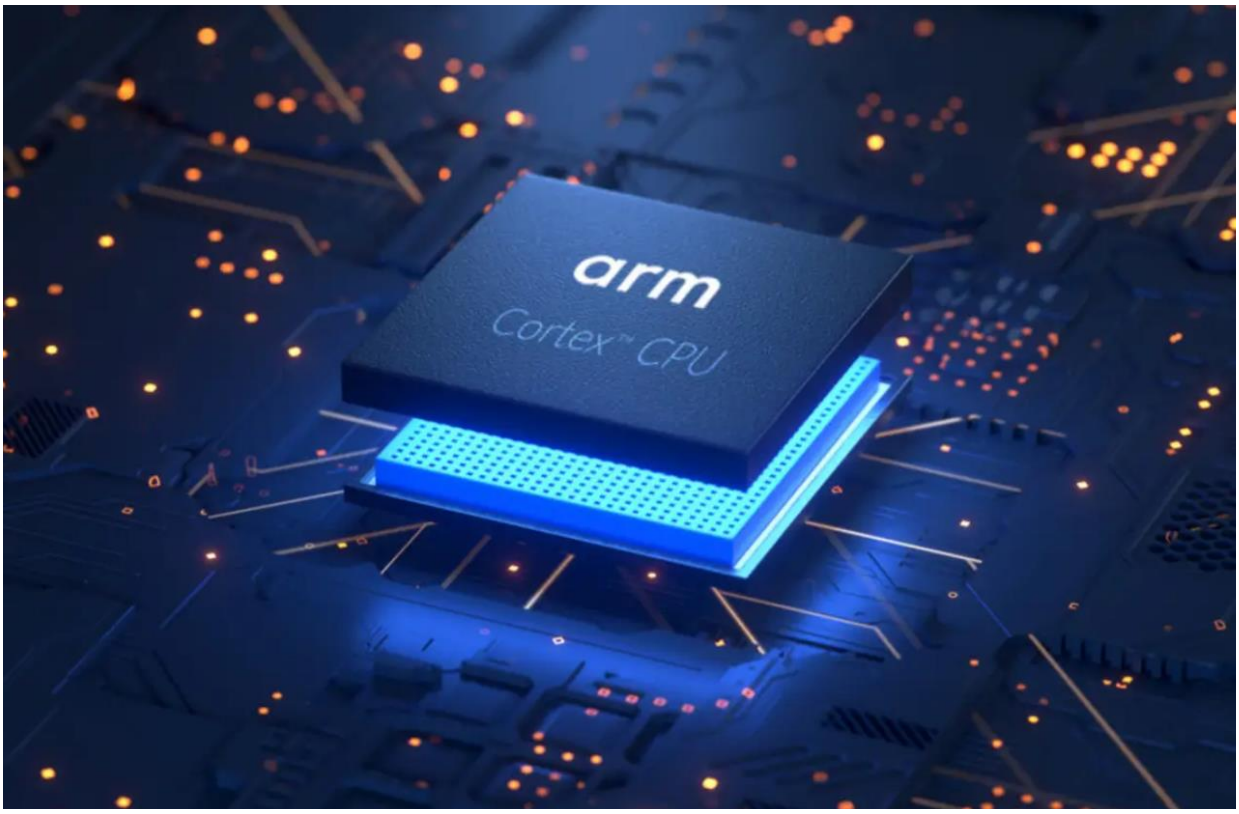
1ο Μέρος Εργασίας Του Μαθήματος "Σχεδιασμός Ενσωματωμένων Συστημάτων"

2023-2024 – ΟΜΑΔΑ 11

Ονοματεπώνυμο:

Μωραΐτη Παναγιώτα, AM: 58054

Σούλη Ευθυμία, AM: 58121



Θέμα: **Εφαρμογή του συσχετισμένου μετασχηματισμού (affine transformation) στην επεξεργασία εικόνας**

Περιεχόμενα

Περιγραφή Αλγορίθμου

Οι συσχετισμένοι μετασχηματισμοί είναι στην πραγματικότητα γραμμικοί μετασχηματισμοί που υλοποιούνται με τον πολλαπλασιασμό ενός σημείου της εικόνας που αναπαρίσταται ως διάνυσμα, με τον πίνακα μετασχηματισμού Affine Matrix. Είναι μετασχηματισμοί, οι οποίοι διατηρούν τις γραμμές και τον παραλληλισμό, αλλά όχι απαραίτητα τις Ευκλείδειες αποστάσεις και γωνίες.

Αν το αρχικό σημείο είναι το (x,y), τότε το σημείο (x’,y’) θα περιγράφεται ως εξής:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} ax + by \\ dx + ey \end{bmatrix} = \begin{bmatrix} a & b \\ d & e \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

όπου ο πίνακας $\begin{bmatrix} a & b \\ d & e \end{bmatrix}$ είναι ο πίνακας μετασχηματισμού.

