// Line 1023
if( rarea < 0.7 || rarea > 1.43 ) continue;
// Line 1030
rlen = ( (wmarker_info[j].pos[0] - prev_info[i].marker.pos[0])
      * (wmarker_info[j].pos[0] - prev_info[i].marker.pos[0])
      + (wmarker_info[j].pos[1] - prev_info[i].marker.pos[1])
       *(wmarker_info[j].pos[1] - prev_info[i].marker.pos[1]))

      / wmarker_info[j].area;
```

```
if( rlen < 0.5 ) break;
        }

if( j == wmarker_num ) {
wmarker_info[wmarker_num] = prev_info[i].marker;
wmarker_num++;

        }
}
```

Αυτό που μας ενδιαφέρει είναι οι πράξεις που γίνονται για την παραγωγή της τιμής του rlen (με κίτρινο χρώμα). Σε αυτό το σημείο επιλέγουμε στο visual studio να δούμε το assembler output για το συγκεκριμένο αρχείο (ardetectmarkerO.c).

Με κίτρινο παρακάτω βλέπουμε την assembly που αντιστοιχει ο παραπάνω κώδικας.

```
.....
; Line 1022
        mov    eax, DWORD PTR _i$[ebp]
        imul   eax, 208                        ; 000000d0H
        fild   DWORD PTR _prev_info[eax]
        mov    ecx, DWORD PTR _j$[ebp]
        imul   ecx, 200                        ; 000000c8H
        mov    edx, DWORD PTR _wmarker_info
        fidiv  DWORD PTR [edx+ecx]
        fstp   QWORD PTR _rarea$[ebp]
; Line 1023
        fld    QWORD PTR __real@3fe6666666666666
        fcomp  QWORD PTR _rarea$[ebp]
        fnstsw      ax
        test   ah, 65                                  ; 00000041H
        je     SHORT $LN3@arDetectMa@2
        fld    QWORD PTR __real@3ff6e147ae147ae1
        fcomp  QWORD PTR _rarea$[ebp]
        fnstsw      ax
        test   ah, 5
        jp     SHORT $LN4@arDetectMa@2
; Line 1030
        mov    eax, DWORD PTR _j$[ebp]
        imul   eax, 200                        ; 000000c8H
        mov    ecx, DWORD PTR _i$[ebp]
        imul   ecx, 208                        ; 000000d0H
        mov    edx, DWORD PTR _wmarker_info
        fld    QWORD PTR [edx+eax+24]
        fsub   QWORD PTR _prev_info[ecx+24]
        mov    eax, DWORD PTR _j$[ebp]
        imul   eax, 200                        ; 000000c8H
        mov    ecx, DWORD PTR _i$[ebp]
```

```
        imul    ecx, 208                        ; 000000d0H
        mov     edx, DWORD PTR _wmarker_info
        fld     QWORD PTR [edx+eax+24]
        fsub    QWORD PTR _prev_info[ecx+24]
        fmulp ST(1), ST(0)
        mov     eax, DWORD PTR _j$[ebp]
        imul    eax, 200                        ; 000000c8H
        mov     ecx, DWORD PTR _i$[ebp]
        imul    ecx, 208                        ; 000000d0H
        mov     edx, DWORD PTR _wmarker_info
        fld     QWORD PTR [edx+eax+32]
        fsub    QWORD PTR _prev_info[ecx+32]
        mov     eax, DWORD PTR _j$[ebp]
        imul    eax, 200                        ; 000000c8H
        mov     ecx, DWORD PTR _i$[ebp]
        imul    ecx, 208                        ; 000000d0H
        mov     edx, DWORD PTR _wmarker_info
        fld     QWORD PTR [edx+eax+32]
        fsub    QWORD PTR _prev_info[ecx+32]
        fmulp ST(1), ST(0)
        faddp ST(1), ST(0)
        mov     eax, DWORD PTR _j$[ebp]
        imul    eax, 200                        ; 000000c8H
        mov     ecx, DWORD PTR _wmarker_info
        fidiv DWORD PTR [ecx+eax]
        fstp    QWORD PTR _rlen$[ebp]
.....
```

Πρόκειται για 36 γραμμές assembly οι οποίες στην περίπτωση του παραδείγματος noopt test run2 θα εκτελούνταν (σε περίπτωση ορθής λειτουργίας) 12x12=144 φορές . Αυτό γιατί το prev_num και το wmarker_num εκφράζουν τον αριθμό των marker που βρέθηκαν στο προηγούμενο frame και τον τρέχον αριθμό marker αντίστοιχα. Συνολικά έχουμε 12x12x36= 5184 εντολές assembly για αυτή την εκχώρηση τιμής.

Βελτιστοποίηση κώδικα με χρήση SSE intrinstics

Η πρώτη λύση που δοκιμάστηκε ήταν ο χειρισμός με compiler intrinstics και συγκεκριμένα τα SSE intrinstics. Τα compiler intinstics είναι συναρτήσεις που κατά την κλήση τους αποφεύγουμε το overhead του function call. Τα SSE intrinstics μπορούν αφαιρετικά να θεωρηθούν ως wrappers συναρτήσεων που κάνουν πράξεις πάνω σε καταχωρητές SSE. **Σε αντίθεση με την κοινή πεποίθηση ότι παράγουν αντίστοιχη βελτιστοποίηση με inline assembly θα δείξουμε ότι χωρίς optimization έχουν χειρότερη συμπεριφορά στο παράδειγμα μας.**

Ο παραπάνω κώδικας με αντίστοιχο τρόπο προσπαθώντας να εκτελέσουμε παράλληλα τις 4 αφαιρέσεις 2 πολλαπλασιασμούς και μια πρόσθεση θα ήταν:

```
//intrinstic variables declarations
__m128 tmp,tmp2,tres;
//declaring 16 byte aligned data
__declspec(align(16)) float p[4];

....

// Line 1038
tmp=_mm_set_ps(wmarker_info[j].pos[0],wmarker_info[j].pos[1],0,0);
// Line 1039
tmp2=_mm_set_ps(prev_info[i].marker.pos[0],prev_info[i].marker.pos[1]
,0,0);
// Line 1040
tres=_mm_sub_ps(tmp,tmp2);
// Line 1042
tres=_mm_mul_ps(tres,tres);
// Line 1043
tmp2=tres;
// Line 1044
tres=_mm_shuffle_ps(tres,tres,_MM_SHUFFLE(2,2,2,2));   // 10 10 10 10
// Line 1045
tres=_mm_add_ps(tres,tmp2);
// Line 1047
_mm_store_ps(&p,tres);
// Line 1051
rlen=p[3]/wmarker_info[j].area;

....
```

Με κίτρινο παρακάτω βλέπουμε την assembly που αντιστοιχει ο παραπάνω κώδικας.

