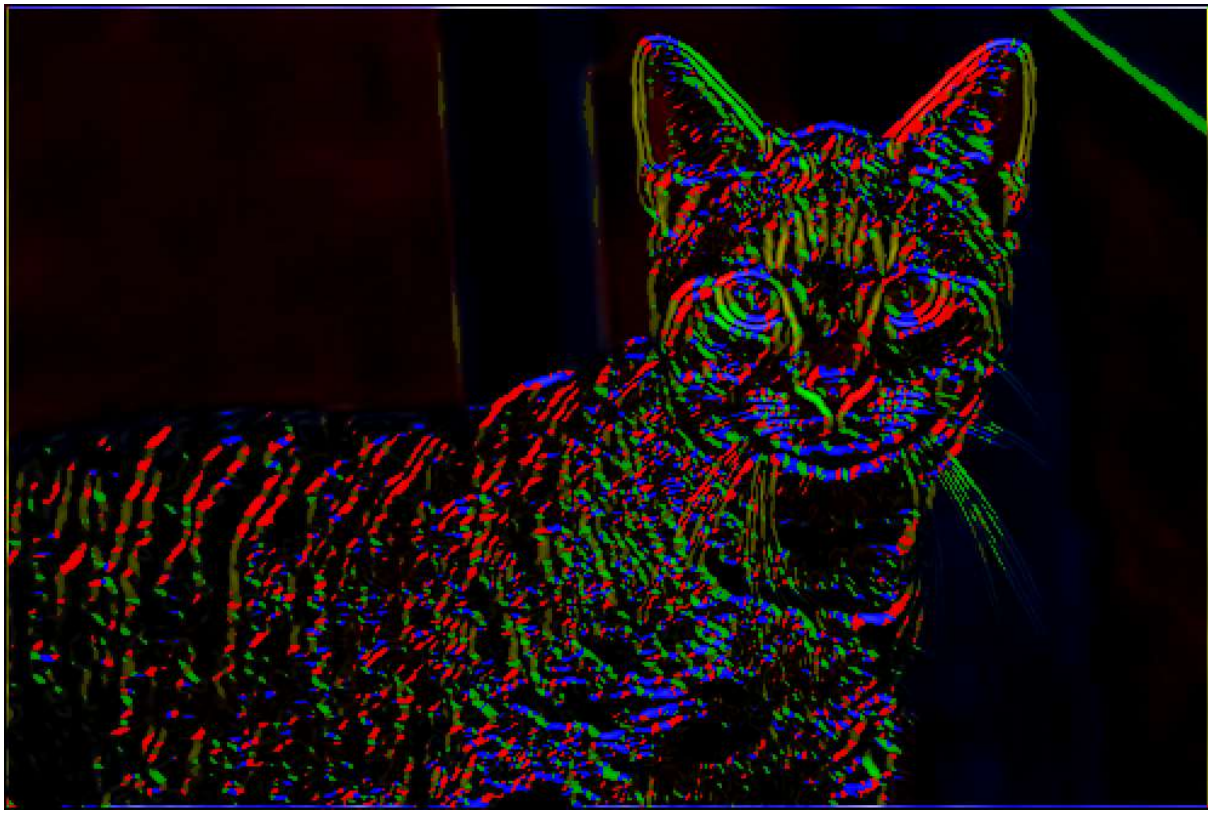
Σχεδιασμός Ενσωματωμένων Συστημάτων

Εργασία 1η

Μαζαράκης Περικλής Α.Μ.: 57595

Παπαδόπουλος Αριστείδης Α.Μ.:57576



Πρόγραμμα προ βελτιώσεων

Αρχικά, χρησιμοποιήσαμε μερικά στοιχεία του κώδικα του βοηθητικού υλικού για την υλοποίηση του κώδικα μας. Συγκεκριμένα, χρησιμοποιήσαμε τις συναρτήσεις read() και write() ώστε να διαβάσουμε την εικόνα και να την εμφανίσουμε στο τέλος, όπως και τον κώδικα για το armulator.

```
/* code for armulator*/
#pragma arm section zidata="ram"
int current_y[N][M];
int current_u[N][M];
int current_v[N][M];
```

Εικόνα 1. Code for armulator

```
void read()
{
  FILE *frame_c;
  if((frame_c=fopen(filename,"rb"))==NULL)
  {printf("current frame doesn't exist\n");exit(-1);}
  for(i=0;i<N;i++){
    for(j=0;j<M;j++)
    {current_y[i][j]=fgetc(frame_c);}}
  for(i=0;i<N;i++){
    for(j=0;j<M;j++)
    {current_u[i][j]=fgetc(frame_c);}}
  for(i=0;i<N;i++){
    for(j=0;j<M;j++)
    {current_v[i][j]=fgetc(frame_c);}}
  fclose(frame_c);
}
```

Εικόνα 2. Read() function

```
void write()
{
  FILE *frame_edges;
  frame_edges=fopen(file_edges,"wb");
  for(i=0;i<N;i++){
    for(j=0;j<M;j++)
    {fputc(current_y[i][j],frame_edges);}}
  for(i=0;i<N;i++){
    for(j=0;j<M;j++)
    {fputc(current_u[i][j],frame_edges);}}
  for(i=0;i<N;i++){
    for(j=0;j<M;j++)
    {fputc(current_v[i][j],frame_edges);}}
  fclose(frame_edges);
}
```

Εικόνα 3. Write() function

Στη συνέχεια, όπως φαίνεται και από το screenshot που θα βρίσκεται παρακάτω, οι δηλώσεις ορισμένων μεταβλητών που παίρνουν μέρος στην main βρίσκονται ακριβώς από πάνω της. Ο λόγος είναι πως ενώ σε ένα σύγχρονο compiler (χρησιμοποιούμε το Atom για τη γραφή του κώδικα σε C), δεν υπάρχει θέμα με το να θέσουμε τις μεταβλητές ως global στην αρχή του κώδικα, όταν τον περάσαμε στο CodeWarrior, εμφάνιζε error. Μετά από σχετική αναζήτηση, είδαμε ότι επειδή ο compiler του CodeWarrior είναι προηγούμενης έκδοσης, οι δηλώσεις πρέπει να γίνονται στην αρχή κάθε block, πρωτού καταχωρηθεί κάποια άλλη εντολή. Για αυτό και υπάρχει πάνω από τη main για να κάνουμε εύκολα την όποια αλλαγή μεταξύ του κώδικα που χρησιμοποιούμε στο Atom και αυτού του armulator.

```
int Ix[N][M],Iy[N][M];
float rescaled_values[N][M];
int gauss_img[N][M];
float min,max;
int sobel1[3][3],sobel2[3][3],gauss[3][3];
float dI[N][M],theta[N][M];

int main() {
```

Εικόνα 4. Initialize variables

Τώρα στη main έχουμε δηλώσει τόσο το φίλτρο gauss όσο και τα φίλτρα Sobel με τρόπο τέτοιο ώστε να μην έχουμε θέμα με το armulator. Ξεκινάμε με τη συνέλιξη του φίλτρου gauss με την grayscale εικόνα, όπου η grayscale είναι το κανάλι Y (υπεύθυνο για τη φωτεινότητα). Στη συνέχεια χρησιμοποιούμε τα φίλτρα Sobel για να βρούμε τις ακμές σε κάθε διάσταση και να βρούμε την ένταση του κάθε πίξελ από το dI. Υπολογίζουμε τη γωνία θ (που θα χρειαστεί για τον καθορισμό του χρώματος σε κάθε προσανατολισμό) και την μετατρέπουμε σε μοίρες από rad. Θέτουμε 2 τιμές max, min μέσω των οποίων βρίσκουμε την μέγιστη και την ελάχιστη τιμή για το dI. Στη συνέχεια, οι τιμές που προκύπτουν παίρνουν μέρος στη διαδικασία της μετατροπής της κλίμακας, η οποία ορίζεται ως εξής:

$$\text{νέα τιμή} = \frac{\text{παλιά τιμή} - \text{ελάχιστο παλιάς κλίμακας}}{\text{μέγιστο παλιάς κλίμακας} - \text{ελάχιστο παλιάς κλίμακας}} * 255$$