Οι affine μετασχηματισμοί περιλαμβάνουν μετασχηματισμούς αλλαγής κλίμακας, περιστροφής, μετατόπισης κ.ά., αλλά και τους συνδυασμούς τους. Για να συνδυαστούν οι διάφοροι απλοί μετασχηματισμοί σε πιο σύνθετους, πρέπει να απεικονίσουμε τους μετασχηματισμούς σε μορφές πινάκων. Για να γίνει αυτό και για τον πίνακα μετατόπισης, ο οποίος δεν αποτελεί γραμμικό μετασχηματισμό, χρησιμοποιούμε ομογενείς μετασχηματισμούς, όπου ισχύουν τα εξής:

$$\begin{bmatrix} x \\ y \end{bmatrix} \Longrightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \begin{bmatrix} a & b & 0 \\ d & e & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax + by \\ dx + ey \\ 1 \end{bmatrix}$$

, και οι πίνακες των κύριων μετασχηματισμών:

Scale: $\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$, Rotate: $\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$ και Translate: $\begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix}$

Στην εργασία, πραγματοποιήθηκε ο affine μετασχηματισμός με συνδυασμό scaling (1:2) , rotation (θ = 45˚) και translation(250,-400) (μετατόπιση), με τη παραπάνω σειρά. Άρα, θα έχουμε:

$$AffineMatrix = Translate * Rotate * Scaling = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix} =$$

$$\begin{bmatrix} 1 & 0 & 250 \\ 0 & 1 & -400 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos 45 & -\sin 45 & 0 \\ -\sin 45 & \cos 45 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \sqrt{2} & -\sqrt{2} & 250 \\ \sqrt{2} & \sqrt{2} & -400 \\ 0 & 0 & 1 \end{bmatrix}$$

Στη συνέχεια, μπορούμε να δούμε τον τελικό πίνακα συσχετισμένου μετασχηματισμού.

scaling, shearing, rotating

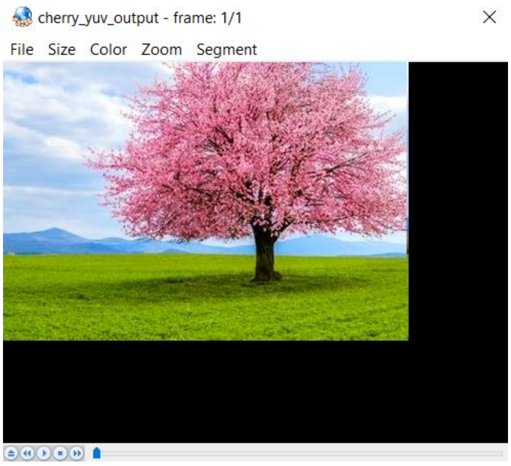
Translation

$$\begin{bmatrix} q_1 \\ q_2 \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ 1 \end{bmatrix}$$

Locking the image on z=1 plane

Παρακάτω, φαίνεται η αρχική εικόνα, χωρίς μετασχηματισμούς και έπειτα ακολουθούν, παραδείγματα εφαρμογής διαφόρων affine matrices. Τέλος, έχουμε τον πίνακα που χρησιμοποιήθηκε στην εργασία (είναι ο παραπάνω):

<u>Αρχική εικόνα</u>:



Μετατόπιση:

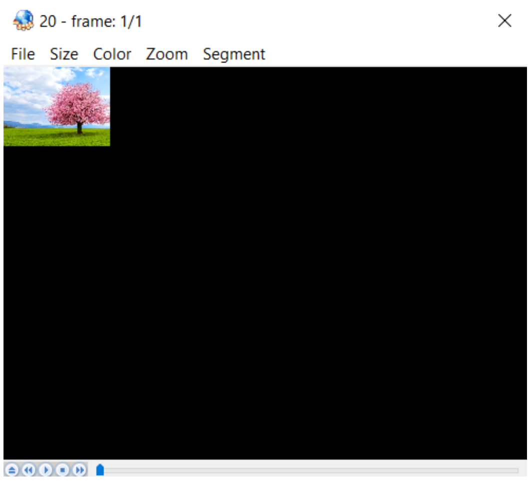
```
float angle_in_degrees = 0.0;
float angle_in_radians = (3.1415 / 180.0) * angle_in_degrees;


float affineMatrix[3][3] = {
    static_cast<float>(cos(angle_in_radians)), -static_cast<float>(sin(angle_in_radians)), 100,
    static_cast<float>(sin(angle_in_radians)), static_cast<float>(cos(angle_in_radians)), 100,
    0, 0, 1
};
```

Περιστροφή:

```
float angle_in_degrees = 5.0;
float angle_in_radians = (3.1415 / 180.0) * angle_in_degrees;


float affineMatrix[3][3] = {
    static_cast<float>(cos(angle_in_radians)), -static_cast<float>(sin(angle_in_radians)), 0,
    static_cast<float>(sin(angle_in_radians)), static_cast<float>(cos(angle_in_radians)), 0,
    0, 0, 1
};
```



Υποδιαίρεση κλίμακας(1:5):

```
float affineMatrix[3][3] = {
    {5, 0, 0},
    {0, 5, 0},
    {0, 0, 1}
};
```

<u>Ο μετασχηματισμός με βάση τον οποίο έγιναν οι συγκρίσεις</u>:

```
float affineMatrix[3][3] = {
{1.414, -1.414, 250},
{1.414, 1.414, -400},
{0, 0, 1}
};
```