```
; Line 1038
        movss   xmm0, DWORD PTR __real@00000000
        movss   DWORD PTR tv1254[ebp], xmm0
        movss   xmm0, DWORD PTR __real@00000000
        movss   DWORD PTR tv1254[ebp+4], xmm0
        mov     eax, DWORD PTR _j$[ebp]
        imul    eax, 200                        ; 000000c8H
        mov     ecx, DWORD PTR _wmarker_info
        fld     QWORD PTR [ecx+eax+32]
        fstp    DWORD PTR tv1250[ebp]
        fld     DWORD PTR tv1250[ebp]
        fstp    DWORD PTR tv1259[ebp]
        movss   xmm0, DWORD PTR tv1259[ebp]
        movss   DWORD PTR tv1254[ebp+8], xmm0
        mov     edx, DWORD PTR _j$[ebp]
        imul    edx, 200                        ; 000000c8H
        mov     eax, DWORD PTR _wmarker_info
        fld     QWORD PTR [eax+edx+24]
```

```
        fstp    DWORD PTR tv1253[ebp]
        fld     DWORD PTR tv1253[ebp]
        fstp    DWORD PTR tv1255[ebp]
        movss   xmm0, DWORD PTR tv1255[ebp]
        movss   DWORD PTR tv1254[ebp+12], xmm0
        movaps  xmm0, XMMWORD PTR tv1254[ebp]
        movaps  XMMWORD PTR _tmp$[ebp], xmm0
; Line 1039
        movss   xmm0, DWORD PTR __real@00000000
        movss   DWORD PTR tv1271[ebp], xmm0
        movss   xmm0, DWORD PTR __real@00000000
        movss   DWORD PTR tv1271[ebp+4], xmm0
        mov     eax, DWORD PTR _i$[ebp]
        imul    eax, 208                        ; 000000d0H
        fld     QWORD PTR _prev_info[eax+32]
        fstp    DWORD PTR tv1268[ebp]
        fld     DWORD PTR tv1268[ebp]
        fstp    DWORD PTR tv1276[ebp]
        movss   xmm0, DWORD PTR tv1276[ebp]
        movss   DWORD PTR tv1271[ebp+8], xmm0
        mov     ecx, DWORD PTR _i$[ebp]
        imul    ecx, 208                        ; 000000d0H
        fld     QWORD PTR _prev_info[ecx+24]
        fstp    DWORD PTR tv1270[ebp]
        fld     DWORD PTR tv1270[ebp]
        fstp    DWORD PTR tv1272[ebp]
        movss   xmm0, DWORD PTR tv1272[ebp]
        movss   DWORD PTR tv1271[ebp+12], xmm0
        movaps  xmm0, XMMWORD PTR tv1271[ebp]
        movaps  XMMWORD PTR _tmp2$[ebp], xmm0
; Line 1040
        movaps  xmm0, XMMWORD PTR _tmp2$[ebp]
        movaps  xmm1, XMMWORD PTR _tmp$[ebp]
        subps   xmm1, xmm0
        movaps  XMMWORD PTR _tres$[ebp], xmm1
; Line 1042
        movaps  xmm0, XMMWORD PTR _tres$[ebp]
        movaps  xmm1, XMMWORD PTR _tres$[ebp]
        mulps   xmm1, xmm0
        movaps  XMMWORD PTR _tres$[ebp], xmm1
; Line 1043
        movaps  xmm0, XMMWORD PTR _tres$[ebp]
        movaps  XMMWORD PTR _tmp2$[ebp], xmm0
; Line 1044
```

```
        movaps xmm0, XMMWORD PTR _tres$[ebp]
        movaps xmm1, XMMWORD PTR _tres$[ebp]
        shufps  xmm1, xmm0, 170                 ; 000000aaH
        movaps XMMWORD PTR _tres$[ebp], xmm1
; Line 1045
        movaps xmm0, XMMWORD PTR _tmp2$[ebp]
        movaps xmm1, XMMWORD PTR _tres$[ebp]
        addps   xmm1, xmm0
        movaps XMMWORD PTR _tres$[ebp], xmm1
; Line 1047
        movaps xmm0, XMMWORD PTR _tres$[ebp]
        movaps XMMWORD PTR _p$[ebp], xmm0
; Line 1051
        mov     eax, DWORD PTR _j$[ebp]
        imul    eax, 200                        ; 000000c8H
        mov     ecx, DWORD PTR _wmarker_info
        fild    DWORD PTR [ecx+eax]
        fdivr   DWORD PTR _p$[ebp+12]
        fstp    QWORD PTR _rlen$[ebp]
```

Μπορούμε να κάνουμε μια μεγάλη σειρά παρατηρήσεων σχετικά με τον παραπάνω κώδικα:

- Οι αριθμητικές πράξεις πραγματοποιούνται σαφώς γρηγορότερα με τις 4 αφαιρέσεις τους 2 πολλαπλασιασμούς και την πρόσθεση να αντιστοιχούν το καθένα σε 4 εντολές assembly.
- Τα προβλήματα εντοπίζονται στις εντολές _mm_set_ps και _mm_store_ps. Με το πρώτο το οποίο καλούμε 2 φορές για να γεμίσουμε τους 2 καταχωρητές να αντιστοιχεί σε 24+22=46 γραμμές συνολικά κώδικα assembly. Η εντολή _mm_store_ps αντιστοιχεί σε 6 εντολές assembly.
- Στο MSDN Library αναφέρεται πως η _mm_set_ps είναι composite intrinstic και δεν αντιστοιχεί σε μια μόνο εντολή assembly αλλά σε μια ακολουθία εντολών.

Συμπέρασμα: **Τα SSE intrinstics στην περίπτωση μας αυξάνουν τον αριθμό εντολών αντί να τον μειώνουν δημιουργώντας κώδικα 72 εντολών assembly.**

Τελική βελτιστοιποίηση με χρήση καταχωρητών SSE και inline assembly

Αντικαθιστούμε τον κώδικα της ανάθεσης τιμής του rlen με το εξής:

```
__asm{//ok tested runs correctly
                                //push eax
      //save eax state for any case
mov eax, j
imul eax, size2
movups xmm0, [eax+wmarker_info]
movaps xmm2, xmm0
shufps xmm0, xmm0,0x5       //5(hex)= 00 00 01 01(bin)
mov eax, i
imul eax, size3
```

```
movups xmm1, prev_info[eax]
shufps xmm1, xmm1,0x5      //5(hex)= 00 00 01 01(bin)
subps  xmm0, xmm1
mulps  xmm0, xmm0
movaps xmm1, xmm0
shufps xmm1, xmm1,0xAA     //AA(hex) = 10 10 10 10(bin)
addss  xmm0, xmm1
shufps xmm2, xmm2,0xAA     //AA(hex) = 10 10 10 10(bin)
mulss  xmm0, xmm2             // we cannot divide by int cause slow
//op emms must be called so we multiply with 1/area wich is float
movss rlen, xmm0

     }
```

Παρακάτω βλέπουμε την assembly που αντιστοιχει ο παραπάνω κώδικας.

```
; Line 438
        mov     eax, DWORD PTR _j$[ebp]
; Line 439
        imul    eax, DWORD PTR _size2
; Line 440
        movups xmm0, XMMWORD PTR _wmarker_info[eax]
; Line 441
        movaps xmm2, xmm0
; Line 442
        shufps  xmm0, xmm0, 5
; Line 443
        mov     eax, DWORD PTR _i$[ebp]
; Line 444
        imul    eax, DWORD PTR _size3
; Line 445
        movups xmm1, XMMWORD PTR _prev_info[eax]
; Line 446
        shufps  xmm1, xmm1, 5
; Line 447
        subps   xmm0, xmm1
; Line 448
        mulps   xmm0, xmm0
; Line 449
        movaps xmm1, xmm0
; Line 450
        shufps  xmm1, xmm1, -86                         ; ffffffaaH
; Line 451
        addss   xmm0, xmm1
; Line 452
        shufps  xmm2, xmm2, -86                         ; ffffffaaH
; Line 453
        mulss   xmm0, xmm2
```

; Line 454

movss XMMWORD PTR _rlen$[ebp], xmm0

Σε αυτό το σημείο ας εξηγήσουμε και σχηματικά πως λειτουργεί ο παραπάνω κώδικας.

mov eax, j: Αποθηκεύουμε την τιμή του j στον καταχωρητή eax.

imul eax, size2 : eax= eax*size2. Πολλαπλασιάζουμε τον καταχωρητή eax με size2όπου size2 το μέγεθος σε BYTES του κάθε στοιχείου

movups xmm0, [eax+wmarker_info]:

καταχωρητής xmm0:

wmarker_info[j] .pos[0]	wmarker_info[j] .pos[1]	1/area	

127 95 63 31 0

movaps xmm2, xmm0

καταχωρητής xmm2:

wmarker_info[j] .pos[0]	wmarker_info[j] .pos[1]	1/area	

127 95 63 31 0

shufps xmm0, xmm0,0x5

καταχωρητής xmm0:

wmarker_info[j] .pos[0]	wmarker_info[j] .pos[0]	wmarker_info[j] .pos[1]	wmarker_info[j] .pos[1]

127 95 63 31 0

mov eax, i

imul eax, size3

movups xmm1, prev_info[eax]

καταχωρητής xmm1:

prev_info[i]. marker.pos[0]	prev_info[i]. marker.pos[1]		

127 95 63 31 0

shufps xmm1, xmm1,0x5

καταχωρητής xmm1:

prev_info[i]. marker.pos[0]	prev_info[i]. marker.pos[0]	prev_info[i]. marker.pos[1]	prev_info[i]. marker.pos[1]

127 95 63 31 0

subps xmm0, xmm1

καταχωρητής xmm0:

wmarker_info[j] .pos[0]- prev_info[i]. marker.pos[0]	wmarker_info[j] .pos[0]- prev_info[i]. marker.pos[0]	wmarker_info[j] .pos[1]- prev_info[i]. marker.pos[1]	wmarker_info[j] .pos[1]- prev_info[i]. marker.pos[1]

127 95 63 31 0

mulps xmm0, xmm0

καταχωρητής xmm0:

(wmarker_info[j] .pos[0]- prev_info[i]. marker.pos[0])^2	(wmarker_info[j] .pos[0]- prev_info[i]. marker.pos[0])^2	(wmarker_info[j] .pos[1]- prev_info[i]. marker.pos[1])^2	(wmarker_info[j] .pos[1]- prev_info[i]. marker.pos[1])^2

127 95 63 31 0

movaps xmm1, xmm0

καταχωρητής xmm1:

(wmarker_info[j] .pos[0]- prev_info[i]. marker.pos[0])^2	(wmarker_info[j] .pos[0]- prev_info[i]. marker.pos[0])^2	(wmarker_info[j] .pos[1]- prev_info[i]. marker.pos[1])^2	(wmarker_info[j] .pos[1]- prev_info[i]. marker.pos[1])^2

127 95 63 31 0

shufps xmm1, xmm1,0xAA

καταχωρητής xmm1:

(wmarker_info[j] .pos[1]- prev_info[i]. marker.pos[1])^2	(wmarker_info[j] .pos[1]- prev_info[i]. marker.pos[1])^2	(wmarker_info[j] .pos[1]- prev_info[i]. marker.pos[1])^2	(wmarker_info[j] .pos[1]- prev_info[i]. marker.pos[1])^2

127 95 63 31 0

addss xmm0, xmm1

καταχωρητής xmm0:

((wmarker_info[j].pos[0]- prev_info[i].marker.pos[0])^2) +((wmarker_info[j].pos[1]- prev_info[i].marker.pos[1])^2)			

127 95 63 31 0

shufps xmm2, xmm2,0xAA

καταχωρητής xmm2:

1/area	1/area	1/area	1/area

127 95 63 31 0

mulss xmm0, xmm2

καταχωρητής xmm0:

((wmarker_info[j].pos[0]- prev_info[i].marker.pos[0])^2) +((wmarker_info[j].pos[1]- prev_info[i].marker.pos[1])^2) *(1/area)			

127 95 63 31 0

movss rlen, xmm0

Παρατηρούμε πως η inline assembly που ξεκινάει από το keyword "__asm" παρουσιάζεται σχεδόν αυτούσια με μικρές αλλαγές. Ο νέος κώδικας είναι 21 γραμμές μόνο. Οπότε θεωρητικά έχουμε 12x12x21=3024 γραμμές assembly. (5184-3024)*(100/5184)=41% γρηγορότερη εκτέλεση για τον συγκεκριμένο βρόγχο.

Βελτιστοποιήσεις MMX

Οι χρήση MMX είναι σαφώς πιο περιορισμένη από τα SSE. Ο λόγος είναι ότι τα MMX λειτουργούν σε δεδομένα τύπου ακεραίου καθώς επίσης είναι απαραίτητο αν χρησιμοποιήσουμε μετά τα MMX την FPU κάνοντας πράξεις σε αριθμούς κινητής υποδιαστολής να παρεμβάλλουμε την εντολή emms.

Η εντολή αυτή είναι απαραίτητη καθώς όπως προαναφέρθηκε στο 3° κεφάλαιο οι καταχωρητές MMX και οι καταχωρητές της FPU έχουν κοινές θέσεις μνήμης. Το πρόβλημα

είναι ότι η εντολή emms είναι σχετικά αργή [MSDN] [Hyde00] οπότε πρέπει να την καλέσουμε λίγες φορές και στα σωστά σημεία. Η πιο κατάλληλη συνάρτηση κρίθηκε η labeling 2.

Αρχικός κώδικας σε ένα βρόγχο