στα όρια τα οποία αναγράφονται στο βήμα 5 της εκφώνησης. Τέλος, αντικαθιστούμε τις νέες τιμές του Y βάση της νέας κλίμακας και καθορίζουμε τα όρια χρωματισμού, αποκλείοντας αρχικά ως πίξελ ακμών όσα έχουν φωτεινότητα πάνω από ένα όριο, το οποίο τέθηκε με αυθαίρετο τρόπο και για κάθε εικόνα είναι διαφορετικό. Έτσι, για όσα πίξελ θεωρούνται εικονοστοιχεία ακμών, ελέγχεται η γωνία θ για να βρεθεί το εύρος γωνιών στο οποίο ανήκει και τελικά το πίξελ λαμβάνει τις αντίστοιχες τιμές στα κανάλια U,V για κάθε προσανατολισμό.

```
int main() {

  read();
  printf("Create Gaussian Image...\n");
  gauss[0][0] = gauss[0][2] = gauss[2][0] = gauss[2][2] = 1;
  gauss[1][1] = 4;
  gauss[0][1] = gauss[1][0] = gauss[1][2] = gauss[2][1] = 2;
  //gauss[3][3] = {{1, 2, 1},{2, 4, 2},{1, 2, 1}}; // gaussian filter
  for (i = 0; i < N; i++)
  {
    for (j = 0; j < M; j++)
    {
      if (i==0) continue; //image boundaries, do nothing (instead of zero padding)
      else if (i==N-1) continue;
      else if (j==0) continue;
      else if (j==M-1) continue;
      else { //convolve gauss filter in grayscale image,grayscale is the Y Channel
      gauss_img[i][j] = current_y[i-1][j-1] * gauss[0][0] + current_y[i-1][j] * gauss[0][1] + current_y[i-1][j+1] * gauss[0][2]
      + current_y[i][j-1] * gauss[1][0] + current_y[i][j] * gauss[1][1] + current_y[i][j+1] * gauss[1][2]
      + current_y[i+1][j-1] * gauss[2][0] + current_y[i+1][j] * gauss[2][1]+ current_y[i+1][j+1] * gauss[2][2];
      }
    }
  }
```

Εικόνα 5. Convolution of grayscale and gauss filter

```
for (i = 0; i < N; i++) {
  for (j = 0; j < M; j++){
    if (i==0) continue; //oriakes sinthikes, instead of zero pad
    else if (i==N-1)continue;
    else if (j==0) continue;
    else if (j==M-1)continue;
    else //determine gradient in each dimension, then calculate angle and magnitude values
    { // convolve gaussian image with sobel filters
      //Y axis edges
      Ix[i][j] = gauss_img[i-1][j-1] * sobel1[2][2] + gauss_img[i-1][j] * sobel1[2][1] + gauss_img[i-1][j+1] * sobel1[2][0]
        + gauss_img[i][j-1] * sobel1[1][2] + gauss_img[i][j] * sobel1[1][1] + gauss_img[i][j+1] * sobel1[1][0]
        + gauss_img[i+1][j-1] * sobel1[0][2] + gauss_img[i+1][j] * sobel1[0][1] + gauss_img[i+1][j+1] * sobel1[0][0];
      //X axis edges
      Iy[i][j] = gauss_img[i-1][j-1] * sobel2[2][2] + gauss_img[i-1][j] * sobel2[2][1] + gauss_img[i-1][j+1] * sobel2[2][0]
        + gauss_img[i][j-1] * sobel2[1][2] + gauss_img[i][j] * sobel2[1][1] + gauss_img[i][j+1] * sobel2[1][0]
        + gauss_img[i+1][j-1] * sobel2[0][2] + gauss_img[i+1][j] * sobel2[0][1] + gauss_img[i+1][j+1] * sobel2[0][0];
```

Εικόνα 6. Convolution of gaussian image with sobel filters

```
        dI[i][j] = pow((pow(Ix[i][j],2)+pow(Iy[i][j],2)),0.5); // calculate magnitude values
        theta[i][j] = atan2(Ix[i][j], Iy[i][j]); // calculate and round angle values
        theta[i][j] = theta[i][j] * (180/3.14159); // radians to degrees, error using M_MPI in armulator
        if (theta[i][j] < 0) theta[i][j] += 180; // convert all angles to positive angles
        if (dI[i][j] > max) max = dI[i][j];//find max value, later to be used for scaling
        if (dI[i][j] < min) min = dI[i][j];// find min value, later to be used for scaling
    }
  }}
```

Εικόνα 7. Calculate magnitude, theta and max/min

```
  for (i = 0; i < N; i++)
  {
    for (j = 0; j < M; j++)
    {
    rescaled_values[i][j] = ((dI[i][j]-min)/(max-min))*255;//scale the magnitude values,convert to integers
    current_y[i][j] = rescaled_values[i][j]; // New Y Channel values
    //determine edge direction and color accordingly
       if (current_y[i][j] > 45){//authaireto orio,45 gia car,30 gia cat,30 gia sunflower
          if (theta[i][j] >= 0 && theta[i][j] < 22.5){ //horizontal edge
                  current_u[i][j] = 255;//blue UV channel values
                  current_v[i][j] = 107;}
          else if (theta[i][j] >= 157.5 && theta[i][j] <= 180) {//horizontal edge
                  current_u[i][j] = 255;//blue UV channel values
                  current_v[i][j] = 107;}
          else if (theta[i][j] >= 22.5 && theta[i][j] < 67.5) {//inclined edge
                  current_u[i][j] = 84;//red UV channel values
                  current_v[i][j] = 255;}
          else if (theta[i][j] >= 67.5 && theta[i][j] < 112.5) {//vertical edge
                  current_u[i][j] = 16;//yellow UV channel values
                  current_v[i][j] = 146;}
          else if (theta[i][j] >= 112.5 && theta[i][j] < 157.5) {//inclined edge
                  current_u[i][j] = 43;//green UV channel values
                  current_v[i][j] = 21;}
          else continue;}
    }
  }
  write();
  return 0;
}
```

Εικόνα 8. Rescale magnitude, and color accordingly

Βελτιστοποιημένο πρόγραμμα

- **Απλές βελτιστοποιήσεις**
  - Αντικατάσταση της pow
  - Μετατροπή των if σε switch-case
  - Δημιουργία συναρτήσεων
  - Μικροαλλαγές στον κώδικα
  - Zero padding

Αρχικά, αντικαταστήσαμε την pow με τον αναλυτικό τύπο της δύναμης (δηλαδή πολλαπλασιασμό του όρου με τον εαυτό του). Με αυτόν τον τρόπο καταφέραμε μείωση στους κύκλους του ρολογιού αφού δε χρειάζεται η είσοδος στη βιβλιοθήκη math για την πράξη της δύναμης. Παρομοίως, επιχειρήσαμε το να επιτύχουμε το ίδιο πράγμα με την κλήση της συνάρτησης της τετραγωνικής ρίζας. Συγκεκριμένα, υλοποιήσαμε δική μας συνάρτηση με δύο διαφορετικούς τρόπους υπολογισμού της ρίζας αλλά δεν είχαμε τα αντίστοιχα αποτελέσματα και οι κύκλοι αυξάνονταν. Φυσικά, το αν θα αυξηθούν οι κύκλοι εξαρτάται από τον τρόπο υπολογισμού της τετραγωνικής ρίζας, αλλά οι δύο υλοποιήσεις που δοκιμάσαμε δεν είχαν το επιθυμητό αποτέλεσμα. Προσπάθεια υπήρξε και για την αποφυγή του atan, αλλά οι προσεγγιστικοί τρόποι υπολογισμού του atan δεν απέδιδαν την επιθυμητή ακρίβεια, οπότε επιλέξαμε να χρησιμοποιήσουμε την έτοιμη συνάρτηση από το header math.h.

```
dI[i][j] = pow((pow(Ix[i][j],2)+pow(Iy[i][j],2)),0.5);
```

Εικόνα 9. Pow

```
dI[i][j] = sqrt((Ix[i][j] * Ix[i][j]) + (Iy[i][j]*Iy[i][j]));
```

Εικόνα 10. Manual power

Σαν αφετηρία για τις μετρήσεις μας παραθέτουμε τους αρχικούς κύκλους ρολογιού που έκανε ο κώδικας που γράψαμε πριν τις τροποποιήσεις και τους νέους κύκλους μετά από κάθε τροποποίηση.

Debugger Internals

Internal Variables | Statistics

Refere...	Instru...	Core_C...	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	402929566	576826639	455853689	90516745	100104896	0	646475330

Εικόνα 11. Initial cycles

Debugger Internals

Internal Variables | Statistics

Refere...	Instru...	Core_C...	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	362037533	511993643	405190444	79116802	93248095	0	577555341

Εικόνα 12. Manual pow cycles

Debugger Internals

Internal Variables | Statistics

Refere...	Instru...	Core_C...	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	827301160	1130255053	926421004	154387547	208764097	0	1289572648

Εικόνα 13. Manual pow and sqrt

Debugger Internals

Internal Variables | Statistics

Refere...	Instru...	Core_C...	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	331070281	466946923	366751885	73191058	98140883	0	538083826

Εικόνα 14. Manual pow and sqrt(2ος τρόπος)

```
float sq_rt(float pixel){
// also known as Newton-Raphson method
  c = (pixel/2)+1;
  c1 = (c+(pixel/c))/2;
  if (c1 < c)
  {
    c = c1;
    c1 = (c + (pixel/c))/2;
  }
  return c;
}
```

Εικόνα 15. Sq_rt function

Εικόνα 16. Sq_rt function results

Όπως γίνεται αντιληπτό, μπορεί να κερδίζουμε πολλά από τους κύκλους του ρολογιού, όμως το τελικό αποτέλεσμα δεν είναι αντιπροσωπευτικό αυτού που θέλουμε να επιτύχουμε. Συνεπώς από τους παραπάνω τρόπους να αποφύγουμε την χρήση εντολών της βιβλιοθήκης math.h, αποδεκτά αποτελέσματα είχε μόνο η μετατροπή της pow.

