Δημιουργία Edges και Outline Effects στην επεξεργασία Εικόνας σε ασπρόμαυρη και σε έγρωμη

Τζίτζος Παύλος 26/11/2023 Καθηγητής: Γ. Συρακούλης
Κουμπάνης Αθανάσιος

Project Repository

Σύνδεσμος

Περιεχόμενα

Θεωρητικό Υπόβαθρο

Μετατροπή YUV σε RGB

Για να μετατρέψουμε την εικόνα σε γκρι πρέπει να μετατρεψουμε το χρωματικό χώρο YUV σε RGB. Τα YUV μοντελοποιούν την εικονα χρησιμοποιώντας 3 διαφορετικα στοιχεία :

Y (luminance) για φωτεινότητα, U για το μπλε χρώμα και V για το κόκκινο.

Σε αντίθεση με το YUV μοντέλο , το RGB μοντέλο χρησιμοποιεί 3 στοιχεία για τα χρώματα:

R (red) για το κόκκινο , G (green) για το πράσινο και B (blue) για το μπλέ.

Για τη μετατροπή από YUV σε RGB χρησιμοποιούμε τις εξισώσεις :

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.13983 \\ 1 & -0.39465 & -0.58060 \\ 1 & 2.03211 & 0 \end{bmatrix} \cdot \begin{bmatrix} Y \\ U \\ V \end{bmatrix}$$

Στο σημείο αυτό πρέπει να έχουμε υπόψιν μας ότι τα δύο αυτά μοντέλα έχουν διαφορετικό εύρος στις τιμές των στοιχείων τους. Το YUV παίρνει τις εξής τιμές :

$0 \leq Y \leq 1$ ή $0 \leq Y \leq 255$

$$-0.5 \leq U \leq 0.5$$

$$-0.5 \leq V \leq 0.5$$

Ενώ το RGB κυμαίνεται μεταξύ των : $0 \leq R$, G , $B \leq 255$.

Οπότε μετά από κάθε μετατροπή είναι αναγκαίο να κάνουμε κανονικοποίηση.

Μετατροπή Έγχρωμης RGB εικόνας σε Ασπρόμαυρη

Επειδή θέλουμε να εντοπίσουμε τις ακμές στην εικόνα μας πριν εφαρμόσουμε το γκαουσιανό φίλτρο μετατρέπουμε την εικόνα από έγχρωμη RGB σε ασπρόμαυρη. Υπάρχουν 3 τρόποι με τους οποίους μπορούμε να μετατρέψουμε μια εικόνα σε ασπρόμαυρη.

Μεθοδος Φωτός

$$I_{grayscale} = \frac{min(R,G,B) - max(R,G,B)}{2}$$

Μέθοδος Αριθμητικού Μέσου

$$I_{grayscale} = \frac{R+G+B}{3}$$

Μέθοδος Φωτεινότητας

$$I_{grayscale} = 0.3R + 0.59G + 0.11B$$

Η τελευταία είναι η πιο αποτελεσματική επειδή το μάτι αντιδρά διαφορετικά σε κάθε χρώμα οπότε βάζοντας βάρη σε κάθε τιμή RGB πετυχαίνουμε να προσαρμόσουμε την γκρι εικόνα στο πως την αντιλαμβανόμαστε .

Το εύρος τιμών των ασπρόμαυρων εικόνων κυμαίνεται από 0 έως 255.

Εφαρμογή του Γκαουσιανού Φίλτρου στην Ασπρόμαυρη εικόνα

Επειδή ο σκοπός της εργασίας είναι να αναγνωρίσουμε τις ακμές στην εικόνα και να τις χρωματίσουμε σε επόμενο βήμα, πρέπει να εφαρμόσουμε το γκαουσιανό φίλτρο στην εικόνα. Αυτό γίνεται με συνέλιξη. Επειδή όμως εφαρμόζουμε την συνέλιξη με μάσκα 3x3 στην πρώτη γραμμή θα χρειαστούμε επιπλέον μία γραμμή. Αλλά και μία στήλη στα αριστερά της εικόνας. Στα δεξία της εικόνας αλλά και στο κάτω μέρος της εικόνας θα χρειαστούμε επίσης μια επιπλέον στήλη και γραμμή αντίστοιχα. Για αυτό τον λόγο δημιουργούμε μια νέα εικόνα I_{padded} .