```
for (j = 1; j < lysize - 1; j++, pnt += poff*2, pnt2 += 2) {
      for(i = 1; i < lxsize-1; i++, pnt+=poff, pnt2++) {

....

....

            pnt2_index = ((*pnt2)-1) * 7;
            work2[pnt2_index+0]++;
            work2[pnt2_index+1]+= i;
            work2[pnt2_index+2]+= j;

....

....

      }

}
```

Όπως παραπάνω παρουσιάζουμε την assembly που αντιστοιχεί ο παραπάνω κώδικας

```
; Line 265
      mov   eax, DWORD PTR _pnt2$[ebp]
      movsx ecx, WORD PTR [eax]
      sub   ecx, 1
      imul  ecx, 7
      mov   DWORD PTR _pnt2_index$[ebp], ecx
; Line 266
      mov   eax, DWORD PTR _pnt2_index$[ebp]
      mov   ecx, DWORD PTR _work2$[ebp]
      mov   edx, DWORD PTR [ecx+eax*4]
      add   edx, 1
      mov   eax, DWORD PTR _pnt2_index$[ebp]
      mov   ecx, DWORD PTR _work2$[ebp]
      mov   DWORD PTR [ecx+eax*4], edx
; Line 267
      mov   eax, DWORD PTR _pnt2_index$[ebp]
      mov   ecx, DWORD PTR _work2$[ebp]
      mov   edx, DWORD PTR [ecx+eax*4+4]
      add   edx, DWORD PTR _i$[ebp]
      mov   eax, DWORD PTR _pnt2_index$[ebp]
      mov   ecx, DWORD PTR _work2$[ebp]
      mov   DWORD PTR [ecx+eax*4+4], edx
; Line 268
      mov   eax, DWORD PTR _pnt2_index$[ebp]
```

```
mov    ecx, DWORD PTR _work2$[ebp]
mov    edx, DWORD PTR [ecx+eax*4+8]
add    edx, DWORD PTR _j$[ebp]
mov    eax, DWORD PTR _pnt2_index$[ebp]
mov    ecx, DWORD PTR _work2$[ebp]
mov    DWORD PTR [ecx+eax*4+8], edx
```

Τελική βελτιστοιποίηση με χρήση καταχωρητών MMX και inline assembly

Υπάρχει λόγος που παρουσιάζουμε την γραμμή με το `pnt2_index`. Αλλά πρώτα θα παρουσιάσουμε την λύση απευθείας με inline assembly και MMX χωρίς να αναλύσουμε πιθανή λύση με intrinstics.

```
....
out3.r1=1;//7 asm lines axtra
out3.r2=i;
out4.r1=j;
out4.r2=j;
....
pnt2_index = (((*pnt2)-1) << 2)+(((*pnt2)-1) << 1)+((*pnt2)-1);
__asm{
      mov eax, pnt2_index
      shl eax,2
      movq mm0, [eax+work2]
      movq mm3, out3
      paddd mm3, mm0
      movq [eax],mm3
      movd mm2, [eax+8]
      movq mm4, out4
      paddd mm4, mm2
      movd [eax+8], mm4
      }
....
....
```

Αρχικά έχουμε δημιουργήσει (δεν φαίνεται στον κώδικα) 2 προσωρινές δομές αποθήκευσης:

```
typedef __declspec(align(8)) struct //total size 64 bit
      {
            int r1,r2;
      }output2;
```

Οι δομή είναι aligned έτσι ώστε να ξέρουμε ακριβώς τον χώρο που καταλαμβάνει κάθε στοιχείο της.

Βλέπουμε αρχικά ότι οι 21 εντολές σε assembly έχουν αντικατασταθεί από 10 με αυτήν την μετατροπή. Ενώ στην πρώτη περίπτωση είχαμε 3 add συνολικά αυτά τώρα έχουν πακεταριστεί σε 2 padd. Η αντίστοιχη assembly για τον βελτιστοποιημένο κώδικα είναι:

```
;Line 741
        mov     eax, DWORD PTR _pnt2$[ebp]
        movsx   ecx, WORD PTR [eax]
        mov     edx, DWORD PTR _pnt2$[ebp]
        movsx   eax, WORD PTR [edx]
        lea     edx, DWORD PTR [eax+eax-2]
        lea     eax, DWORD PTR [edx+ecx*4-4]
        mov     ecx, DWORD PTR _pnt2$[ebp]
        movsx   edx, WORD PTR [ecx]
        lea     eax, DWORD PTR [eax+edx-1]
        mov     DWORD PTR _pnt2_index$[ebp], eax
; Line 748
        mov     eax, DWORD PTR _pnt2_index$[ebp]
; Line 749
        shl     eax, 2
; Line 751
        movq    mm0, MMWORD PTR _work2$[ebp+eax]
; Line 752
        movq    mm3, MMWORD PTR _out3
        paddd   mm3, mm0
; Line 754
        movq    MMWORD PTR [eax], mm3
; Line 755
        movd    mm2, DWORD PTR [eax+8]
; Line 756
        movq    mm4, MMWORD PTR _out4
; Line 757
        paddd   mm4, mm2
; Line 758
        movd    DWORD PTR [eax+8], mm4
```

Προτού αναλύσουμε τον λόγο ύπαρξης του line 741 θα ασχοληθούμε λίγο με τις γραμμές που έχουν κίτρινο.