<u>Αρχικός κώδικας</u>

Στον αρχικό κώδικα εφαρμόζουμε στα τρία κανάλια current_y, current_u, current_v τη συνάρτηση applyAffineTransform, η οποία δέχεται ως είσοδο το κάθε κανάλι για το οποίο καλείται, τον πίνακα affineMatrix και ένα αριθμό ενδεικτικό για το ποιο κανάλι χρησιμοποιείται. Επειδή η μετασχηματισμένη εικόνα γράφεται κατ’ ευθείαν στην αρχική (originalImage[N][M]), πρέπει να μηδενιστεί αρχικά. Για τα κανάλια u, v μηδενισμός σημαίνει κάθε Pixel να έχει τιμή 128 (μαύρο χρώμα), ενώ για το y θέτουμε το 0. Τα περιεχόμενα της αρχικής εικόνας αποθηκεύονται προσωρινά σε ένα buffer πίνακα, πάνω στον οποίο γίνονται οι μετασχηματισμοί και αποθηκεύονται πάλι πίσω στην originalImage.

```
void applyAffineTransform(int originalImage[N][M], float affineMatrix[3][3],int channel)
{
    // Copy the original values to the buffer
    for (i = 0; i < N; i++)
    {
        for (j = 0; j < M; j++)
        {
            buffer[i][j] = originalImage[i][j];
            if (channel == 2 || channel == 3)
                originalImage[i][j] = 128;
            else
                originalImage[i][j] = 0;
        }
    }

    for (y = 0; y < N; y++) {
        for (x = 0; x < M; x++) {
            // Apply affine transformation
            int newX = (int)(affineMatrix[0][0] * x + affineMatrix[0][1] * y + affineMatrix[0][2]);
            int newY = (int)(affineMatrix[1][0] * x + affineMatrix[1][1] * y + affineMatrix[1][2]);

            if (newX >= 0 && newX < M - 1 && newY >= 0 && newY < N - 1)
            {
              originalImage[y][x] = buffer[newY][newX];
            }
        }
    }
}
```

Βελτιστοποιήσεις και αποτελέσματα

α) Αρχικά να εφαρμοστούν οι μετασχηματισμοί βρόχων για την κανονικοποίηση της δομής του αλγορίθμου αλλά και να εφαρμοστεί οποιοσδήποτε άλλος μετασχηματισμός για τη βελτίωση της απόδοσης του αλγορίθμου

Αρχικός αλγόριθμος με buffer και originalImage

Internal Variables | Statistics

Reference P...	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	219564722	300925972	235330000	51993394	28307687	0	315631081

Instructions: Δείχνουν το συνολικό αριθμό εντολών που απαιτούνται για να ολοκληρωθεί ο κώδικας.

Core Cycles: Κύκλοι του επεξεργαστή για την εκτέλεση του προγράμματος.

S (Sequential) Cycles: Κύκλοι του επεξεργαστή, στους οποίους γίνονται διαδοχικές προσπελάσεις στη μνήμη.

N (Non-Sequential) Cycles: Κύκλοι του επεξεργαστή, στους οποίους γίνονται μη διαδοχικές προσπελάσεις στη μνήμη.

I (Internal Cycles): Κύκλοι που δεν γίνεται κλήση στη μνήμη.

C (Co-processor) Cycles: Κύκλοι που αφορούν τη χρήση άλλης βοηθητικής μονάδας επεξεργασίας.

Total Cycles: Συνολικοί κύκλοι.

Γενικά στόχος είναι η μείωση των core cycles.

Μηδενισμός και ανάθεση σε πίνακα transformed εκτός της συνάρτησης

Για να αποφευχθεί η δομή ελέγχου μέσα στην συνάρτηση για το αν έχουμε κανάλι y,u ή v, φτιάχνουμε ένα πίνακα transformed, ο οποίος μηδενίζεται στην main και στη συνάρτηση applyAffineTransform γίνεται σε αυτόν η εγγραφή των στοιχείων της νέας εικόνας. Το παραπάνω γίνεται για κάθε κανάλι. Τέλος γίνεται ανάθεση των δεδομένων του transformed σε κάθε ένα από τα κανάλια. Πιο συγκεκριμένα, η συνάρτηση γίνεται τώρα:

```
void applyAffineTransform(int originalImage[N][M], int transformedImage[N][M], float affineMatrix[3][3])
{
    for (y = 0; y < N; y++)
    {
        for (x = 0; x < M; x++)
        {
        // Apply affine transformation
        int newX = (int)(affineMatrix[0][0] * x + affineMatrix[0][1] * y + affineMatrix[0][2]);
        int newY = (int)(affineMatrix[1][0] * x + affineMatrix[1][1] * y + affineMatrix[1][2]);

        if (newX >= 0 && newX < M - 1 && newY >= 0 && newY < N - 1)
        {
            transformedImage[y][x] = originalImage[newY][newX];
        }
    }
  }
}
```

Στη main έχουμε τρεις φορές το παρακάτω για κάθε κανάλι:

```
int main()
{
    read();

    for (i = 0; i < N; i++) {
        for (j = 0; j < M; j++) {
            transformed[i][j] = 0;
        }
    }

    applyAffineTransform(current_y, transformed, affineMatrix);
    for (i = 0; i < N; i++)
    {
        for (j = 0; j < M; j++)
        {
            current_y[i][j] = transformed[i][j];
        }
    }
```

Μόνο που για τα current_u,current_v αντί για μηδέν τα στοιχεία του transformed αρχικοποιούνται στο 128. Και όπως βλέπουμε παρακάτω μειώνονται και τα Core_cycles και τα Instructions.