Η επαυξημένη εικόνα είναι :

$$I_{y,padded} = \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & I_{y,grayscale} & \vdots \\ 0 & \cdots & 0 \end{bmatrix}$$

Η συνέλιξη του φίλτρου με την εικόνα είναι :

$$I_{grayscale,filtered} = (G_{kernel} \bigstar I_{grayscale})(x,y) = \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} G_{kernel}(x-i, y-j) I_{grayscale}(i,j)$$

Υπολογισμός της Βαθμίδας της εικόνας

Εφόσον έχουμε εφαρμόσει το γκαουσιανό φίλτρο υπολογίζουμε την βαθμίδα της φιλτραρισμένης εικόνας που αντιπροσωπεύει τον ρυθμό αλλαγής της έντασης της φωτινότητας του κάθε pixel. Η βαθμίδα της εικόνας I ορίζεται ως:

$\nabla I = \begin{bmatrix} I_x & I_y \end{bmatrix}'$, όπου $I_x = \frac{\partial I}{\partial x}$ και $I_y = \frac{\partial I}{\partial y}$.

Για να υπολογίσουμε τις παραγώγους χρησιμποιήσαμε φίλτρα sobel και τα εφαρμόσαμε στην φιλτραρισμένη εικόνα με συνέλιξη. Οι μάσκες των φίλτρων sobel είναι :

$$S_{kernel,x} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$S_{kernel,y} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Υπολογισμός της Έντασης και της Γωνίας των pixel της εικόνας

Έχοντας την βαθμίδα υπολογισμένη μπορούμε να υπολογίσουμε της ένταση και την γωνία του κάθε pixel. Η ένταση αντιπροσωπεύει την ένταση του χρώματος γκρι / λευκό και είναι το μέτρο της βαθμίδας ενώ η γωνία δείχνει την κλίση της κατεύθυνσης της ένατσης πάνω σε ένα x,y επίπεδο , το επίπεδο της εικόνας. Οι τύποι για τον υπολογισμό τους είναι αντίστοιχα:

$$|\nabla I| = \sqrt{{I_x}^2 + {I_y}^2}$$

$$\theta = \arctan\left(\frac{I_x}{I_y}\right)$$

Η ένταση έχει εύρος τιμών μεταξύ 0 και 255.

Η γωνία έχει εύρος τιμών μεταξύ 0 έως 180 μοίρες.

Χρωματισμός των pixel βάση της Έντασης και της Γωνίας

Επειδή σε μια εικόνα μια ακμή μπορεί να δείχνει προς οποιαδήποτε κατεύθυνση χρησιμοποιούμε διαφορετικά χρώματα ανάλογα την γωνία που έχουμε υπολογίσει για εκείνο το pixel και το χρωματίζουμε . Έτσι όλα τα pixel που ανήκουν επάνω στην ακμή θα

έχουν ίδιο χρώμα και η ένταση του χρώματος αυτού εξαρτάται από την ένταση (μέτρο) της κλίσης που υπολογίσαμε προηγουμένως. Ορίζουμε τις εξής περιοχές:

$\leq \theta°$	$\theta° <$	Στρογγυλοποίηση ($\theta°$)	Χρώμα ομάδας	Ακμές
0	22.5	0	κίτρινο	Κατακόρυφες
22.5	67.5	45	πράσινο	Διαγώνιες προς δεξιά
67.5	112.5	90	μπλέ	Οριζόντιες
112.5	157.5	135	κόκκινο	Διαγώνιες προς αριστερά
157.5	180	0	κίτρινο	Κατακόρυφες