Στη συνέχεια, μετατρέψαμε όσες if ήταν δυνατό με switch-case. Συγκεκριμένα, βρήκαμε πως αν βάλουμε στην κορυφή της switch το πιο συχνό “case” που πραγματοποιείται, κερδίζουμε μερικούς κύκλους και από αυτό, καθώς εκτελούνται λιγότεροι έλεγχοι.

```
for (j = 0; j < M; j++)
{
  if (i==0) continue; //image boundaries, do nothing (instead of zero padding)
  else if (i==N-1) continue;
  else if (j==0) continue;
  else if (j==M-1) continue;
  else { //convolve gauss filter in grayscale image,grayscale is the Y Channel
  gauss_img[i][j] = current_y[i-1][j-1] * gauss[0][0] + current_y[i-1][j] * gauss[0][1] + current_y[i-1][j+1] * gauss[0][2]
  + current_y[i][j-1] * gauss[1][0] + current_y[i][j] * gauss[1][1] + current_y[i][j+1] * gauss[1][2]
  + current_y[i+1][j-1] * gauss[2][0] + current_y[i+1][j] * gauss[2][1]+ current_y[i+1][j+1] * gauss[2][2];
```

Εικόνα 17. Original code with if-else

```
for (i = 0; i < N; i++)
{
  switch (i) {
    default:{
        for (j = 0; j < M; j++)
        { switch (j) {
          default:{ //convolve gauss filter in grayscale image,grayscale is the Y Channel
            gauss_img[i][j] = current_y[i-1][j-1] * gauss[0][0] + current_y[i-1][j] * gauss[0][1] + current_y[i-1][j+1] * gauss
            + current_y[i][j-1] * gauss[1][0] + current_y[i][j] * gauss[1][1] + current_y[i][j+1] * gauss[1][2]
            + current_y[i+1][j-1] * gauss[2][0] + current_y[i+1][j] * gauss[2][1]+ current_y[i+1][j+1] * gauss[2][2];}
          case 0:continue;
          case M-1:continue;}
  case 0:continue;
  case N-1:continue;}
```

Εικόνα 18. New code with switch-case

Debugger Internals

Internal Variables | Statistics

Refere...	Instru...	Core_C...	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	363926047	516180176	408574865	79919307	93123925	0	581618097

Εικόνα 19. Cycles with the switch implementation

Επίσης, δημιουργήσαμε συναρτήσεις τόσο για τα φίλτρα, όσο και για την κλίμακα και το χρωματισμό, με σκοπό να υπάρχουν λιγότερες γραμμές κώδικα μέσα στη main και να γίνονται απλώς κλήσεις των συναρτήσεων.

```
int main() {

  read();
  printf("Create Gaussian Image...\n");
  gauss[0][0] = gauss[0][2] = gauss[2][0] = gauss[2][2] = 1;
  gauss[1][1] = 4;
  gauss[0][1] = gauss[1][0] = gauss[1][2] = gauss[2][1] = 2;
  //gauss[3][3] = {{1, 2, 1},{2, 4, 2},{1, 2, 1}}; // gaussian filter
  for (i = 0; i < N; i++)
  {
    for (j = 0; j < M; j++)
    {
      if (i==0) continue; //image boundaries, do nothing (instead of zero padding)
      else if (i==N-1) continue;
      else if (j==0) continue;
      else if (j==M-1) continue;
      else { //convolve gauss filter in grayscale image,grayscale is the Y Channel
      gauss_img[i][j] = current_y[i-1][j-1] * gauss[0][0] + current_y[i-1][j] * gauss[0][1] + current_y[i-1][j+1] * gauss[0][2]
      + current_y[i][j-1] * gauss[1][0] + current_y[i][j] * gauss[1][1] + current_y[i][j+1] * gauss[1][2]
      + current_y[i+1][j-1] * gauss[2][0] + current_y[i+1][j] * gauss[2][1]+ current_y[i+1][j+1] * gauss[2][2];
      }
    }
  }
```

Εικόνα 20. Initial code in the main

```
int main() {
  read();
  gauss_filter();
  sobel_filtering();
  scale_color();
  write();
  return 0;
}
```

Εικόνα 21. Main after the functions

Debugger Internals

Internal Variables | Statistics

Refere...	Instr...	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	368689029	522694245	414035100	80739530	93357467	0	588132097

Εικόνα 22. New cycles with clear main

Ακολούθησαν ορισμένες μικροαλλαγές στον κώδικα τις οποίες θα περιγράψουμε παρακάτω. Μια εκ τις οποίες, είναι η αλλαγή του ελέγχου της for, που μας γλυτώνει μερικούς κύκλους ρολογιού, αν από for(i=0; i<N; i++) γίνει της μορφής for(i=N; i--), αφού δε χρειάζεται σε κάθε κύκλο να γίνεται η σύγκριση του i με το N και έχοντας θέσει το i ως integer, δεν πρόκειται να πέσουμε ποτέ κάτω από το 0.Ουσιαστικά, με αυτόν τον τρόπο αποφεύγουμε τους ελέγχους που υλοποιούνται στο τέλος ενός iteration του loop. Επιπλέον, η εύρεση της γωνίας theta και μετατροπή της σε μοίρες γίνονται στην ίδια γραμμή και κόβουν μερικούς κύκλους από την εκτέλεση του προγράμματος.

Επιπρόσθετα, παρατηρήσαμε πως με μετατόπιση μερικών υπολογισμών/αναθέσεων σε διαφορετικούς βρόγχους επανάληψης από αυτούς που ήταν αρχικά, καταφέραμε να μειώσουμε περαιτέρω μερικούς κύκλους ρολογιού. Τέλος, διαφοροποιώντας την παρακάτω if-else-if statement καταφέραμε μια επιπλέον μικρή μείωση στους απαιτούμενους κύκλους ρολογιού.

```
if (theta[i][j] >= 0 && theta[i][j] < 22.5){ //horizontal edge
        padded_y[i][j] = current_y[i][j];
        current_u[i][j] = 255;//blue UV channel values
        current_v[i][j] = 107;}
else if (theta[i][j]>= 157.5 && theta[i][j] <= 180) {//horizontal edge
  padded_y[i][j] = current_y[i][j];
        current_u[i][j] = 255;//blue UV channel values
        current_v[i][j] = 107;}
else if (theta[i][j] >= 22.5 && theta[i][j] < 67.5) {//inclined edge
  padded_y[i][j] = current_y[i][j];
        current_u[i][j] = 84;//red UV channel values
        current_v[i][j] = 255;}
else if (theta[i][j] >= 67.5 && theta[i][j] < 112.5) {//vertical edge
  padded_y[i][j] = current_y[i][j];
        current_u[i][j] = 16;//yellow UV channel values
        current_v[i][j] = 146;}
else if (theta[i][j] >= 112.5 && theta[i][j] < 157.5) {//inclined edge
  padded_y[i][j] = current_y[i][j];
        current_u[i][j] = 43;//green UV channel values
        current_v[i][j] = 21;}
else continue;}
```

Εικόνα 23. Previous If-else-if loop

```
if (theta[i][j] >= 0 && theta[i][j] < 22.5){ //horizontal edge
        current_u[i][j] = 255;//blue UV channel values
        current_v[i][j] = 107;}
else if (theta[i][j] >= 157.5 && theta[i][j] <= 180) {//horizontal edge
        current_u[i][j] = 255;//blue UV channel values
        current_v[i][j] = 107;}
else if (theta[i][j] >= 22.5 && theta[i][j] < 67.5) {//inclined edge
        current_u[i][j] = 84;//red UV channel values
        current_v[i][j] = 255;}
else if (theta[i][j] >= 67.5 && theta[i][j] < 112.5) {//vertical edge
        current_u[i][j] = 16;//yellow UV channel values
        current_v[i][j] = 146;}
else  { // last possible outcome
        current_u[i][j] = 43;//inclined edge
        current_v[i][j] = 21;//green UV channel values
      }
```

Εικόνα 24. New if-else-if loop

Το zero-padding, αποτελεί ιδιαίτερη περίπτωση βελτιστοποίησης. Ειδικότερα, αυτό που εξοικονομείται είναι ο έλεγχος σε κάθε συνέλιξη, αν έχουν ξεπεραστεί τα όρια της εικόνας ή όχι, από την στιγμή που με το zero-padding προστίθενται μηδενικά, για να γίνει κανονικά η συνέλιξη. Όμως, στην ουσία δημιουργείται ένας νέος πίνακας(διαστάσεων μεγαλύτερων της αρχικής εικόνας) και έτσι διατρέχεται εκ νέου με περισσότερες επαναλήψεις, ενώ επιπλέον καταλαμβάνεται μεγαλύτερο μέρος στην μνήμη(για τις μετέπειτα εργασίες). Έτσι, παρόλο που προσπαθήσαμε να υλοποιήσουμε zero-padding με τον τρόπο που ακολουθεί, όπως φαίνεται, τα αποτελέσματα δεν ήταν ικανοποιητικά.

```
void zeropad(){
  for (x=N;x--;){
  if (x==0 || x==N+1) padded_y[x][j] = 0;
  else {for (y =M+2; y--;){
    if (y==0 || y == M+1) padded_y[x][y] =0;
    else padded_y[x][y] = current_y[x][y];}
}}}
```