```
mov eax, pnt2_index :μεταφέρουμε την τιμή του pnt2_index στο eax
shl eax,2 : κάνουμε ολίσθηση (αντι για πολλαπλασιασμό)
```

movq mm0, [eax+work2]

καταχωρητής mm0:

work2[pnt2_index+0]	work2[pnt2_index+1]

63 47 31 15 0

movq mm3, out3

καταχωρητής mm3:

1	i

63 47 31 15 0

paddd mm3, mm0

καταχωρητής mm3:

work2[pnt2_index+0]+1	work2[pnt2_index+1]+i

63 47 31 15 0

Κ.λ.π. ...

Θα μπορούσαμε θεωρητικά να περνάμε στον καταχωρητή απ'ευθείας και τις 3 τιμές και να κάνουμε 1 μόνο πράξη πρόσθεσης, ωστόσο αυτό δεν είναι εφικτό καθώς κάθε ακέραιος θα αναπαριστόνταν από 16 bit πληροφορίας και στην δική μας περίπτωση οι τιμές του work2 κατά βάση δεν είναι εφικτό να αναπαρασταθούν από 16 bit. Παραταύτα όμως υπάρχει κέρδος και από την πράξη της πρόσθεσης και από τον τρόπο μεταφοράς δεδομένων αφού τις 2 πρώτες τιμές τις αποθηκεύουμε και τις φορτώνουμε σας ένα quad word (64 bit).

Βελτιστοποιήσεις υψηλού επιπέδου.

Στον προηγούμενο κώδικα μετατρέψαμε το αρχικό `pnt2_index = ((*pnt2)-1) * 7;` στο `pnt2_index = (((*pnt2)-1) << 2)+(((*pnt2)-1) << 1)+((*pnt2)-1);` για ποιόν λόγο όμως το κάναμε αυτό; Τα σύμβολα «<<» και «>>» δηλώνουν αριστερή και δεξιά ολίσθηση αντίστοιχα. Γνωρίζουμε όμως ότι ο πολλαπλασιαμός ή η διαίρεση με δυνάμεις του 2 μπορούν να παρασταθούν με ολισθήσεις.

Δηλαδή:
i=i*2 είναι ισοδύναμο με i=i<<1
i=i*4 είναι ισοδύναμο με i=i<<2
i=i/8 είναι ισοδύναμο με i=i>>3 κτλ

Οπότε η προηγούμενη γραμμή είναι σαν να αναπαριστά την πράξη:
`pnt2_index = ((*pnt2)-1) * 4)+ ((*pnt2)-1) * 2)+ (*pnt2)-1);`

Έτσι βγάζοντας το `pnt2_index` κοινό παράγοντα καταληγουμε στο αρχικό. Ο λόγος που κάνουμε αυτήν την αντικατάσταση καθώς και άλλες παρόμοιες δεν είναι προφανής καθώς η assembly που παράγεται στην βελτιστοποιήμενη περίπτωση είναι πιο πολλές γραμμές. Κοιτάζοντας προσεκτικότερα όμως παρατηρούμε πως η εντολή "imul" έχει

αντικατασταθεί από εντολές "lea" **ο λόγος λοιπόν που γίνεται η αντικατάσταση αυτή είναι ότι αντικαθιστούμε μια σχετικά ακριβή πράξη (πολλαπλασιασμό) με περισσότερες φθηνές πράξεις (προσθέσεις και ολισθήσεις).**

Σε σημεία του κώδικα έγιναν και απλές αντικαστάσεις πολλαπλασιασμών και διαιρέσεων με απλές ολισθήσεις. Γενικά ο συγκεκριμένος compiler που χρησιμοποιήθηκε ανταποκρίθηκε σωστά και ήδη είχε αντικαταστήσει με ολισθήσεις του πολλαπλασιασμούς σε επίπεδο assembly, όμως κρίθηκε σωστό παραταύτα από τον συγγραφέα να προσφύγει σε αυτές τις αντικαταστάσεις καθώς το ARtoolkit προορίζεται να τρέχει κάτω από αρκετά OS και σταθηκε αδύνατον να δοκιμαστεί η συμπεριφορά όλων των compiler που μπορούν να χρησιμοποιηθούν.

Άλλες βελτιστοποιήσεις που χρησιμοποιήθηκαν ήταν το loop unrolling και το loop flipping. Το loop flipping συγκεκριμένα κάνει λιγότερους ελένχους και είναι μια απλή αντικατάσταση του for() με ένα do{..}while(). Σε κάποια σημεία επίσης ήταν εφικτό να γίνουν αλλαγές σε δομές τύπου

```
if(something){
        code1
}
else{
        code2
}
```

και μελετώντας την συχνότητα το πρόγραμμα διαλέγει ένα από τα 2 να μετατραπεί σε:

```
code2
if(!something){
        (undo)code2
        code1
}
```

Με την προϋπόθεση βέβαια ότι το πρόγραμμα συνήθως προτιμάει την είσοδο στο else. Αυτό μας βοηθάει διότι οι μηχανισμοί branch prediction που υπάρχουν συνήθως προτιμούν την πρώτη επιλογή, πιθανό λάθος όμως στο branch prediction έχει μεγάλο penalty που εξαρτάται από το μέγεθος του pipeline του cpu (στην συγκεκριμένη περίπτωση ο Pentium 4 θα είχε σοβαρό πρόβλημα).

Πλατφόρμες και προγραμματιστικά εργαλεία

Όπως και κατά την μελέτη του ARtoolkit η **πλατφόρμα ανάπτυξης** του συστήματος μας παρέμεινε η ίδια. Η πλατφόρμα ανάπτυξης έτρεχε λειτουργικό σύστημα, "Windows XP Professional" με service pack 2. Η κάμερα που χρησιμοποιήθηκε ήταν μια logitech, και

επίσης ο λόγος που προτιμήσαμε τα Windows αντί των linux καθώς προσφέρουν πολύ καλύτερη συμβατότητα σε όλες τις συσκευές συνδεδεμένες σε θύρα USB. Χρησιμοποιήσαμε Microsoft Visual Studio 2005, blender και Maya Learning Edition (τα 2 τελευταία για την σχεδίαση μοντέλων).

Η **πλατφόρμα εκτέλεσης** μπορεί να ποικίλλει καθώς το πρόγραμμα διαθέτει τις απαραίτητες ρουτίνες για την εύρεση των τεχνολογιών που υποστηρίζει το σύστημα όπως την έκδοση του OpenGL και τις τεχνολογίες που υποστηρίζει ο επεξεργαστής MMX,SSE όπως προτείνεται από διάφορες πηγές [Int02]. Στην συγκεκριμένη μέθοδο της assembly δεν παρέχεται κώδικας χειρισμού λάθους σε περίπτωση που ο επεξεργαστής δεν υποστηρίζει την εντολή, αλλά αυτό κρίθηκε εκτός του ενδιαφέροντος της τρέχουσας εργασίας καθώς πρόκειται για πολύ παλιά μοντέλα.

Η εφαρμογή μπορεί να τρέχει σε πληθώρα πλατφορμών αλλά το προτεινόμενο σύστημα είναι ένας επεξεργαστής που να υποστηρίζει SSE , και μια κάρτα γραφικών που να υποστηρίζει σε hardware επίπεδο OpenGL 2.0. Η ύπαρξη επεξεργαστή πολλών πυρήνων μπορεί να βελτιώσει πάρα πολύ την απόδοση καθώς μέρος του προγράμματος έχει συγγραφεί με τέτοιο τρόπο ώστε να εκμεταλλεύεται την παράλληλη επεξεργασία τμημάτων του κώδικα.

Αξιολόγηση

Σε αυτό το κεφάλαιο θα ακολουθήσει η αξιολόγηση του συστήματος που δημιουργήσαμε. Αυτό θα γίνει πειραματικά με παρόμοια μεθοδολογία όπως αυτή που ακολουθήθηκε παραπάνω κατά την αξιολόγηση του ARtoolkit.

Μεθοδολογία ελέγχου

Το σενάριο που χρησιμοποιήσαμε κατά τον έλεγχο και στην πειραματική ανάλυση μας ήταν το μοντέλο του τρισδιάστατου σκακιού. Στο κεφάλαιο 1.2 αναφερόμαστε στους λόγους που επιλέχθηκε το πρόβλημα του σκακιού για τον έλεγχο του τελικού συστήματος.

Αναλυτική παρουσίαση ελέγχου

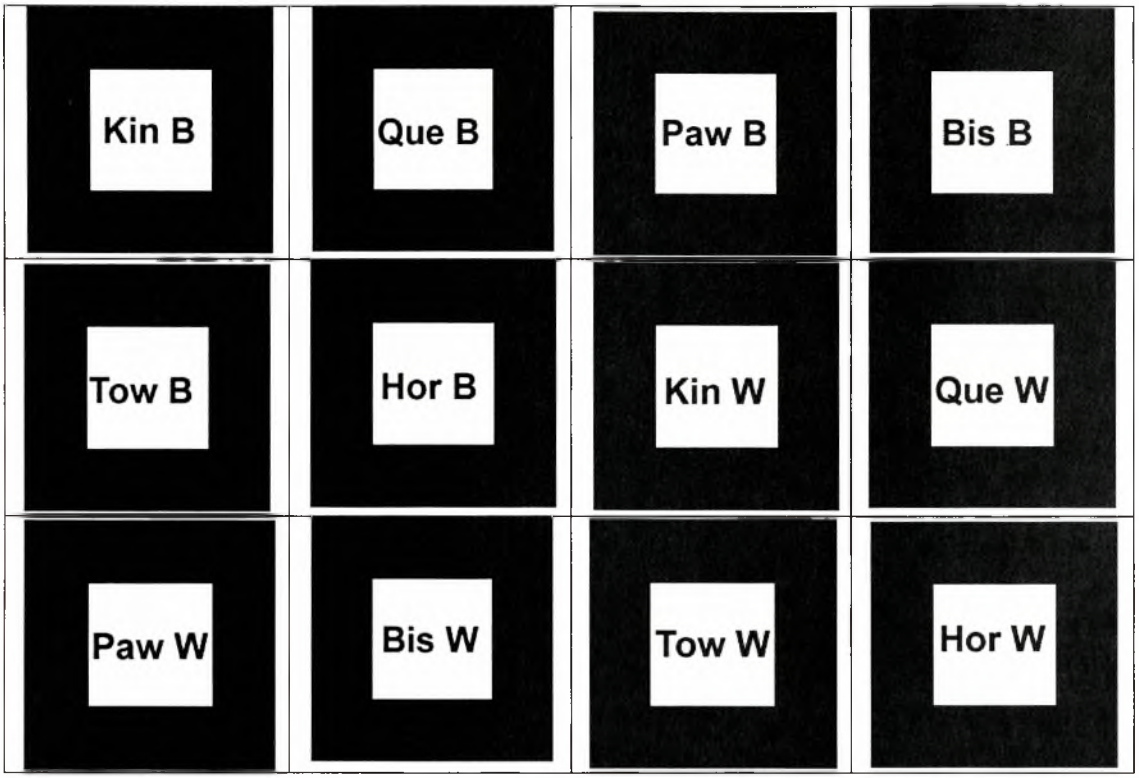
Στην ενότητα αυτή παρουσιάζουμε αναλυτικά τον έλεγχο του συστήματος σύμφωνα με το σενάριο που περιγράφηκε στην προηγούμενη ενότητα.

Benchmarks

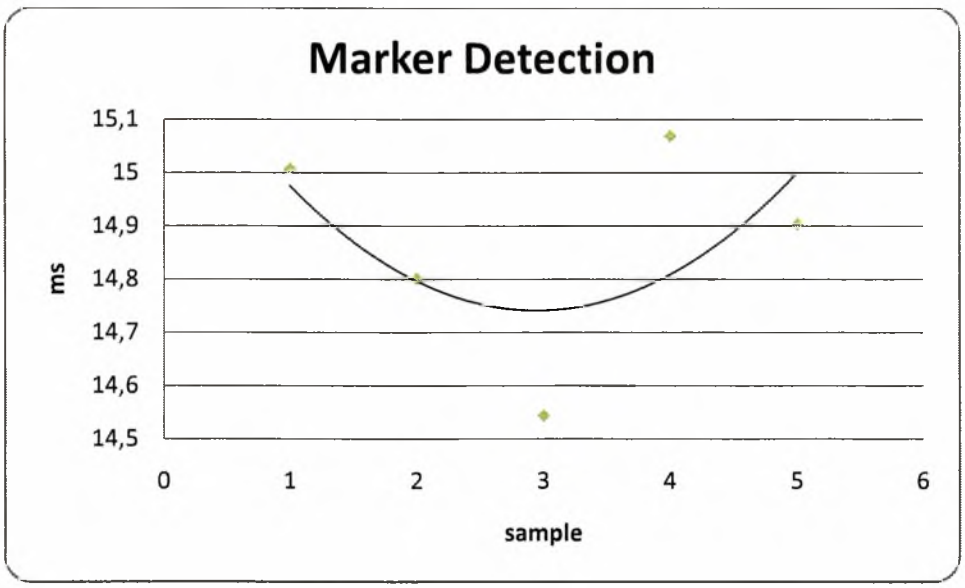
allopt test-run 15sec

Ρυθμίσεις	
χρωματική απόχρωση	RGB
ανάλυση	640x480
αριθμός marker στο frame	12

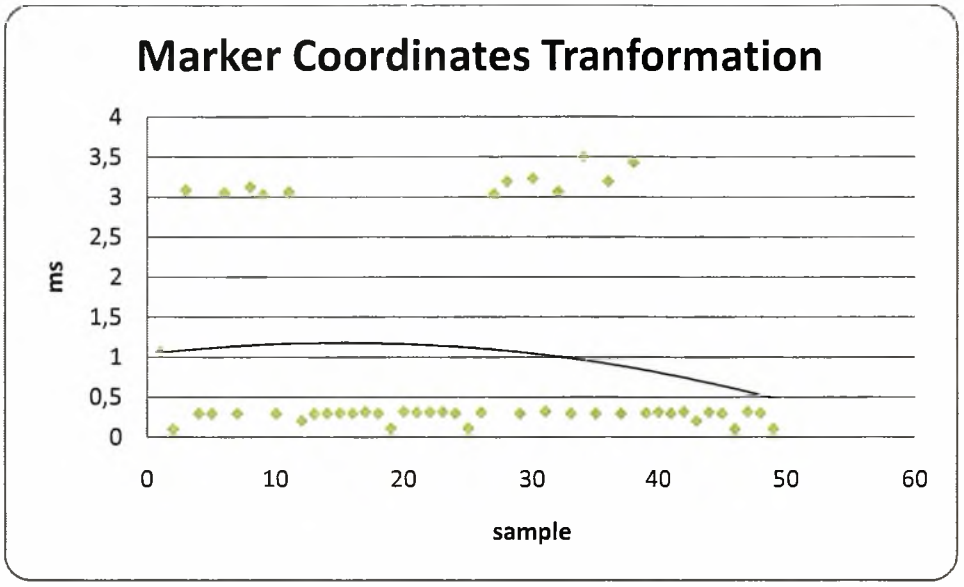
Σχηματική αναπαράσταση frame:



Marker Detection Times:



Marker Coordinates Transformation:



Επεξηγήσεις

Παρατηρούμε ότι το πρόγραμμα τρέχει σαφώς γρηγορότερα τώρα και συγκεκριμένα τα υποσυστήματα για το marker detection και το coordinates transformation. Διαλέξαμε να τρέξουμε μόνο το παράδειγμα με τους 12 markers καθώς όσο περισσότεροι marker υπάρχουν τόσο πιο εμφανείς είναι οι διαφορές (οι βρόγχοι τρέχουν περισσότερες φορές).