Κώδικας C

Ο κώδικας υλοποιήθηκε αρχικά χρησιμοποιώντας το Visual Studio για να μπορέσουμε να κάνουμε debugging εύκολα και το github για να μπορέσουμε να διατηρήσουμε διαφορετικές εκδόσεις λόγω βελτιστοποιήσεων αλλά και για να μπορούμε να δουλέψουμε από διαφορετικές συσκευές στην ίδια εργασία. Η εικόνα με την οποία δοκιμάσαμε τον κώδικα μας είχε διαστάσεις 496x376 και ήταν της μορφής YUV 420.

read_image

1. Το πρώτο βήμα στην εκτέλεση του προγράμματος είναι να φορτώσουμε την εικόνα από την μνήμη. Η συνάρτηση αυτή κάνει αυτή την δουλειά , με παράμετρο filename που έχουμε δηλώσει στην αρχή του αρχείου.

write_id_image

2. Προκειμένου να ελέγξουμε μετά από κάθε αλγόριθμο το αποτέλεσμα γράψαμε συναρτήσεις για αυτόν τον σκοπό. Τα ονόματα αυτών των συναρτήσεων έχουν την παρακάτω δομή:

   void write_*id*_image() ,

   όπου *id* είναι το αναγνωριστικό της μεθόδου που ελέγχουμε.

id1_to_id2

3. Χρειάστηκε σε πολλά σημεία να μετατρέψουμε την εικόνα από μια μορφή σε μια άλλη. Οι συναρτήσεις αυτής της δομής εκτελούν αυτές τις μετατροπές:

   void *id1*_to_*id2*() ,

όπου *id1* είναι το χρωματικό μοντέλο που διαθέτουμε και *id2* είναι το χρωματικό μοντέλο στο οποίο θέλουμε να μετατρέψουμε την εικόνα μας.

grayscale_id

4. Οι συναρτήσεις αυτές χρησιμοποιούνται από την void rgb_to_grayscale(int sel) που ανοίκει στην παραπάνω κατηγορία συναρτήσεων. Η ιδιαιτερότητα αυτής της συνάρτησης είναι δίνει την δυνατότητα να διαλέξουμε ποια μέθοδο επιθυμούμε να χρησιμοποιήσουμε κατά την μετατροπή μιας έγχρωμης rgb εικόνας σε ασπρόμαυρη. Οι συναρτήσεις αυτές έχουν την παρακάτω δομή :

   void grayscale_*id*() ,

   όπου *id1* είναι το χρωματικό μοντέλο που διαθέτουμε και *id2* είναι το χρωματικό μοντέλο στο οποίο θέλουμε να μετατρέψουμε την εικόνα μας.

id1_filter{_id2}

5. Όπως δηλώνει το όνομα filter οι συναρτήσεις αυτές υλοποιούν τις συνελίξεις των εικόνων με τις μάσκες (kernels). Οι συναρτήσεις αυτές έχουν την παρακάτω δομή :

   void *id1*_filter{_*id2*}() ,

   όπου *id1* είναι ο τύπος του φίλτρου που εφαρμόζουμε πχ γκαουσιανό (gaussian) και το *id2* είναι μια παράμετρος που χρησιμοποιείται από τα φίλτρα sobel για να δείξει αν μιλάμε για την παράγωγο ως προς x ή ως προς y.

id_calc

6. Για τους υπολογισμούς που χρησιμοποιούν την βαθμίδα οι συναρτήσεις έχουν την μορφή αυτή :

   void *id*_calc() ,

   όπου *id* είναι ο τύπος της μαθηματικής ιδιότητας της βαθμίδας πχ angle

   Την ίδια δομή έχει και η συνάρτηση που υπολογίζει την βαθμίδα :

   void gradient_calc()

   Μια ματιά στο εσωτερικό αυτής της συνάρτησης :
   Η συνάρτηση αυτή καλεί τις συναρτήσεις φίλτρων sobel (sobel_filter_x και sobel_filter_y)

min{Number} , max{Number}

7. Επειδή σε διάφορα σημεία χρειάστηκε να υπλογισθεί το ελάχιστο ή το μέγιστο στοιχείο μεταξύ 3ων στοιχείων υλοποιήθηκαν οι συναρτήσεις αυτές που δίνουν το αποτέλεσμα αυτό :

   void min{*Number*}() ,

   όπου Number είναι ο αριθμός των στοιχείων που συγκρίνει πχ 2 για να βρει τον ελάχιστο μεταξύ τους.