Internal Variables | Statistics

Reference P...	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	217363609	297806782	232388625	52368000	27755266	0	312511891

Loop Unrolling (για 3 επαναλήψεις)

Γενικά με το loop unrolling, έχουμε αύξηση της ταχύτητας εκτέλεσης λόγω μείωσης των εντολών ελέγχου που συμβάλλουν στο flow control του for loop, άρα έχουμε μείωση των instructions. Επίσης, με το loop unrolling δίνεται η ευκαιρία στον επεξεργαστή να εκτελέσει παράλληλα εντολές, μειώνοντας έτσι τα core cycles. Βέβαια, από ένα σημείο και μετά σταματά να βελτιώνεται η ταχύτητα.

Η συνάρτηση τώρα γίνεται:

```
void applyAffineTransform(int originalImage[N][M], int transformedImage[N][M], float affineMatrix[3][3]) {
    for (y = 0; y < N; y++)
    {
        for (x = 0; x < M;  x += 3)
        {
            // Apply affine transformation
            int newX1 = (int)(affineMatrix[0][0] * x + affineMatrix[0][1] * y + affineMatrix[0][2]);
            int newY1 = (int)(affineMatrix[1][0] * x + affineMatrix[1][1] * y + affineMatrix[1][2]);

            int newX2 = (int)(affineMatrix[0][0] * (x+1) + affineMatrix[0][1] * y + affineMatrix[0][2]);
            int newY2 = (int)(affineMatrix[1][0] * (x+1) + affineMatrix[1][1] * y + affineMatrix[1][2]);

            int newX3 = (int)(affineMatrix[0][0] * (x+2) + affineMatrix[0][1] * y + affineMatrix[0][2]);
            int newY3 = (int)(affineMatrix[1][0] * (x+2) + affineMatrix[1][1] * y + affineMatrix[1][2]);

            if (newX1 >= 0 && newX1 < M - 1 && newY1 >= 0 && newY1 < N - 1)
            {
                transformedImage[y][x] = originalImage[newY1][newX1];
            }

            if (newX2 >= 0 && newX2 < M - 1 && newY2 >= 0 && newY2 < N - 1)
            {
                transformedImage[y][x+1] = originalImage[newY2][newX2];
            }

            if (newX3 >= 0 && newX3 < M - 1 && newY3 >= 0 && newY3 < N - 1)
            {
                transformedImage[y][x+2] = originalImage[newY3][newX3];
            }
        }
    }
}
```

Αυτό που αλλάζει είναι ότι τώρα το x αυξάνεται με βήμα 3, οπότε επαναλαμβάνουμε 3 φορές manually τη διαδικασία υπολογισμού των καινούργιων x, y και εγγραφής των στοιχείων του originalImage στον transformed.

Όπως, ήταν αναμενόμενο, έχουμε βελτίωση και στα instructions και στα core_cycles

Internal Variables | Statistics

Reference P...	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	188299051	262887565	199497039	51389211	23070481	0	273956731

Loop Unrolling (για 6 επαναλήψεις), βελτίωση και στα instructions και στα core_cycles

Internal Variables | Statistics

Reference P...	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	179884207	251852491	190525929	49940154	21531784	0	261997867

Loop Fusion (μαζί με Loop Unrolling για 6 επαναλήψεις), 3 πίνακες transformed

Με την τεχνική fusion, δηλαδή ενσωμάτωσης των πολλαπλών βρόχων σε ένα (συνένωση βρόχων) μειώνεται ο αριθμός των επαναλήψεων και των εντολών ελέγχου, οπότε έχουμε λιγότερες διακλαδώσεις, άρα πιο γρήγορη εκτέλεση.

Στον κώδικα δημιουργήθηκαν 3 πίνακες transformed διαφορετικοί για κάθε κανάλι και αρικοποιήθηκαν όλοι μαζί. Το ίδιο έγινε και για την ανάθεση των τιμών των transformed πινάκων στους current.

```
int main()
{
    read();

    for (i = 0; i < N; i++) {
        for (j = 0; j < M; j++) {
            transformed_y[i][j] = 0;
            transformed_u[i][j] = 128;
            transformed_v[i][j] = 128;
        }
    }

    applyAffineTransform(current_y, transformed_y, affineMatrix);
    applyAffineTransform(current_u, transformed_u, affineMatrix);
    applyAffineTransform(current_v, transformed_v, affineMatrix);
    for (i = 0; i < N; i++)
    {
        for (j = 0; j < M; j++)
        {
            current_y[i][j] = transformed_y[i][j];
            current_u[i][j] = transformed_u[i][j];
            current_v[i][j] = transformed_v[i][j];
        }
    }

    write();
    printf("Image Written");

    return 0;
}
```