Επίλογος

Εδώ λέμε ότι θα συνοψίσουμε την παρουσίαση της διπλωματικής.

Σύνοψη και συμπεράσματα

Στη διπλωματική αυτή έγινε μια ενδελεχής παρουσίαση του τρόπου λειτουργίας των εφαρμογών augmented reality. Παρουσιάστηκαν εφαρμογές της τεχνολογίας και οι δυνατότητες της. Προχωρήσαμε σε διεξοδική ανάλυση της πιο διαδεδομένης βιβλιοθήκης από αυτές, του ARtoolkit. Δημιουργήθηκε μια πληθώρα εφαρμογών μερικές από τις οποίες παρουσιάστηκαν σε αυτό το κείμενο.

Αρχικά δημιουργήσαμε μια εφαρμογή με την ζητούμενη λειτουργικότητα. Τα στάδια διαδικασίας δημιουργίας της αναλύθηκαν διεξοδικά κάτι που μπορεί να χρησιμοποιηθεί και σαν manual για την δημιουργία περαιτέρω εφαρμογών. Παρουσιάζεται αναλυτικά η συμπεριφορά σε 2 ξεχωριστές περιπτώσεις και τα αντίστοιχα γραφήματα για συγκριτικούς λόγους.

Κατά τη δημιουργία της τελικής εφαρμογής έγινε λεπτομερής ανάλυση των προβλημάτων που παρουσιάστηκαν στη μη βελτιστοποιημένη εφαρμογή, ενώ δόθηκαν προτάσεις για τρόπους βελτιστοποίησης. Παρουσιάσαμε ένα μέρος της πληθώρας των προγραμματιστικών επιλογών που δοκιμάστηκαν δίνοντας λεπτομερή αιτιολόγηση γιατί κάποιες λύσεις απορρίφθηκαν. Τέλος εμφανίζονται οι βέλτιστες λύσεις κάθε υποσυστήματος εμφανίζοντας τα συγκριτικά πλεονεκτήματα τους. Το formal verification τμημάτων της εφαρμογής επιβεβαιώνει την ορθότητα των συλλογισμών μας.

Μελλοντικές επεκτάσεις

Ο αναγνώστης έχοντας διαβάσει το παρόν κείμενο έχει συγκεντρώσει γρήγορα την γνώση που χρειάζεται για τον τομέα του Augmented reality μπορώντας γρήγορα να εντοπίσει τυχόν αδυναμίες στην ήδη υπάρχουσα τεχνολογία. Η διπλωματική αυτή μπορεί να χρησιμοποιηθεί και σαν παράδειγμα η case study βελτιστοποίησης συστημάτων καθώς κάνει

λεπτομερή αναφορά σε όλα τα ήδη μεθοδολογιών που χρησιμοποιούνται σε τέτοιες περιπτώσεις. Δεν πρέπει βέβαια να αποκλείσουμε το ενδεχόμενο η διπλωματική να χρησιμοποιηθεί σαν reference manual καθώς παρουσιάζονται κάποια πολύ συγκεκριμένα θέματα σχετικά με SIMD εντολές καθώς και threads που ο αναγνώστης ίσως αγνοεί και είναι πολύ δύσκολο να εντοπίσει references για αυτά. Μελλοντικές επεκτάσεις θα μπορούσαν να περιλαμβάνουν μελέτες πάνω στην δημιουργία markerless augmented reality libraries η ακόμα και ανάπτυξη νέων αλγορίθμων για χρήση αυτού του είδους των εφαρμογών. Δεν μπορούμε να αποκλείσουμε βελτιστοποιήσεις σχετικά με μεταφορά ενός φόρτου επεξεργασίας στην κάρτα γραφικών με μορφή shader αν και παρουσιάζεται μεγάλη πολυπλοκότητα στο σημείο αυτό.

Βιβλιογραφία

[Shan50] Programming a Computer for Playing Chess, Claude E. Shannon, Philosophical magazine 1950

[TCDS00] Bruce Thomas, Ben Close, John Donoghue, John Squires, Phillip De Bondi, Michael Morris and Wayne Piekarski. ARQuake: An Outdoor/Indoor Augmented Reality First Person Application

[WLHJ04] Adrian Woolard, Vali Lalioti, Nicholas Hedleg, Joey Julien, Matt Hammond, Neil Carrigan.Using ARToolkit to prototype Future Entertainment Scenarios

[PT03] Wayne Piekarski, Bruce H. Thomas, Tinmith – Mobile Augmented Reality Modelling Demostration. ISMAR 2003

[PWCG01] Jarrell Pair, Jeff Wilson, Jeff Chastine, Maribeth Gandy.The Duran Duran Project: The Augmented Reality Toolkit in Live performance