   Ομοίως για την :

   void max{*Number*}()

   Μια ματιά στο εσωτερικό των συνάρτησεων min3 , max3 :
   Οι συναρτήσεις αυτές καλούν τις συναρτήσεις min2 , max2 αντίστοιχα.

   Μια ματιά στο εσωτερικό των συνάρτησεων min2 , max2 :
   Οι συναρτήσεις αυτές συγκρίνουν τα 2 στοιχεία που τους έχουν δωθεί και επιστρέφουν το μικρότερο ή το μεγαλύτερο από τα 2. Σε περίπτωση που είναι ίσα επιστρέφουν -1. Αυτη την περίπτωση την διαχειρίζονται οι συναρτήσεις που τις κάλεσαν.

find_min, find_max , linear_scaling και scale_magnitude_image

8. Προκειμένου να κάνουμε κανονικοποίηση στις τιμές των pixel της εικόνας εφαρμόζουμε μια απλή ευθεία γραμμή :

$$y = a(x - x_0) + y_0$$

   όπου *id1* είναι ο τύπος του φίλτρου που

   Πριν όμως εφαρμόσουμε την ευθεία γραμμή πρέπει να βρούμε το ελάχιστο και το μέγιστο των τιμών των pixel για αυτό χρησιμοποιούμε τις συναρτήσεις find_min() και find_max().

scale_rgb_image, find_min_{id}, find_max_{id}

9. Όταν μετατρέπουμε την εικόνα από YUV σε RGB δεν αρκεί να βρούμε τις τιμές των χρωμάτων RGB μόνο. Πρέπει να προσέξουμε το εύρος των τιμών τους να είναι σύμφωνο με την θεωρία όπως έχει αναφερθεί παραπάνω. Για αυτόν τον λόγο η συνάρτηση scale_rgb_image εφαρμόζει κανονικοποίηση πάνω σε κάθε τιμή των RGB

πινάκων. Οι συναρτήσεις find_min_{id} και find_max_{id} βρίσκουν τη μικρότερη και τη μεγαλύτερη τιμή του πίνακα με αναγνωριστικό το id. Το id μπορεί να είναι r , g ή b ανάλογα σε ποιο frame εφαρμόζουμε την αναζήτηση. Η αναζήτηση είναι η πιο απλή αλγοριθμικά συγκρίνει όλα τα στοιχεία μεταξύ τους. Φυσικά υπάρχει περιθώριο αλλαγής.

colour_image

10. Το τελευταίο βήμα του απρογράμματος είναι να βάψει την εικόνα σύμφωνα με την γωνία και την ένταση(μέτρο) της βαθμίδας που έχει κάθε pixel.

Είδη Βελτιστοποίησης

Στην προσπάθεια να γράψουμε το παραπάνω πρόγραμμα χρησιμοποιώντας εργαλεία που μετράνε την κατανάλωση μνήμης , επεξεργαστή , χρόνου , πράξεων στον επεξεργαστή και άλλων χρήσιμων πόρων όπως έχει το Visual Studio αλλά και το Armulator παρατηρήσαμε ότι υπήρχαν υψηλές καταναλώσεις μνήμης και άλλων πόρων. Θα θέλαμε να κάνουμε τον κώδικά μας πιο ποιοτικό και για να το πετύχουμε αυτό έχουμε στη διάθεση μας διάφορες τεχνικές βελτιστοποίησης του κώδικά μας ανάλογα την μετρική που θέλουμε να αυξήσουμε ή να μειώσουμε.

Η ποιότητα κώδικα μπορεί να σημαίνει πολλά πράγματα και μπορεί να μετρηθεί με πολλούς τρόπους και σε διαφορετικά στάδια. Αρκεί να αναφέρουμε μερικά παραδείγματα:

1. Ταχύτητα Εκτέλεσης (Runtime Speed)
2. Μέγεθος Κώδικα
3. Ενέργεια που καταναλώνει ο επεξεργαστής κατά της εκτέλεση

Παρακάτω παρουσιάζονται ορισμένες τεχνικές Βελτιστοποίησης συνοπτικά.