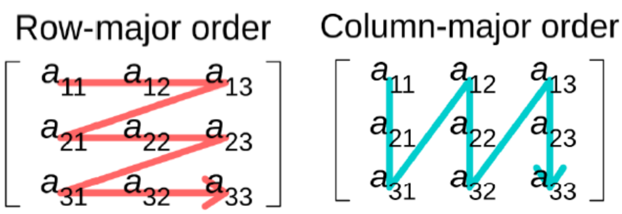
Και πάλι μειώνονται και τα Instructions και τα core cycles

Internal Variables | Statistics

Reference P...	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	177476234	246858386	188120944	48089080	20793738	0	257003762

Loop Interchange (μαζί με Loop Unrolling για 6 επαναλήψεις και Loop Fusion), δεν βοήθησε

Το Loop Interchange είναι μια τεχνική βελτιστοποίησης, στην οποία μπορούμε να αλλάξουμε τη σειρά των βρόχων όταν προσπελαύνουμε δισδιάστατους πίνακες. Η προσπέλαση μπορεί να γίνει είτε κατά γραμμές (row major order), είτε κατά στήλες (column major order). Η C αποθηκεύει τους πίνακες, κατά γραμμές, οπότε αν την κάνουμε προσπέλαση με τον ίδιο τρόπο με τον οποίο αποθηκεύονται τα στοιχεία, τότε θα έχουμε περισσότερα cache hit, οπότε ο κώδικάς μας θα τρέχει πιο γρήγορα. Στον κώδικα μας, οι προσπελάσεις γίνονται κατά γραμμές, οπότε όταν αλλάξαμε τους βρόχους ώστε οι προσπελάσεις να γίνονται κατά στήλες, όπως ήταν αναμενόμενο ο κώδικας έτρεξε πιο αργά.



Οι αλλαγές έγιναν μόνο στη main και εφόσον διαπιστώσαμε αυτό που περιμέναμε δεν τις εφαρμόσαμε στη συνάρτηση.

```
int main()
{
    read();

    for (i = 0; i < M; i++) {
        for (j = 0; j < N; j++) {
            transformed_y[j][i] = 0;
            transformed_u[j][i] = 128;
            transformed_v[j][i] = 128;
        }
    }

    applyAffineTransform(current_y, transformed_y, affineMatrix);
    applyAffineTransform(current_u, transformed_u, affineMatrix);
    applyAffineTransform(current_v, transformed_v, affineMatrix);
    for (i = 0; i < M; i++)
    {
        for (j = 0; j < N; j++)
        {
            current_y[j][i] = transformed_y[j][i];
            current_u[j][i] = transformed_u[j][i];
            current_v[j][i] = transformed_v[j][i];
        }
    }

    write();
    printf("Image Written");

    return 0;

}
```

Τα Instructions και τα core cycles δεν μειώνονται, αντιθέτως αυξάνονται. Οπότε, δεν υπάρχει κάποια βελτίωση

Internal Variables | Statistics

Reference P...	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	179502399	248883557	190146609	48089329	20792995	0	259028933

Loop tiling για 2 στήλες και 6 γραμμές (Loop Unrolling για 6 επαναλήψεις, Loop Fusion)

Η τεχνική Loop Tiling, χωρίζει έναν εμφωλευμένο βρόχο σε blocks (tiles). Δηλαδή σε κάθε επανάληψη αντί να γίνεται προσπέλαση ενός στοιχείου, γίνεται προσπέλαση ενός ολόκληρου block του πίνακα. Με αυτόν τον τρόπο ο βρόγχος διαιρείται σε μικρότερα τμήματα, τα οποία μπορούν να εκτελεστούν παράλληλα. Επίσης, με τη χρήση tiles πιθανώς εκμεταλλευόμαστε την κρυφή μνήμη.

Τα tiles που χρησιμοποιήσαμε εμείς αποτελούνται από 2 γραμμές και 6 στήλες (12 στοιχεία). Επεκτείναμε στην ουσία το loop unrolling και στις στήλες.