Εικόνα 25. Zero-Pad function

Debugger Internals

Internal Variables | Statistics

Refere...	Instru...	Core_C...	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	361740167	513216194	404648832	80689646	94093771	0	579432249

Εικόνα 26. Zero-pad results

- **Βελτιστοποιήσεις βρόχων**
  - Loop fission/fusion
  - Loop unrolling
  - Loop interchange
  - Loop collapsing
  - Loop inversion
  - Loop skewing
  - Loop tiling

Ξεκινώντας με το loop fission όπου πρακτικά «σπάμε» ένα loop σε 2 μικρότερες ώστε να παραλληλοποιήσουμε σε ένα βαθμό το loop και να πετύχουμε καλύτερους χρόνους. Αυτό έχει καλύτερη απόκριση σε πολυπύρηνα συστήματα που μπορούν να διασπάσουν μια δουλεία σε πολλές μικρότερες. Το loop fusion, δεν επιχειρήθηκε, καθότι για να υλοποιηθεί, οι δύο βρόγχοι που θα ενωθούν, χρειάζεται να έχουν τα ίδια όρια. Στην δική μας

περίπτωση, επειδή διατρέχουμε πίνακες που απεικονίζουν εικόνες και επιπρόσθετα, δεν είναι δυνατό να πραγματοποιηθεί loop fusion.

```
void scale_color(){
  for (i = 0; i < N; i++)
  {
    for (j = 0; j < M; j++)
    {
    current_y[i][j] = ((dI[i][j]-min)/(max-min))*255;; // New Y Channel values
    //determine edge direction and color accordingly
       if (current_y[i][j] > 45){//authaireto orio,45 gia car,30 gia cat,30 gia sunflower
          if (theta[i][j] >= 0 && theta[i][j] < 22.5){ //horizontal edge
                  current_u[i][j] = 255;//blue UV channel values
                  current_v[i][j] = 107;}
          else if (theta[i][j] >= 157.5 && theta[i][j] <= 180) {//horizontal edge
                  current_u[i][j] = 255;//blue UV channel values
                  current_v[i][j] = 107;}
          else if (theta[i][j] >= 22.5 && theta[i][j] < 67.5) {//inclined edge
                  current_u[i][j] = 84;//red UV channel values
                  current_v[i][j] = 255;}
          else if (theta[i][j] >= 67.5 && theta[i][j] < 112.5) {//vertical edge
                  current_u[i][j] = 16;//yellow UV channel values
                  current_v[i][j] = 146;}
          else if (theta[i][j] >= 112.5 && theta[i][j] < 157.5) {//inclined edge
                  current_u[i][j] = 43;//green UV channel values
                  current_v[i][j] = 21;}
          else continue;}
```

Εικόνα 27. Scale_color before fission

```
void scale_color(){
  printf("Scaling & coloring...\n");
  for (i = N; i--;)//fissioned loop
  {
    for (j = M; j=j-3;)//unrolled
    {
  current_y[i][j] = ((dI[i][j]-min)/(max-min))*255; // New Y Channel values//scale the magnitude values,convert to integers
  current_y[i][j-1] = ((dI[i][j-1]-min)/(max-min))*255;
  current_y[i][j-2] = ((dI[i][j-2]-min)/(max-min))*255;
    }
}
  for (i = N; i--;)
  {
    for (j = M; j--;)
    {
    //determine edge direction and color accordingly
       if (current_y[i][j] > 45){//authaireto orio,45 gia car,30 gia cat,30 gia sunflower
          if (theta[i][j] >= 0 && theta[i][j] < 22.5){ //horizontal edge
                  current_u[i][j] = 255;//blue UV channel values
                  current_v[i][j] = 107;}
          else if (theta[i][j] >= 157.5 && theta[i][j] <= 180) {//horizontal edge
                  current_u[i][j] = 255;//blue UV channel values
                  current_v[i][j] = 107;}
          else if (theta[i][j] >= 22.5 && theta[i][j] < 67.5) {//inclined edge
                  current_u[i][j] = 84;//red UV channel values
                  current_v[i][j] = 255;}
          else if (theta[i][j] >= 67.5 && theta[i][j] < 112.5) {//vertical edge
                  current_u[i][j] = 16;//yellow UV channel values
                  current_v[i][j] = 146;}
          else if (theta[i][j] >= 112.5 && theta[i][j] < 157.5) {//inclined edge
                  current_u[i][j] = 43;//green UV channel values
                  current_v[i][j] = 21;}
          else continue;}
```

Εικόνα 28. Scale_color with fission

Debugger Internals

Internal Variables | Statistics

Refere...	Instr...	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	360361777	510097277	402944322	79532286	93312345	0	575788953

Εικόνα 29. Cycles with fission.

Εν συνεχεία, με το loop unrolling, ουσιαστικά «ξετυλίγουμε» τη loop, ώστε ο μετρητής της τελευταίας να κάνει λιγότερες φορές update και να εκτελούνται λιγότερες διακλαδώσεις. Έτσι μειώνονται σημαντικά οι κύκλοι του ρολογιού. Στην περίπτωση μας φτάσαμε τα 2 unrolls. Όταν δοκιμάσαμε να πάμε στο 3ο είδαμε πως δεν ωφελούσε πλέον τον κώδικα μας και αύξανε τους κύκλους, για αυτό και παραμείναμε στα 2. Επειδή η εργασία ασχολείται με δυσδιάστατους πίνακες, είναι δυνατό το loop unrolling μόνο στην εμφωλευμένη loop, όπου διατρέχεται η μία διάσταση του πίνακα. Αξίζει επίσης να σημειωθεί, πως αν υλοποιούνταν το loop unrolling σε βρόγχους που περιέχουν if-statements, τότε οποιοδήποτε κέρδος από το loop unrolling καταναλίσκεται στους πολλαπλούς ελέγχους που προκύπτουν από το πλήθος των if-statement. Τέλος, εφαρμόζοντας την τεχνική του loop unrolling στις δοσμένες συναρτήσεις write-read βελτιστοποιείται ακόμα περισσότερο το τελικό αποτέλεσμα.

```
Ix[i][j] = gauss_img[i-1][j-1] * sobel1[2][2] + gauss_img[i-1][j] * sobel1[2][1] + gauss_img[i-1][j+1] * sobel1[2][0]
  + gauss_img[i][j-1] * sobel1[1][2] + gauss_img[i][j] * sobel1[1][1] + gauss_img[i][j+1] * sobel1[1][0]
  + gauss_img[i+1][j-1] * sobel1[0][2] + gauss_img[i+1][j] * sobel1[0][1] + gauss_img[i+1][j+1] * sobel1[0][0];
//X axis edges
Iy[i][j] = gauss_img[i-1][j-1] * sobel2[2][2] + gauss_img[i-1][j] * sobel2[2][1] + gauss_img[i-1][j+1] * sobel2[2][0]
  + gauss_img[i][j-1] * sobel2[1][2] + gauss_img[i][j] * sobel2[1][1] + gauss_img[i][j+1] * sobel2[1][0]
  + gauss_img[i+1][j-1] * sobel2[0][2] + gauss_img[i+1][j] * sobel2[0][1] + gauss_img[i+1][j+1] * sobel2[0][0];
```