Είδη Βελτιστοποίησης βάση του εύρους(scope) της μεθόδου

1. Τοπική Βελτιστοποίηση (Local Optimization) , που βασίζεται στην μέθοδο Απαρίθμησης Τοπικών Τιμών (Local Value Numbering - LVN) και αφορά μικρά κομμάτια κώδικα που μπορούν να εκτελεστούν σειριακά. Στην μέθοδο αυτή βασίζονται οι παρακάτω :
    a. Αναδήπλωση Σταθερών (Constant folding)
    b. Αλγεβρική Απλοποίηση (Algebraic simplification)

* Οι αλγόριθμοι αυτοί είναι εκτός του εύρους της συγκεκριμένης εργασίας αλλά προστέθηκαν για αναφορά μόνο.

2. Βελτιστοποίηση Περιοχής (Regional or Loop-Level Optimization). Αφορούν περιοχές στο πρόγραμμα που περιλαμβάνουν βρόχους και δομές ελέγχου. Μερικές μέθοδοι είναι :
    a. Απαρίθμηση Υπερ-τοπικών Τιμών (Superlocal Value Numbering), είναι μια επέκταση της LVN
    b. Ξεδίπλωμα Βρόχου (Loop unrolling)

3. Βελτιστοποίηση Προγράμματος (Global Optimization). Λόγω του μεγάλου εύρους (scope) και ότι περιλαμβάνουν περιοχές με βρόχους και δομές ελέγχου οι βελτιστοποιήσεις αυτές εφαρμόζονται κατά το στάδιο της ανάλυσης πριν ξαναγράψουν τον κώδικα.
    a. Εντοπισμός Μη αρχικοποιημένων μεταβλητών με ζωντανές δομές (Finding Uninitialized Variables with Live Sets)
    b. Γενικευμένη Τοποθέτηση Κώδικα (Global Code Placement)

4. Βελτιστοποίηση Διαδικασιών (Interprocedural Optimization)
    a. Αντικατάσταση Inline μεθόδων (Inline Substitution)
    b. Τοποθέτηση Διεργασιών (Procedure Placement)

Βαθμωτή Βελτιστοποίηση (Scalar Optimization)

Η βαθμωτή βελτιστοποίηση αφορά την βελτιστοποίηση του κώδικα που έχει αναλάβει ένα νήμα (thread) ελέγχου. Βασίζεται στις παραπάνω μεθόδους βελτιστοποίησης αλλά και στην Στατική Ανάλυση[ref]. Οι τεχνικές που αφορούν την μέθοδο αυτή είναι:

1. Καθαρισμός Άχριστου και Απροσπέλαστου Κώδικα
2. Μετακίνηση Κώδικα (Lazy Code Motion)
3. Εξιδίκευση Υπολογισμού
4. Απαλοιφή Περιττών Υπολογισμών
5. Ενεργοποίηση άλλων Μετασχηματισμών[ρεφ] , όπως είναι ο μετασχηματισμός loop unswitching που βγάζει έξω από έναν βρόχο τις δομές ελέγχου που δεν μεταβάλλονται (loop-invariant control-flow)

Στο σημείο θα αναφέρουμε τις μεθόδους βελτιστοποίησης που μας ενδιαφέρουν περισσότερο σε αυτή τη φάση της εργασίας:

1. Loop Unrolling
2. Loop Fusion - Loop Fission
3. Loop-Invariant Code Motion (LICM)
4. Strength Reduction
5. Dead Code Elimination
6. Common Subexpression Elimination (CSE)
7. Function Inlining – Inline Expansion