Παρακάτω φαίνεται ότι ο κώδικας του loop unrolling επαναλαμβάνεται μια ακόμα φορά για τη 2^{η} γραμμή του tile.

```
void applyAffineTransform(int originalImage[N][M], int transformedImage[N][M], float affineMatrix[3][3]) {
    for (y = 0; y < N; y += 2)
    {
        for (x = 0; x < M;  x += 6)
        {
            // Apply affine transformation
            int newX1 = (int)(affineMatrix[0][0] * x + affineMatrix[0][1] * y + affineMatrix[0][2]);
            int newY1 = (int)(affineMatrix[1][0] * x + affineMatrix[1][1] * y + affineMatrix[1][2]);

            int newX2 = (int)(affineMatrix[0][0] * (x+1) + affineMatrix[0][1] * y + affineMatrix[0][2]);
            int newY2 = (int)(affineMatrix[1][0] * (x+1) + affineMatrix[1][1] * y + affineMatrix[1][2]);

            int newX3 = (int)(affineMatrix[0][0] * (x+2) + affineMatrix[0][1] * y + affineMatrix[0][2]);
            int newY3 = (int)(affineMatrix[1][0] * (x+2) + affineMatrix[1][1] * y + affineMatrix[1][2]);

            int newX4 = (int)(affineMatrix[0][0] * (x+3) + affineMatrix[0][1] * y + affineMatrix[0][2]);
            int newY4 = (int)(affineMatrix[1][0] * (x+3) + affineMatrix[1][1] * y + affineMatrix[1][2]);

            int newX5 = (int)(affineMatrix[0][0] * (x+4) + affineMatrix[0][1] * y + affineMatrix[0][2]);
            int newY5 = (int)(affineMatrix[1][0] * (x+4) + affineMatrix[1][1] * y + affineMatrix[1][2]);

            int newX6 = (int)(affineMatrix[0][0] * (x+5) + affineMatrix[0][1] * y + affineMatrix[0][2]);
            int newY6 = (int)(affineMatrix[1][0] * (x+5) + affineMatrix[1][1] * y + affineMatrix[1][2]);

            if (newX1 >= 0 && newX1 < M - 1 && newY1 >= 0 && newY1 < N - 1)
            {
                transformedImage[y][x] = originalImage[newY1][newX1];
            }

            if (newX2 >= 0 && newX2 < M - 1 && newY2 >= 0 && newY2 < N - 1 && x+1 < M)
            {
                transformedImage[y][x+1] = originalImage[newY2][newX2];
            }

            if (newX3 >= 0 && newX3 < M - 1 && newY3 >= 0 && newY3 < N - 1 && x+2 < M)
            {
                transformedImage[y][x+2] = originalImage[newY3][newX3];
            }
```

```
// Apply affine transformation
newX1 = (int)(affineMatrix[0][0] * x + affineMatrix[0][1] * (y+1) + affineMatrix[0][2]);
newY1 = (int)(affineMatrix[1][0] * x + affineMatrix[1][1] * (y+1) + affineMatrix[1][2]);

newX2 = (int)(affineMatrix[0][0] * (x+1) + affineMatrix[0][1] * (y+1) + affineMatrix[0][2]);
newY2 = (int)(affineMatrix[1][0] * (x+1) + affineMatrix[1][1] * (y+1) + affineMatrix[1][2]);

newX3 = (int)(affineMatrix[0][0] * (x+2) + affineMatrix[0][1] * (y+1) + affineMatrix[0][2]);
newY3 = (int)(affineMatrix[1][0] * (x+2) + affineMatrix[1][1] * (y+1) + affineMatrix[1][2]);

newX4 = (int)(affineMatrix[0][0] * (x+3) + affineMatrix[0][1] * (y+1) + affineMatrix[0][2]);
newY4 = (int)(affineMatrix[1][0] * (x+3) + affineMatrix[1][1] * (y+1) + affineMatrix[1][2]);

newX5 = (int)(affineMatrix[0][0] * (x+4) + affineMatrix[0][1] * (y+1) + affineMatrix[0][2]);
newY5 = (int)(affineMatrix[1][0] * (x+4) + affineMatrix[1][1] * (y+1) + affineMatrix[1][2]);

newX6 = (int)(affineMatrix[0][0] * (x+5) + affineMatrix[0][1] * (y+1) + affineMatrix[0][2]);
newY6 = (int)(affineMatrix[1][0] * (x+5) + affineMatrix[1][1] * (y+1) + affineMatrix[1][2]);
// Interpolate colors
if (newX1 >= 0 && newX1 < M - 1 && newY1 >= 0 && newY1 < N - 1 && y+1 < N)
{
    transformedImage[y+1][x] = originalImage[newY1][newX1];
}

if (newX2 >= 0 && newX2 < M - 1 && newY2 >= 0 && newY2 < N - 1 && x+1 < M && y+1 < N)
{
    transformedImage[y+1][x+1] = originalImage[newY2][newX2];
}

if (newX3 >= 0 && newX3 < M - 1 && newY3 >= 0 && newY3 < N - 1 && x+2 < M && y+1 < N)
{
    transformedImage[y+1][x+2] = originalImage[newY3][newX3];
}
```

Αν και υπάρχει αύξηση στα total cycles πιθανώς αν αυξήσουμε τις γραμμές του tile να δούμε κάποια βελτίωση. Βέβαια, τα instructions μειώνονται ελάχιστα, εκτέλεσης λόγω μείωσης των εντολών ελέγχου που συμβάλλουν στο flow control του for loop.

Internal Variables | Statistics

Reference P...	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	177472001	248565188	187562050	49707109	21441405	0	258710564

Loop Collapsing

Στην τεχνική loop collapsing, ένας δισδιάστατος πίνακας αποθηκεύεται στη μνήμη σαν ένα διάνυσμα, δηλαδή τα στοιχεία του είναι αποθηκευμένα σειριακά. Στη C η οποία είναι row-major γλώσσα, αποθηκεύεται πρώτα η πρώτη γραμμή, μετά η δεύτερη κτλ. Οπότε, μπορούμε να χρησιμοποιήσουμε ένα δείκτη (pointer) για να διατρέξουμε τα στοιχεία του πίνακα αντί για μια εμφωλευμένη for, η προσπέλαση των στοιχείων είναι συνεχόμενη. Η ενοποίηση των βρόγχων μπορεί να διευκολύνει την παραλληλοποίηση.