Εικόνα 30. Ix, Iy image calculation before unrolling

```
//Y axis edges
  Ix[i][j] = gauss_img[i-1][j-1] * sobel1[2][2] + gauss_img[i-1][j] * sobel1[2][1] + gauss_img[i-1][j+1] * sobel1[2][0]
    + gauss_img[i][j-1] * sobel1[1][2] + gauss_img[i][j] * sobel1[1][1] + gauss_img[i][j+1] * sobel1[1][0]
    + gauss_img[i+1][j-1] * sobel1[0][2] + gauss_img[i+1][j] * sobel1[0][1] + gauss_img[i+1][j+1] * sobel1[0][0];
  Ix[i][j+1] = gauss_img[i-1][j] * sobel1[2][2] + gauss_img[i-1][j+1] * sobel1[2][1] + gauss_img[i-1][j+2] * sobel1[2][0]
      + gauss_img[i][j] * sobel1[1][2] + gauss_img[i][j+1] * sobel1[1][1] + gauss_img[i][j+2] * sobel1[1][0]
      + gauss_img[i+1][j] * sobel1[0][2] + gauss_img[i+1][j+1] * sobel1[0][1] + gauss_img[i+1][j+2] * sobel1[0][0];
  Ix[i][j+2] = gauss_img[i-1][j+1] * sobel1[2][2] + gauss_img[i-1][j+2] * sobel1[2][1] + gauss_img[i-1][j+3] * sobel1[2][0]
      + gauss_img[i][j+1] * sobel1[1][2] + gauss_img[i][j+2] * sobel1[1][1] + gauss_img[i][j+3] * sobel1[1][0]
      + gauss_img[i+1][j+1] * sobel1[0][2] + gauss_img[i+1][j+2] * sobel1[0][1] + gauss_img[i+1][j+3] * sobel1[0][0];
//X axis edges
  Iy[i][j] = gauss_img[i-1][j-1] * sobel2[2][2] + gauss_img[i-1][j] * sobel2[2][1] + gauss_img[i-1][j+1] * sobel2[2][0]
    + gauss_img[i][j-1] * sobel2[1][2] + gauss_img[i][j] * sobel2[1][1] + gauss_img[i][j+1] * sobel2[1][0]
    + gauss_img[i+1][j-1] * sobel2[0][2] + gauss_img[i+1][j] * sobel2[0][1] + gauss_img[i+1][j+1] * sobel2[0][0];
  Iy[i][j+1] = gauss_img[i-1][j] * sobel2[2][2] + gauss_img[i-1][j+1] * sobel2[2][1] + gauss_img[i-1][j+2] * sobel2[2][0]
      + gauss_img[i][j] * sobel2[1][2] + gauss_img[i][j+1] * sobel2[1][1] + gauss_img[i][j+2] * sobel2[1][0]
      + gauss_img[i+1][j] * sobel2[0][2] + gauss_img[i+1][j+1] * sobel2[0][1] + gauss_img[i+1][j+2] * sobel2[0][0];
  Iy[i][j+2] = gauss_img[i-1][j+1] * sobel2[2][2] + gauss_img[i-1][j+2] * sobel2[2][1] + gauss_img[i-1][j+3] * sobel2[2][0]
      + gauss_img[i][j+1] * sobel2[1][2] + gauss_img[i][j+2] * sobel2[1][1] + gauss_img[i][j+3] * sobel2[1][0]
      + gauss_img[i+1][j+1] * sobel2[0][2] + gauss_img[i+1][j+2] * sobel2[0][1] + gauss_img[i+1][j+3] * sobel2[0][0];
```

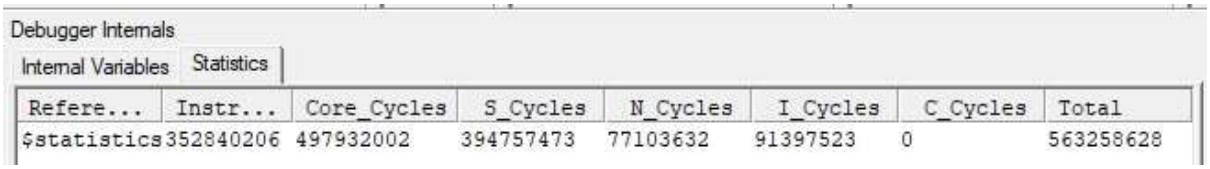
Εικόνα 31. Ix,Iy image calculation after unrolling

Debugger Internals

Internal Variables | Statistics

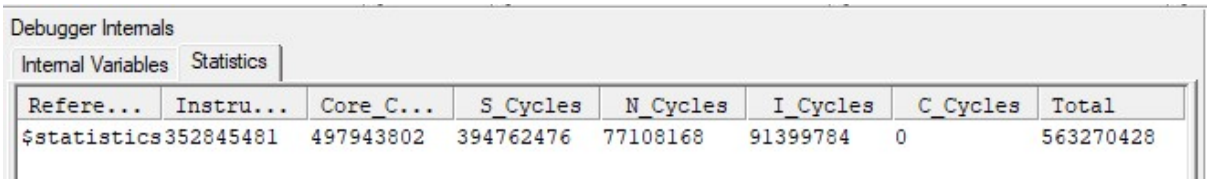
Refere...	Instr...	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	354011815	499448872	396203756	77011163	91738680	0	564953599

Εικόνα 32. First unroll



Εικόνα 33. Second unroll

Εν συνεχεία, το loop interchange επιχειρήθηκε όπου αυτό ήταν δυνατό, δηλαδή αλλάζοντας την εμφωλευμένη loop σε εξωτερική και το αντίστροφο. Όμως, αλλαγή του υπάρχοντος κώδικα όπως ορίζει η βελτιστοποίηση αυτή θα είχε ως αποτέλεσμα την ανάγνωση στοιχείων κατά στήλη και όχι κατά σειρά, όπως είναι τώρα, με αποτέλεσμα πιο αργή ανάγνωση δεδομένων και αποθήκευση αποτελεσμάτων , ανεξάρτητα από το γεγονός πως για την παρούσα εργασία θεωρούμε ιδανική μνήμη. Ενδεικτικά, τα αποτελέσματα της βελτιστοποίησης αυτής φαίνονται στην εικόνα που ακολουθεί.



Εικόνα 34. Loop interchange

Πράγματι, δεν βοήθησε στον κώδικα μας το loop interchange, καθώς αύξησε τους κύκλους κατά μια μικρή ποσότητα(περίπου 20.000 κύκλους).

Ακόμη, το loop collapsing, δεν είναι δυνατό να χρησιμοποιηθεί στον κώδικα μας, καθώς απαιτείται η πρόσβαση στα στοιχεία του πίνακα να γίνεται ανά ένα. Αντιθέτως, σε όλους τους βρόγχους επαναλήψεων της υλοποίησής μας, υπάρχει πολλαπλή πρόσβαση σε στοιχεία που είναι αποθηκευμένα σε πίνακες και σε διάφορες θέσεις του πίνακα, επομένως το loop collapsing είναι μη υλοποιήσιμο.

Επίσης, η βελτιστοποίηση του loop inversion χρησιμοποιείται όταν υπάρχει εντολή while,η οποία μετατρέπεται σε συνδυασμό if και do while με σκοπό να αποφευχθούν στάδια αναμονής, όμως στην δική μας υλοποίηση δεν υπάρχει τέτοια εντολή, επομένως είναι δυνατό να τεσταριστεί, χωρίς τροποποίηση του υπάρχοντος κώδικα.

Μία ακόμα βελτιστοποίηση που δοκιμάστηκε, είναι αυτή του loop skewing, κατά την οποία αλλάζει ο τρόπος δεικτοδότησης του πίνακα που διατρέχεται από το loop.Ωστόσο, επειδή τα αποτελέσματα δεν ήταν ικανοποιητικά, δεν χρησιμοποιήθηκε τελικά.

```
void scale_color(){
  for (i = 0; i < N; i++)
  {
    for (j = 0; j < M; j++)
    {
    current_y[i][j] = ((dI[i][j]-min)/(max-min))*255;; // New Y Channel values
```

Εικόνα 35. scale_color before loop skewing

```
void scale_color(){
  printf("Scaling & coloring...\n");
  for (i = 0;i<N; i++)//fissioned loop
  {
    for (j = i;j<N+M; j++)//unrolled
    {
  current_y[i][j-i] = ((dI[i][j-i]-min)/(max-min))*255; // New Y Channel values//scale the magnitude values,convert to integers
  //current_y[i][j-1] = ((dI[i][j-1]-min)/(max-min))*255;
  //current_y[i][j-2] = ((dI[i][j-2]-min)/(max-min))*255;
```

Εικόνα 36. scale_color after loop skewing

Debugger Internals

Internal Variables | Statistics

Refere...	Instru...	Core_C...	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	358232298	505262870	400463234	78363108	92561237	0	571387579

Εικόνα 37. Loop skewing

Τέλος, μια βελτιστοποίηση που δεν χρησιμοποιήθηκε είναι το loop tiling. Σύμφωνα με αυτήν, τα δεδομένα διατρέχονται σε block, με το μέγεθος του block να αποτελεί μια παράμετρο της βελτιστοποίησης αυτής. Έτσι, είναι δυνατό το μέγεθος του block να είναι ίσο με το μέγεθος της cache, επομένως να υπάρχουν συνεχώς cache hits και να αυξάνεται η ταχύτητα. Όμως, από την στιγμή που θεωρούμε ιδανική μνήμη, δεν αναμένουμε να βελτιώσει τους συνολικούς κύκλους. Υλοποιήθηκε όπου ήταν δυνατό και πράγματι, όπως φαίνεται παρακάτω δεν βοήθησε, αντιθέτως χειροτέρεψε τις απαιτήσεις σε κύκλους.

```
void magn_theta_calc(){
  printf("Calculating magnitude and theta\n");
  for (i = N; i--;) {//fissioned loop
    for (j = M; j--;){
      dI[i][j] = sqrt((Ix[i][j] * Ix[i][j]) + (Iy[i][j]*Iy[i][j])); // calculate magnitude values.
      theta[i][j] = atan2(Ix[i][j], Iy[i][j]) * (180/3.14159); // calculate and convert  angle values to degrees
      if (theta[i][j] < 0) theta[i][j] += 180; // convert all angles to positive angles
      if (dI[i][j] > max) max = dI[i][j];//find max value, later to be used for scaling
      else if (dI[i][j] < min) min = dI[i][j];
      else continue;
    }}
}
```