8. Constant Folding
9. Copy Propagation
10. Code Motion
11. Fast I/O
12. Global Common Subexpression Elimination

Βελτιστοποιημένος Κώδικας και Μετρήσεις

Για να πετύχουμε υψηλή ποιότητα στον κώδικα αρχικά θέλαμε να είναι όλες οι συναρτήσεις inline σαν πρώτο βήμα. Όμως αυτό δεν είναι εφικτό σε όλες για αυτό και τοποθετήσαμε μόνο τα βασικά blocks μέσα στην main. Το επόμενο βήμα είναι να πετύχουμε βελτιστοποίηση στην κατανάλωση του επεξεργαστή για και ειδικά στα πιο κρίσιμα σημεία. Το visual studio βοηθάει με τον CPU profiler σαν πρώτη εικόνα. Επίσης το ίδιο IDE δίνει την δυνατότητα να βλέπουμε την Process Memory με τα private bytes. Έτσι μπορούμε δοκιμάζοντας τον κώδικα να μειώσουμε την κατανάλωση μνήμης που χρειάζεται.

Μετρήσεις με το Visual Studio IDE

Σύμφωνα με το Visual Studio το μη-βελτιστοποιημένο πρόγραμμα χρρειάζεται 26 MB και χρειάζεται 2.4 s για να εκτελεστεί.Το βελτιστοποιημένο πρόγραμμα χρειάζεται 24 MB και εκτελείται σε 2.5553 s όταν έχουμε σβήσει όλα τα σχόλια , έχουμε κρατήσει μόνο τις συναρτήσεις που είναι απολύτως απαραίτητες για την παραγωγή του σωστού αποτελέσματος και εφόσον τις ενσωματώσαμε όλες στην main.

Αναπτύσσοντας την συνάρτηση padder() σε 3 διαφορετικούς βρόχους έριξε την επεξεργαστική ισχύ από 25% που ήταν προηγουμένως και για τον μη-βελτιστοποιημένο και για τον βελτιστοποιημένο κώδικα σε 22% κατά την εκτέλεση της συγκεκριμένης συνάρτησης.

Έπειτα εφαρμόζουμε loop unrolling στην συνάρτηση που κάνει κανονικοποίηση το κόκκινο frame αφού του φορτώσουμε δεδομένα από την εικόνα YUV.

Η υψηλή κατανάλωση της μνήμης επιχειρήσαμε να την μειώσουμε δεσμεύοντας όλο και λιγότερη μνήμη στις global μεταβλητές.

Από 26MB σε 24MB βάζοντας όλες τις απαραίτητες συναρτήσεις μέσα στην main.

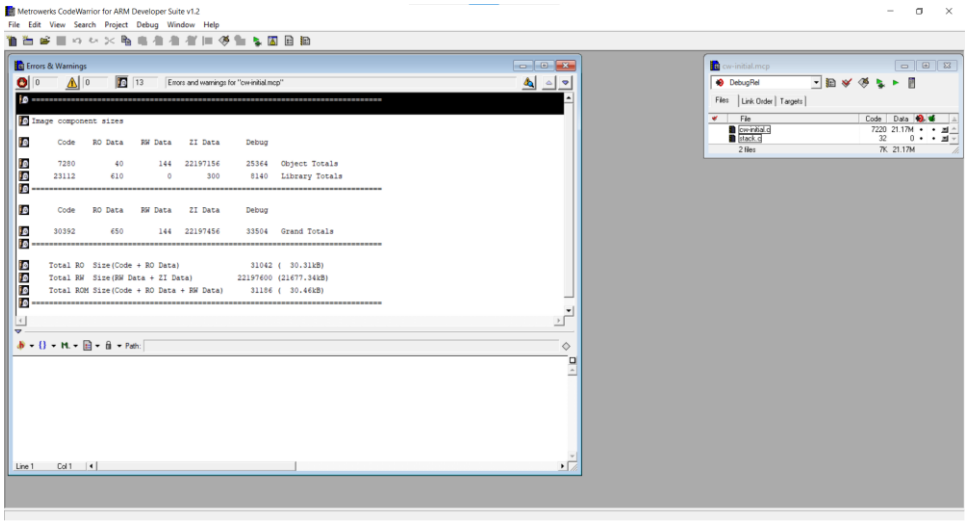
Από 24MB σε 21MB μειώνοντας κατά 3 τους πίνακες (σβήσαμε τους frame_grayscale_r , g , b) και κρατήσαμε μόνο τους πίνακες που αρχηκά λεγόντουσαν frame_y , frame_u , frame_v και frame_r , frame_g , frame_b και τους μετονομάσαμε σε frame_1_a , frame_1_b frame_1c και frame_2_a frame_2_b frame_2_c

Από 21 MB σε 17MB σβήνοντας τους πίνακες coloured_image που χρησιμοποιούνται στο βήαμ 7 και τους αντικαταστήσαμε με τους παραπάνω frame_1_{a,b,c} και frame_2_{a,b,c}.