[KZK06] Claudio Kimer, Member, IEEE, Ezequiel R. Zorzal, Tereza G. Kirner, Case Studies on the Development of Games Using Augmented Reality
2006 IEEE International Conference on Systems, Man, and Cybernetics
October 8-11, 2006, Taipei, Taiwan

[HNSI06] Yutaka Harada , Napoleon Nazir , Yoshinori Shiote and Tomotaka Ito
Human-machine Collaboration System for Fine Assembly Process
SICE-ICASE International Joint Conference 2006
Oct. 18-21, 2006 in Bexco, Busan, Korea

[LXYF04] Guangyong Li, Ning Xi, Mengmeng Yu, Wai-Keung Fung.
Development of Augmented Reality System for AFM-Based Nanomanipulation. IEEE/ASME TRANSACTIONS ON MECHATRONICS, VOL. 9, NO. 2, JUNE 2004

[JSLJ01] Dr. Paul Jackson, Dr. Joan Ealey-Sawyer, Dr. I-Li Lu, Dr. Stephen Jones

Testing Information Delivery Methods Using Augmented Reality
IEEE and ACM International Symposium on Augmented Reality, 2001

[Fia05] Mark Fiala, Comparing ARTag and ARToolkit Plus Fiducial Marker Systems IEEE International Workshop on Haptic Audio Visual Environments and their Applications 2005

[OXM01] Charles B. Owen, Fan Xiao, Paul Middlin, What is the best fiducial?

[KB99] Hirokazu Kato, Mark Billinghurst, Marker Tracking and HMD calibration for a video-based Augmented Reality Conferencing System , IWAR 1999

[MYN06] Jonathan Mooser, Suya You, Ulrich Neumann, TRICODES: A BARCODE-LIKE FIDUCIAL DESIGN FOR AUGMENTED REALITY MEDIA, 2006 IEEE

[TKO07] Keisuke Tateno, Itaru Kitahara,Yuichi Ohtaa Nested Marker for Augmented Reality, IEEE Virtual Reality Conference 2007

[DBD04] Daniel F. Abawi, Joachim Bienwald, Ralf Dorner, Accuracy in Optical Tracking with Fiducial Markers: An Accuracy Function for ARToolkit IEEE&ACM International Symposium on Mixed and Augmented Reality-ISMAR 2004

[KBP00] H. Kato, M. Billinghurst, I. Poupyrev, K. Imamoto, K.Tachibana,Virtual Object Manipulation on a Table-Top AR Environment , ISAR 2000

[Hyde00] Randall Hyde, The Art of Assembly Language
http://webster.cs.ucr.edu/AoA/index.html (2000)

[Int02] Intel Corporation, IA-32 Intel® Architecture Software Developer's Manual Volume 3: System Programming Guide, 2002

[Int99-2] Intel Corporation, Intel Architecture Software Developer's Manual Volume 2: Instruction Set Reference, 1999

[Int99-1] Intel Corporation, Intel Architecture Software Developer's Manual Volume 1: Basic Architecture, 1999

[Pin05] Thomas Pintaric, DirectShow Video Lib
http://sourceforge.net/projects/dsvideolib/

[OGLSu] Richard S. Wright, Jr. , Benjamin Lipchak , Nicholas Haemel. OpenGL

SuperBible Fourth Edition , Addison Wesley

[OGLRe] Jackie Neider, Tom Davis, and Mason Woo, OpenGL programming guide aka "The Red Book" by Silicon Graphics , Addison Wesley

[OGLBl] OpenGL reference manual aka "The Blue Book" by Silicon Graphics , Addison Wesley

[Fia04] Mark Fiala, ARTag Revision 1.A Fiducial Marker System Using Digital Techniques, National Research Council of Canada, November 2004

[MSDN] Microsoft Corporation MSDN library

[TRIP] Lopez de Ipina, D.TRIPQ a Low Cost Vision-Based Location System for Ubiquitous Computing. In 2001 Workshop on Perceptive User Interfaces. 2001