Εικόνα 39. Magn_theta_calc before loop tilling

```
int block = 64;
int w,z;
void magn_theta_calc(){
  printf("Calculating magnitude and theta\n");
  for (i = 0; i<N;i+=2) {//fissioned loop
    for (j = 0; j<M; j+=2){
      for (z = i; z<MIN(i+2,N);z++){
        for (w = j; w<MIN(j+2,M);w++){
      dI[z][w] = sqrt((Ix[z][w] * Ix[z][w]) + (Iy[z][w]*Iy[z][w])); // calculate magnitude values.
      theta[z][w] = atan2(Ix[z][w], Iy[z][w]) * (180/3.14159); // calculate and convert  angle values to degrees
      if (theta[z][w] < 0) theta[z][w] += 180; // convert all angles to positive angles
      if (dI[z][w] > max) max = dI[z][w];//find max value, later to be used for scaling
      else if (dI[z][w] < min) min = dI[z][w];
      else continue;}
    }}
}}
```

Εικόνα 40. Magn_theta_calc after loop tilling

Debugger Internals

Internal Variables | Statistics

Refere...	Instru...	Core_C...	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	357171363	506977139	399730700	79264329	93377358	0	572372387

Εικόνα 41. Loop tilling results

- **Τελικό Πρόγραμμα και αποτελέσματα**

Αν εφαρμοστούν συνδυαστικά όλες οι παραπάνω τεχνικές με τον κατάλληλο τρόπο, προκύπτουν οι τελικοί κύκλοι ρολογιού που χρειάζονται, όπως φαίνεται στην εικόνα παρακάτω.

Debugger Internals

Internal Variables | Statistics

Refere...	Instru...	Core_C...	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	350350893	492708098	392435326	74526018	91060110	0	558021454

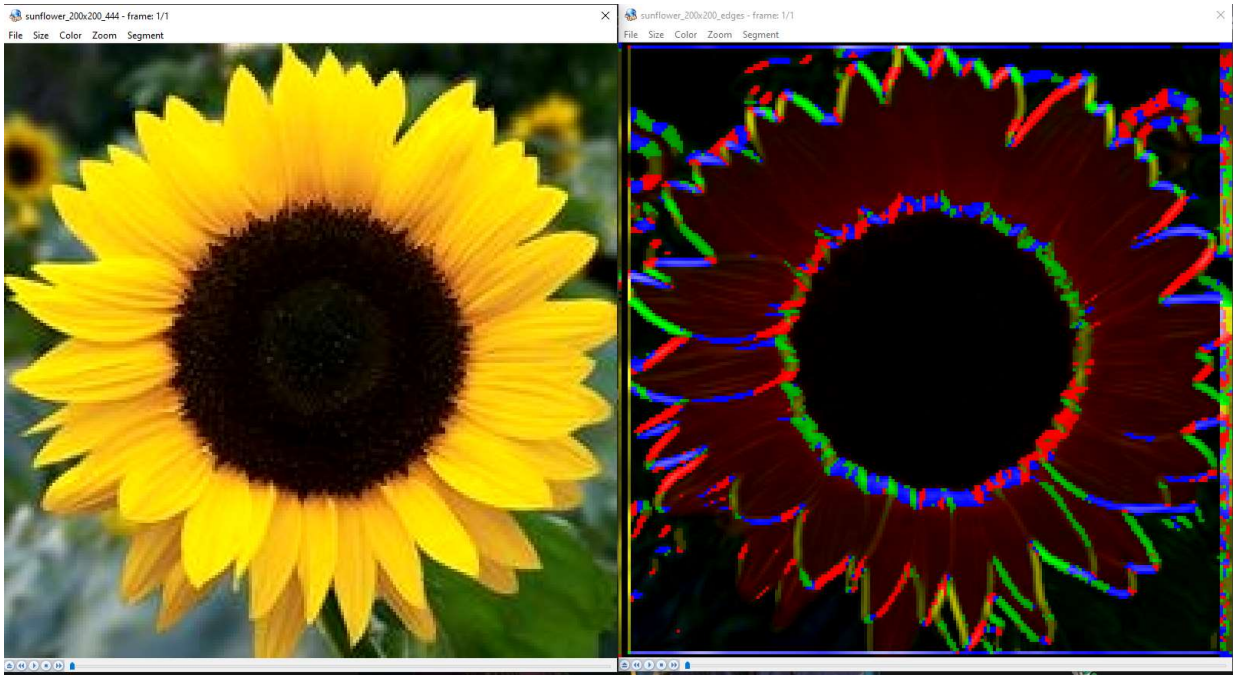
Εικόνα 42. Optimized code results for car image

Έτσι, έχουν εξοικονομηθεί συνολικά:

$$reduced\_cycles = initial\_clock\_cycles - final\_clock\_cycles \rightarrow$$

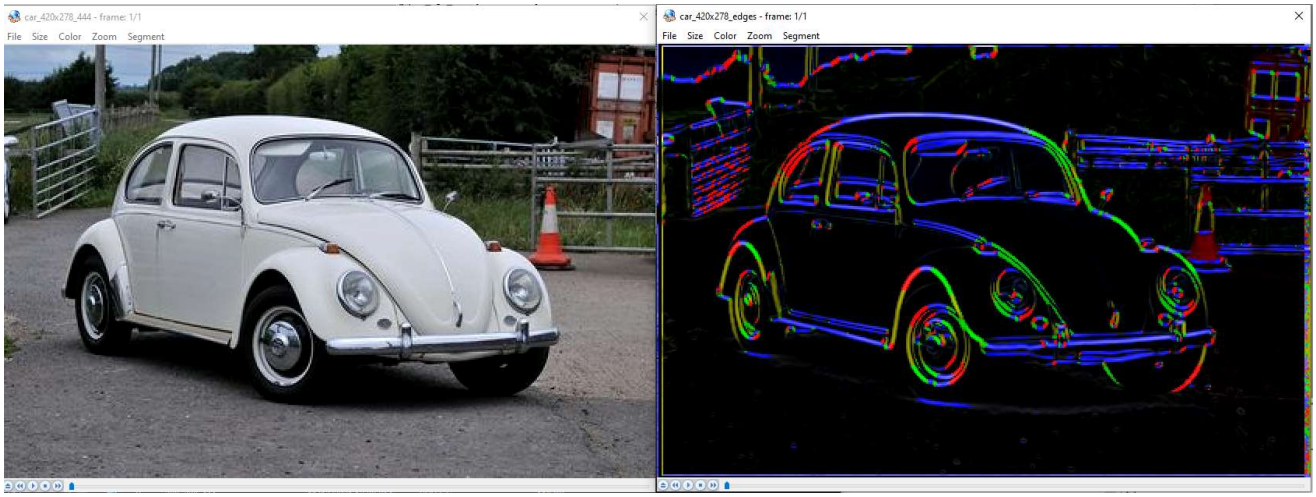
$$reduced\_cycles = 646.475.330 - 558.021.454 = 88.453.876$$

Οι κύκλοι, είναι αρκετοί και πάλι, δεδομένου όμως του προβλήματος που επιλύει ο αλγόριθμος, που περιέχει (χρήση μεγάλων δισδιάστατων πινάκων, συνέλιξη δύο διαστάσεων αρκετές φορές, έλεγχος για την φορά του κάθε πίξελ κ.τ.λ.) έχει γίνει ικανοποιητική βελτιστοποίηση. Ακολουθούν τα αποτελέσματα σε εικόνες.



Εικόνα 43. Εικόνα sunflower πριν και μετά τον αλγόριθμο

Εύκολα μπορεί να διαπιστώσει κανείς τις ακμές που εντοπίστηκαν στην δεξιά εικόνα, καθώς και την αστοχία του τρόπου υλοποίησης του αλγορίθμου, καθώς φαίνεται ότι οι περιοχές που αντιστοιχούν στα όρια της εικόνας (περιοχές που δεν υπολογίστηκε συνέλιξη), παρέμειναν ως είχαν και ο αλγόριθμος τις θεώρησε ως ακμές (οι συνεχείς γραμμές στα όρια της εικόνας).