Η τεχνική αυτή μπορεί να βοηθήσει στην αύξηση της ταχύτητας του προγράμματος αν συνδυαστεί κατάλληλα με άλλες τεχνικές βελτιστοποίησης.

Loop Collapsing μέσα στη συνάρτηση, δεν βοήθησε τα total cycles είναι περισσότερα ακόμα και από το αρχικό πρόγραμμα.

```
void applyAffineTransform(int originalImage[N][M], int transformedImage[N][M], float affineMatrix[3][3])
{
    int *o = &originalImage[0][0];
    int *t = &transformedImage[0][0];

    for (i = 0; i < N * M; i++)
    {
        // Calculate x and y indices from the flattened index
        int x = i % M;
        int y = i / M;

        // Apply affine transformation
        int newX = (int)(affineMatrix[0][0] * x + affineMatrix[0][1] * y + affineMatrix[0][2]);
        int newY = (int)(affineMatrix[1][0] * x + affineMatrix[1][1] * y + affineMatrix[1][2]);

        if (newX >= 0 && newX < M - 1 && newY >= 0 && newY < N - 1)
        {
            int interpolatedValue = *(o + newY * M + newX);
            *(t + y * M + x) = interpolatedValue;
        }
    }
}
```

Internal Variables | Statistics

Reference P...	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	242955346	325953400	259815795	53642100	27200614	0	340658509

Loop Collapsing μέσα στη συνάρτηση και στη main μαζί με loop fusion, δεν βοήθησε τα total cycles και εδώ είναι περισσότερα ακόμα και από το αρχικό πρόγραμμα.

```
int *currentYPtr = &current_y[0][0];
int *transformedYPtr = &transformed_y[0][0];
int *currentUPtr = &current_u[0][0];
int *transformedUPtr = &transformed_u[0][0];
int *currentVPtr = &current_v[0][0];
int *transformedVPtr = &transformed_v[0][0];

int main()
{
    read();

    for (i = 0; i < N; i++) {
       for (j = 0; j < M; j++) {
           transformed_y[i][j] = 0;
           transformed_u[i][j] = 128;
           transformed_v[i][j] = 128;
       }
    }


    applyAffineTransform(current_y,  transformed_y, affineMatrix);
    applyAffineTransform(current_u, transformed_u, affineMatrix);
    applyAffineTransform(current_v,  transformed_v, affineMatrix);
    for (i = 0; i < N * M; i++)
    {
        *(currentYPtr + i) = *(transformedYPtr + i);
        *(currentUPtr + i) = *(transformedUPtr + i);
        *(currentVPtr + i) = *(transformedVPtr + i);

    }


    write();
    printf("Image Written");
```

Ο πιθανός λόγος που δε βλέπουμε κάποια βελτίωση είναι ότι με τη χρήση δεικτών χρειάζονται επιπλέον υπολογισμοί για την εύρεση της θέσης των δεδομένων στη μνήμη, οπότε το overhead που προστίθεται μας επιβραδύνει το πρόγραμμα αντί αν το επιταχύνει.

Internal Variables | Statistics

Reference Po...	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	242201659	325931071	259065832	53448659	28121689	0	340636180

Loop tiling για 4 στήλες και 6 γραμμές (Loop Unrolling για 6 επαναλήψεις, Loop Fusion)

Debugger Internals

Internal Variables | Statistics

Reference P...	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	170291237	240763781	179924512	49222273	21762372	0	250909157

Loop tiling για 4 στήλες και 7 γραμμές (Loop Unrolling για 7 επαναλήψεις, Loop Fusion)

Debugger Internals

Internal Variables | Statistics

Reference P...	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	168758408	238881803	178390519	48907501	21577119	0	248875139

Loop tiling για 4 στήλες και 8 γραμμές (Loop Unrolling για 8 επαναλήψεις, Loop Fusion), το βέλτιστο (λιγότερα core cycles)

Debugger Internals

Internal Variables | Statistics

Reference P...	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	167581976	237388349	177181816	48642691	21438531	0	247263038

Τα instructions και τα core cycles μειώνονται (και τα total cycles), απλά όσο αυξάνουμε το μέγεθος του tile ο ρυθμός μείωσης μειώνεται. Κάποια στιγμή θα δούμε ότι αν αυξήσουμε πάρα πολύ το μέγεθος θα πάρουμε περισσότερα total cycles, δηλαδή χειρότερο αποτέλεσμα. Αυτό συμβαίνει επειδή αν το tile είναι πολύ μεγάλο δε χωράει στη μνήμη cache και έχουμε πολλά cache misses, άρα το πρόγραμμα καθυστερεί περισσότερο.

β) Στη συνέχεια να βρεθεί το μέγεθος του πίνακα δεδομένων και ο αριθμός των προσπελάσεων που γίνονται σε αυτόν

Οι πίνακες που χρησιμοποιήσαμε είναι οι current_y, current_u, current_v, transformed_y, transformed_u, transformed_v.