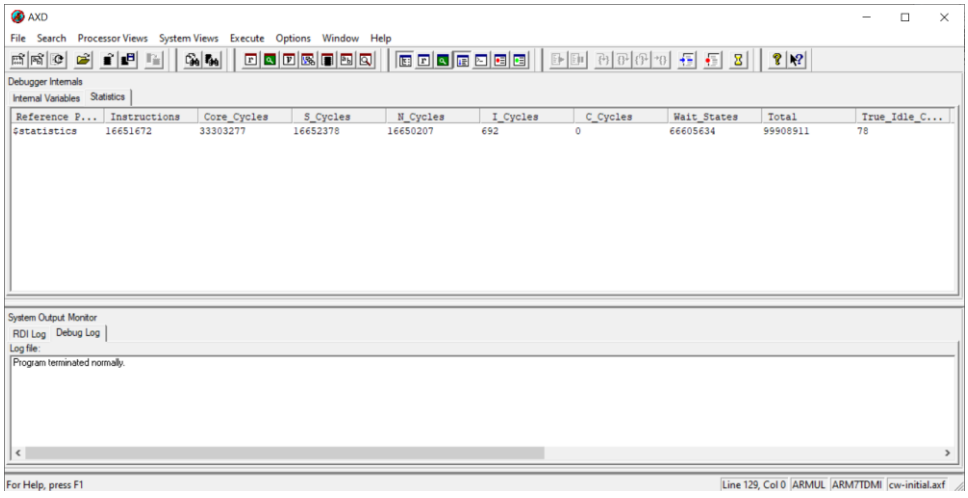
Μετρήσεις με το CodeWarrior IDE

Χρειάστηκε να γίνουν κάποιες αλλαγές στον κώδικα , προκειμένου να μπορέσει να εκτελεστεί σωστά από το CodeWarrior IDE. Αυτές είναι η περιοχή pragma arm section και οι διορθώσεις των pow(), arctan() διότι είχαν unhandled overflow. Επίσης χρειάστηκαν τα αρχεία scatter.txt, memory.map και stack. Το πρώτο δείχνει τον χώρο αποθήκευσης (δομή μνήμης στο λογισμικό) , το δεύτερο την δομή της μνήμης στο υλικό και το τρίτο δηλώνει την δομή δεδομένων την οποία έχει η μνήμη.

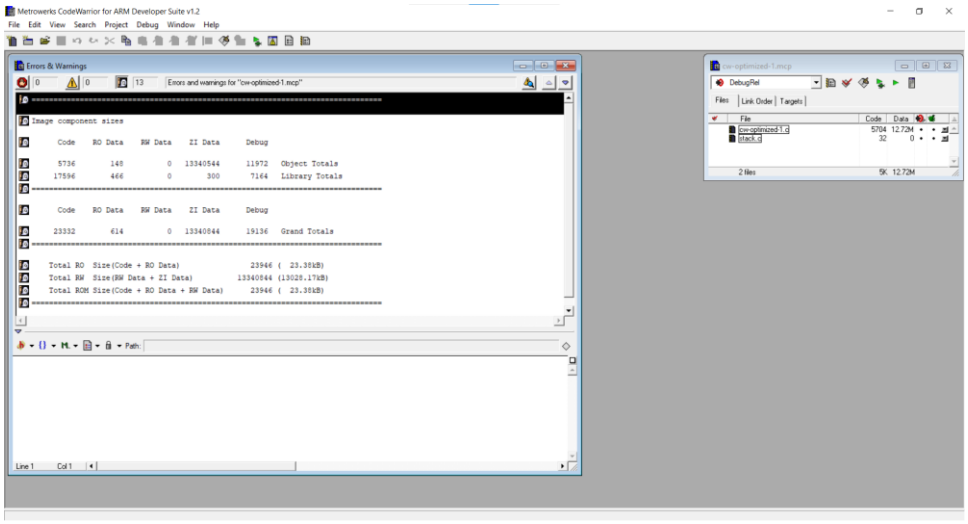
Τα στοιχεία του αρχικού προγράμματος φαίνονται παρακάτω:



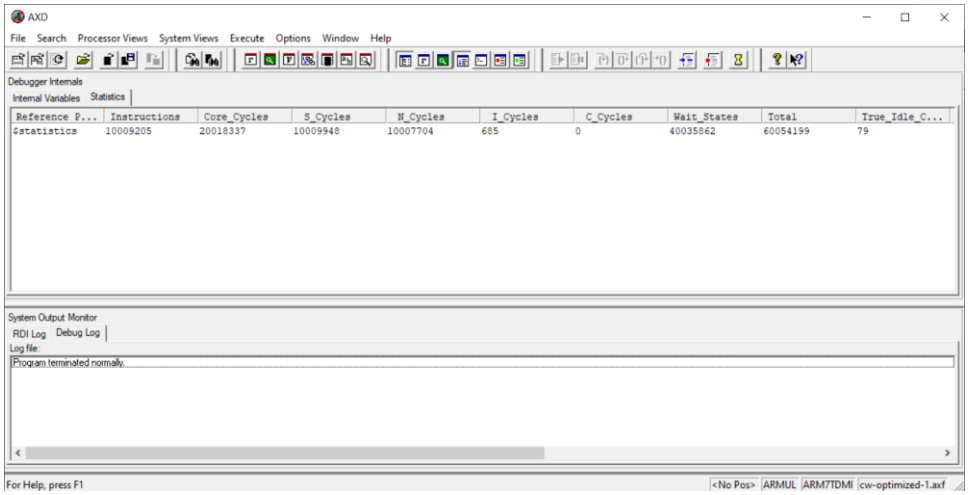
Και:



Τα στοιχεία του βελτιστοποιημένου προγράμματος είναι:



Και:



Σε μορφή πινάκων:

initial					
Code	RO Data	RW Data	ZI Data	Debug	
7280	40	144	22197156	25364	Object Totals
23112	610	0	300	8140	Library Totals
30392	650	144	22197456	33504	Grand Totals
Total RO Size(Code + RO Data)				31042	
Total RW Size(RW Data + ZI Data)				22197600	
Total ROM Size(Code + RO Data + RW Data)				31186	

optimized					
Code	RO Data	RW Data	ZI Data	Debug	
5736	148	0	13340544	11972	Object Totals
17596	466	0	300	7164	Library Totals
23332	614	0	13340844	19136	Grand Totals
Total RO Size(Code + RO Data)				23946	
Total RW Size(RW Data + ZI Data)				13340844	
Total ROM Size(Code + RO Data + RW Data)				23946	

Και επίσης έχουμε :

	Instuctions	Core Cycles	S Cycles	N Cycles	I Cycles	C Cycles	Wait Cycles	Total	True Idle Cycles
initial	16651672	33303277	16652378	16650207	692	0	66605634	99908911	78
optimized	10009205	20018337	10009948	10007704	685	0	40035862	60054199	79

Αναφορές

1. Engineering A Compiler by Keith D. Cooper and Linda Torczon, 3rd Edition, Elsevier publications, 2023
2. Digital Image Processing by R. Gonzalez and R. Woods, 4th Edition, Pearson, 2018
3. Digital Video and HD - Algorithms and Interfaces by Charles Poynton, 2nd Edition, Elsevier publications, 2012
4. Compilers: Principles, Techniques and Tools by Alfred V. Aho, Monica S. Lam, Ravi Sethi and Jeffrey D. Ullman, 2nd Edition, Addison-Wesley, 2006
5. Best Practice Guide - Deep Learning - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Schematic-illustration-of-a-convolutional-operation-The-convolutional-kernel-shifts-over_fig2_332190148 [accessed 26 Nov, 2023]