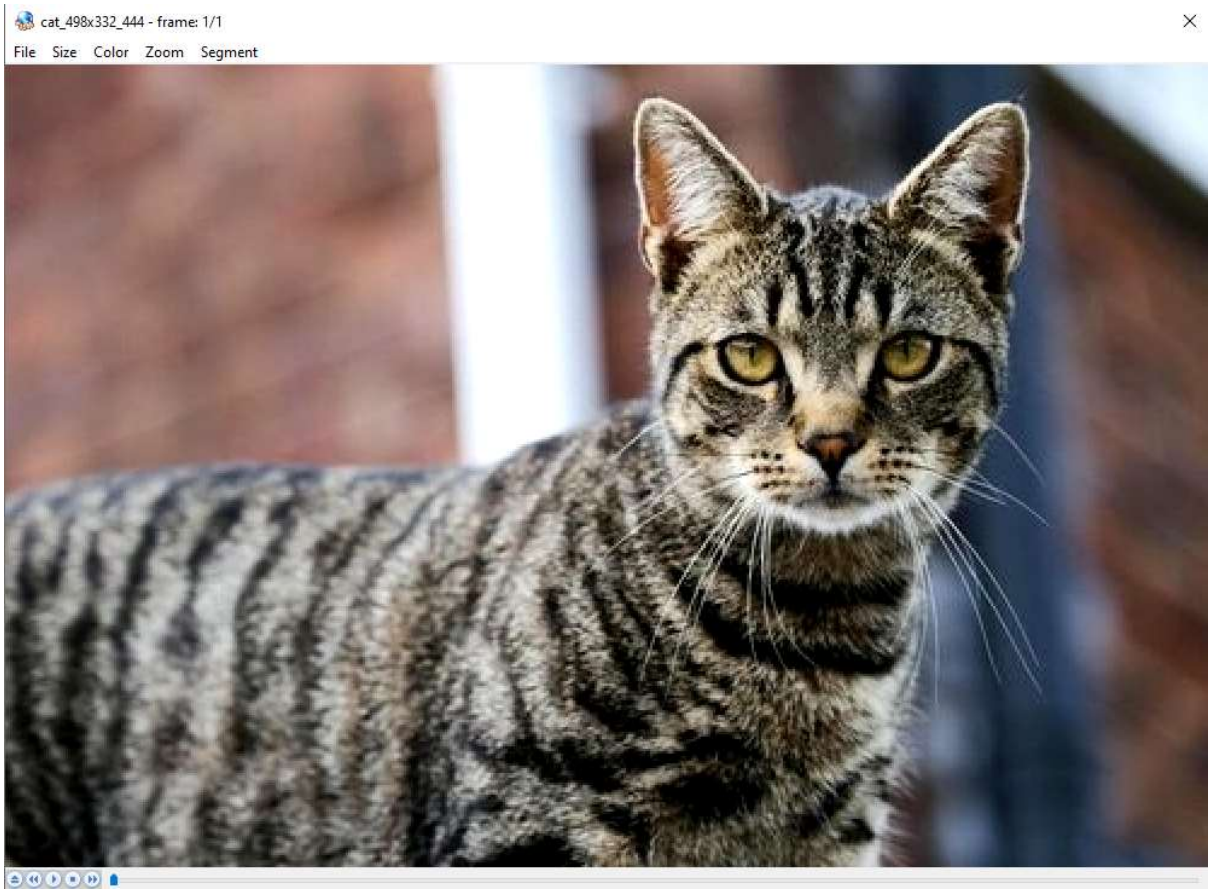
Εικόνα 44. Εικόνα car πριν και μετά τον αλγόριθμο

Παρατηρήσαμε ότι τρέχοντας την αρχική optimized υλοποίηση, είχε ως αποτέλεσμα memory violations για τις επόμενες, μεγαλύτερες σε διαστάσεις εικόνες. Η λύση ήταν να μειωθούν τα loop-unrolls, με αποτέλεσμα φυσικά τους αυξημένους κύκλους ρολογιού που χρειάζεται για να μην εμφανίζεται το συγκεκριμένο σφάλμα. Έτσι, παρακάτω παρατίθεται ένας πίνακας,

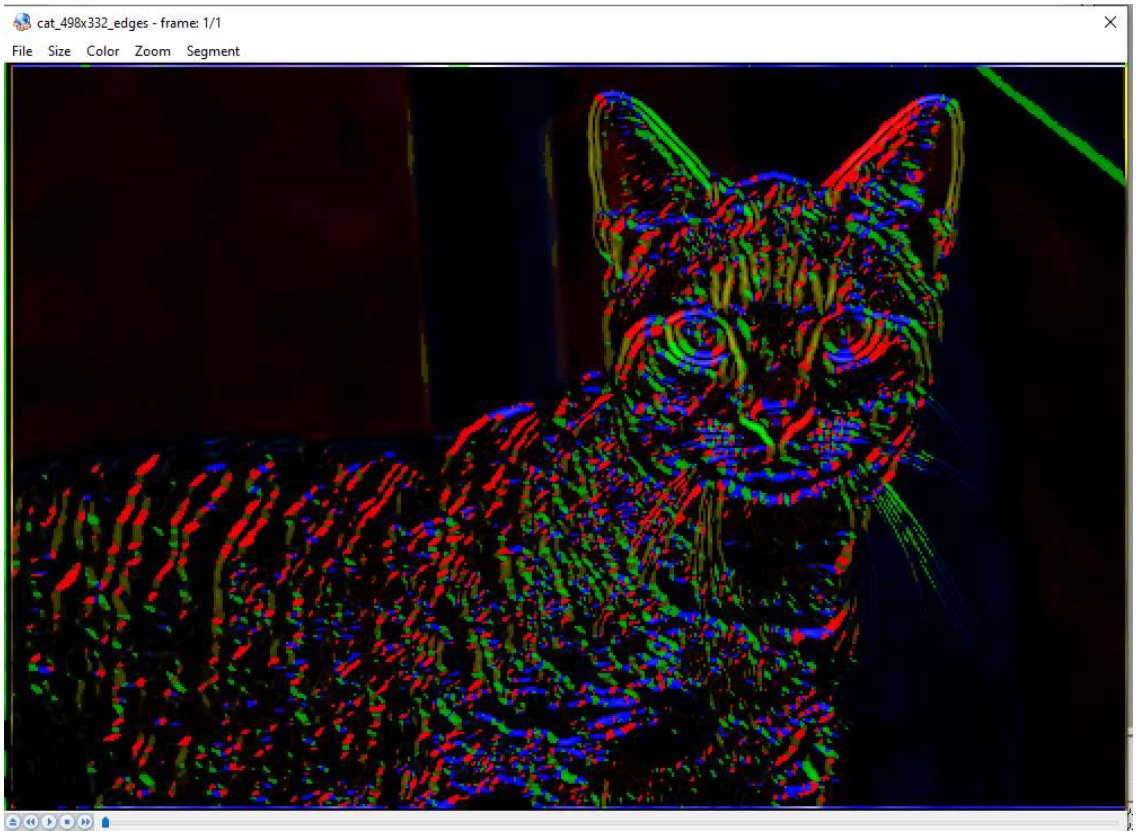
συγκρίνοντας τους κύκλους ρολογιού του αρχικού προγράμματος με αυτούς του τελικού.

Εικόνα	Διαστάσεις	Αρχικοί Κύκλοι	Τελικοί κύκλοι
cat	498x332	927.251.387	804.480.437
cherry	496x372	1.017.018.326	879.365.975

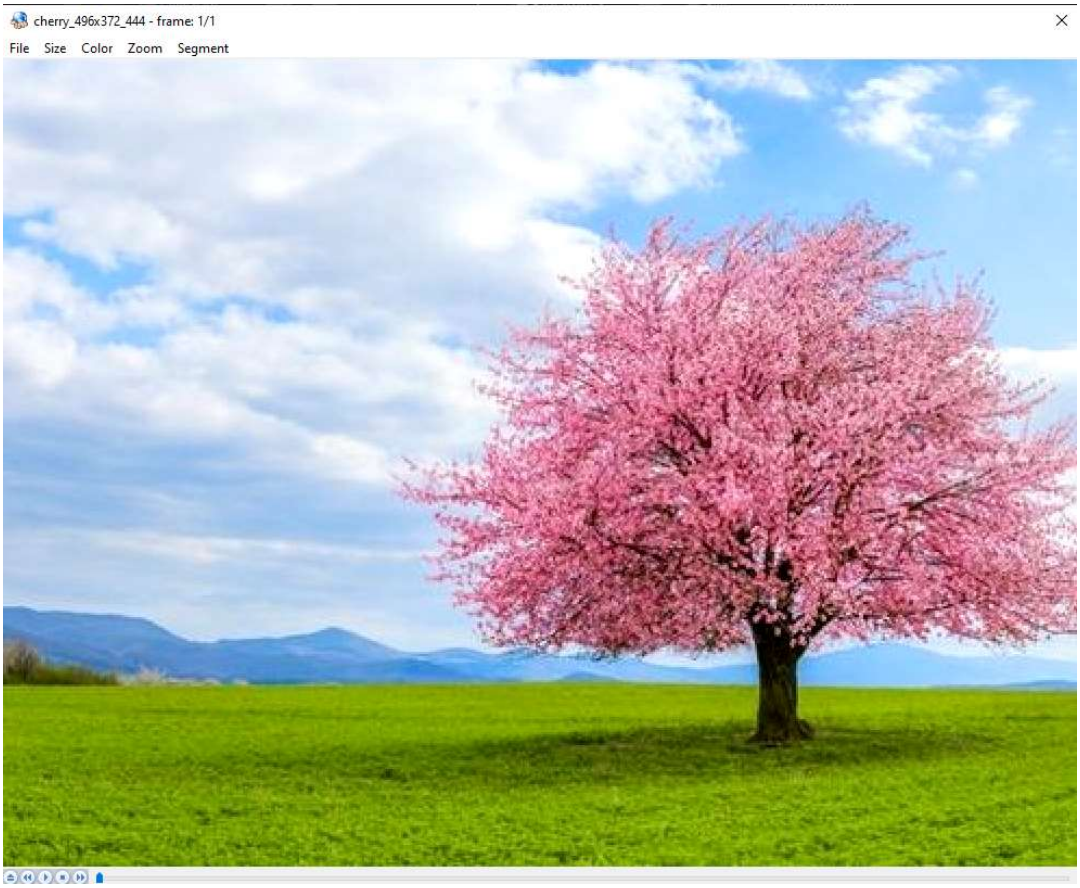
Ακόμα και με την μείωση των loop-unrolls, επιτυγχάνεται σημαντική μείωση (περίπου 13.5%) στους κύκλους ρολογιού που χρειάζονται.