Οι πίνακες current περιέχουν τις τιμές για τα pixel της εικόνας, τις οποίες διαβάζουν οι πίνακες transformed. Οι πίνακες transformed αρχικά μηδενίζονται (θέτουμε όλα τα στοιχεία τους με μαύρο χρώμα) και στη συνέχεια σε αυτούς αποθηκεύονται τα μετασχηματισμένα σημεία, δηλαδή το αποτέλεσμα του συσχετισμένου μετασχηματισμού. Τέλος, η τελική εικόνα αποθηκεύεται στους πίνακες current. Σε κάθε περίπτωση οι πίνακες είναι 3, επειδή οι εικόνες μας είναι σε μορφή yuv, οπότε οι 3 πίνακες αναφέρονται στα 3 διαφορετικά κανάλια.

Όλοι οι παραπάνω πίνακες είναι μεγέθους MxN, δηλαδή όσο και η αρχική εικόνα.

Οι προσπελάσεις που γίνονται σε αυτούς είναι:

Για current_y, current_u, current_v, διαβάζω MxN φορές και γράφω MxN φορές στον καθένα, άρα MxNx3 για διάβασμα και MxNx3 για γράψιμο, συνολικά (MxNx3)x2 προσπελάσεις.

Ακριβώς τα ίδια ισχύουν για τους transformed, απλά επειδή αυτούς τους μηδενίζουμε αρχικά έχουμε επιπλέον MxNx3 για το γράψιμο, άρα (MxNx3)x2 για διάβασμα και MxNx3 για γράψιμο, συνολικά (MxNx3)x3 προσπελάσεις.

Όλα όσα αναφέραμε επαληθεύονται από τους μετρητές (counters) που χρησιμοποιήσαμε στον κώδικα (για την εικόνα με το δέντρο).

ARM7TDMI - Console

```
Image Written
Times for reading and writting in current y, u, v: 1107072
Times for reading and writting in transformed matrix: 1660608
```

γ) Ολοκληρώνοντας τις δυνατές εκτελέσεις-βελτιστοποιήσεις να πραγματοποιηθούν μετρήσεις για την ταχύτητα εκτέλεσης του αλγορίθμου με τη βοήθεια του προσομοιωτή ARMulator του επεξεργαστή ARM

Στις παρακάτω εικόνες φαίνονται οι κύκλοι για τους οποίους εκτελείται ο αλγόριθμος για το αρχικό πρόγραμμα και για το βελτιωμένο πρόγραμμα για διαφορετικές εικόνες.

Εικόνα 1

Internal Variables | Statistics

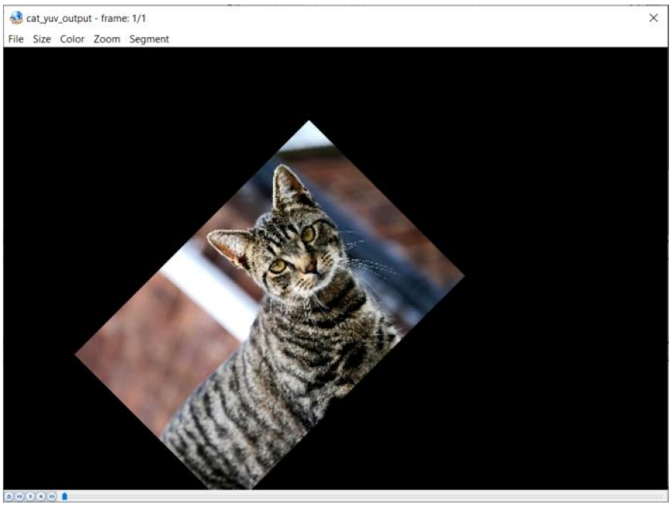
Reference P...	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	219564722	300925972	235330000	51993394	28307687	0	315631081

Debugger Internals

Internal Variables | Statistics

Reference P...	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	167581976	237388349	177181816	48642691	21438531	0	247263038

Εικόνα 2



Debugger Internals

Internal Variables | Statistics

Reference P...	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	198752439	271587312	212848295	46552600	27497688	0	286898583

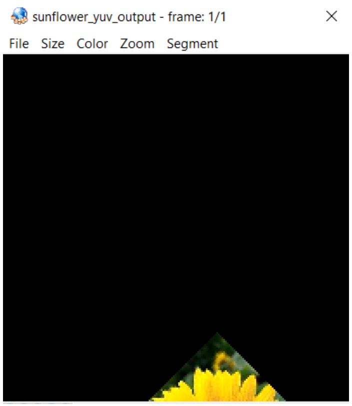
Debugger Internals

Internal Variables | Statistics

Reference P...	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	153208894	217268630	161690542	44677684	21382422	0	227750648

Target | Image | Files | Class

Εικόνα 3



Internal Variables | Statistics

Reference P...	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	47531300	65321051	51106559	11314133	6464428	0	68885120

Debugger Internals

Internal Variables | Statistics

Reference P...	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	36058185	51460965	38248689	10718691	4893304	0	53860684

Βιβλιογραφία

https://en.wikipedia.org/wiki/Affine_transformation

http://homepages.inf.ed.ac.uk/rbf/HIPR2/affine.htm

https://people.computing.clemson.edu/~dhouse/courses/401/notes/affines-matrices.pdf

https://medium.com/@junfeng142857/affine-transformation-why-3d-matrix-for-a-2d-transformation-8922b08bce75