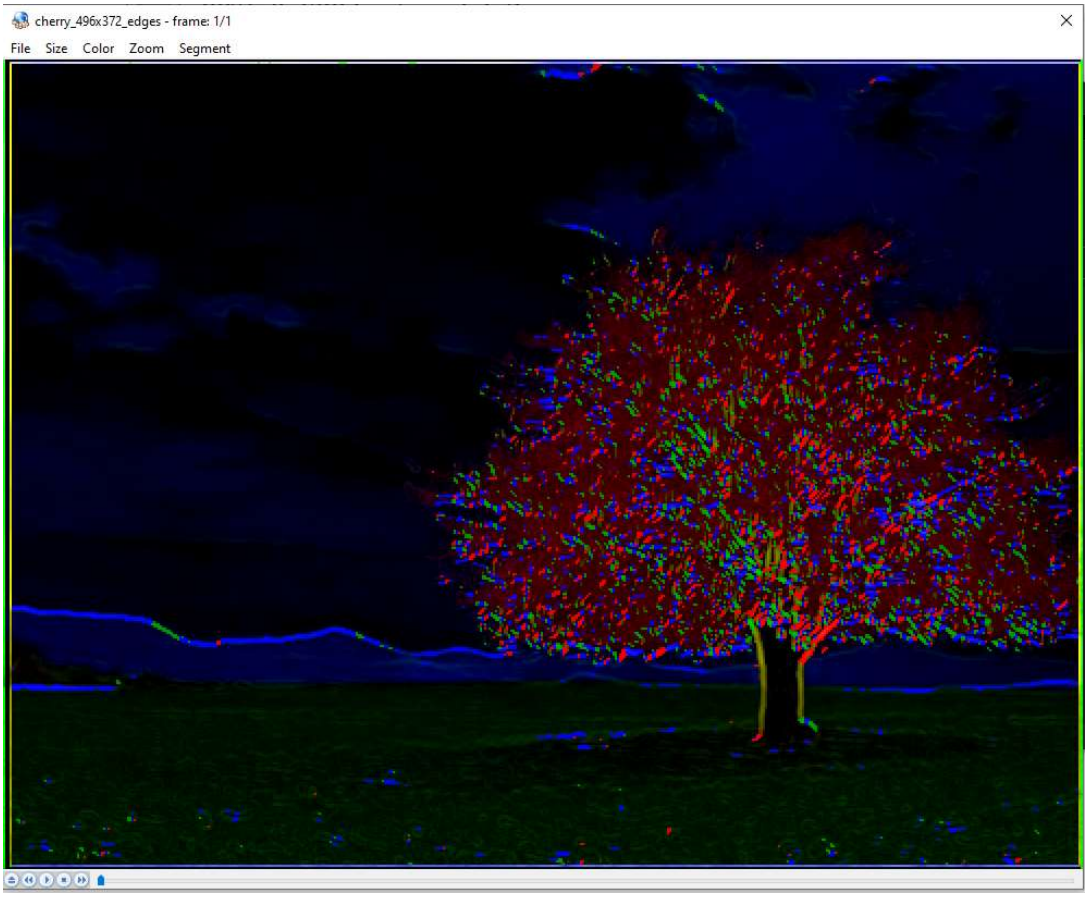
Εικόνα 45. Εικόνα cat πριν τον αλγόριθμο



Εικόνα 46. Εικόνα cat μετά τον αλγόριθμο



Εικόνα 47. Εικόνα cherry πριν τον αλγόριθμο



Εικόνα 48. Εικόνα cherry μετά τον αλγόριθμο

Πίνακας δεδομένων και προσπελάσεις

Γενικά, τα δεδομένα με τα οποία ασχολείται η εργασία είναι εικόνες, που αποτυπώνονται ως 3 πίνακες, διαστάσεων ίδιων με αυτές της αρχικής εικόνας. Αυτοί οι πίνακες, αντιστοιχούν στα 3 κανάλια με τα οποία απεικονίζεται ο χρωματικός χώρος της εικόνας, το Y(φωτεινότητα) και τα U,V(χρωμικότητα). Οι τιμές του κάθε πίνακα είναι μεταξύ 0-255 και έχουν την μορφή integer(8bit).Με βάση τα παραπάνω, το μέγεθος κάθε πίνακα προκύπτει ως εξής:

Εικόνα	Μέγεθος
Car	3*420*278*1 byte(8 bit) = 350.280 bytes
Cat	3*498*332*1 byte(8 bit) = 496.008 bytes
Cherry	3*496*372*1 byte(8 bit) = 553.536 bytes
Sunflower	3*200*200*1 byte(8 bit) = 120.000 bytes

Οι προσπελάσεις μετρούνται με την εξής λογική:
Κάθε φορά που χρησιμοποιείται ένας πίνακας, αυξάνεται ένας μετρητής κατά το πλήθος των χρήσεων που υλοποιούνται. Για παράδειγμα, κατά την συνέλιξη της αρχικής εικόνας με το φίλτρο gauss, ο μετρητής αυξάνεται κατά 9, ενώ λόγω του loop unroll, γίνεται 3 φορές προσπέλαση του πίνακα Y.
Έτσι, τα αποτελέσματα για κάθε πίνακα είναι τα ακόλουθα:

Πίνακας	Αριθμός Προσπελάσεων
current_Y	1.399.745
current_U	248.547
current_V	248.547

Όπως αναμενόταν, οι προσπελάσεις του καναλιού Y είναι πολύ περισσότερες από αυτές των υπόλοιπων καναλιών, αφού οι τιμές του καναλιού Y χρησιμοποιούνται ,κατά την δημιουργία της Γκαουσσιανής εικόνας και κατά της ανάθεσης των νέων φωτεινοτήτων του. Αντίθετα, τα κανάλια U,V προσπελάσσονται μόνο κατά τον χρωματισμό των ακμών, ενώ τέλος και τα 3 κανάλια έχουν τις ίδιες προσπελάσεις κατά την ανάγνωση και εγγραφή τους.

Σημειώσεις:

1. Εκτός και αν αναγράφεται διαφορετικά, τα screenshot τραβήχτηκαν εντοπίζοντας ακμές και ελέγχοντας τον κώδικα χρησιμοποιώντας την εικόνα car. Επίσης οι προσπελάσεις και το μέγεθος των πινάκων δεδομένων έχουν ως αναφορά την εικόνα αυτή, με τον υπολογισμό να είναι παρόμοιος και για τις υπόλοιπες εικόνες.
2. Όλες οι εικόνες που χρησιμοποιήθηκαν είχαν κωδικοποίηση 444.
3. Οι τελικές εικόνες δεν περιέχουν τα αρχικά χρώματα, καθώς οι τιμές όλων των φωτεινοτήτων αντικαθίστανται από το αντίστοιχο μέγεθος στο βήμα 4 του αλγορίθμου.
4. Οι δηλώσεις των μεταβλητών είναι global, καθώς έτσι υπαγορεύει η έκδοση του compiler της C που έχει το Codewarrior/Armulator.
5. Ενδέχεται να υπάρχουν μικροδιαφορές στους κύκλους ρολογιού των παραδοτέων αρχείων με αυτά του report. Αυτό οφείλεται σε διάφορες μικροαλλαγές που γίνονται με σκοπό την βελτίωση των αποτελεσμάτων.
6. Το όριο στην συνάρτηση scale είναι αυθαίρετο. Εμείς βρήκαμε ότι τα καλύτερα οπτικά αποτελέσματα δίνουν οι τιμές 45 για την εικόνα car, 30 για τις υπόλοιπες.