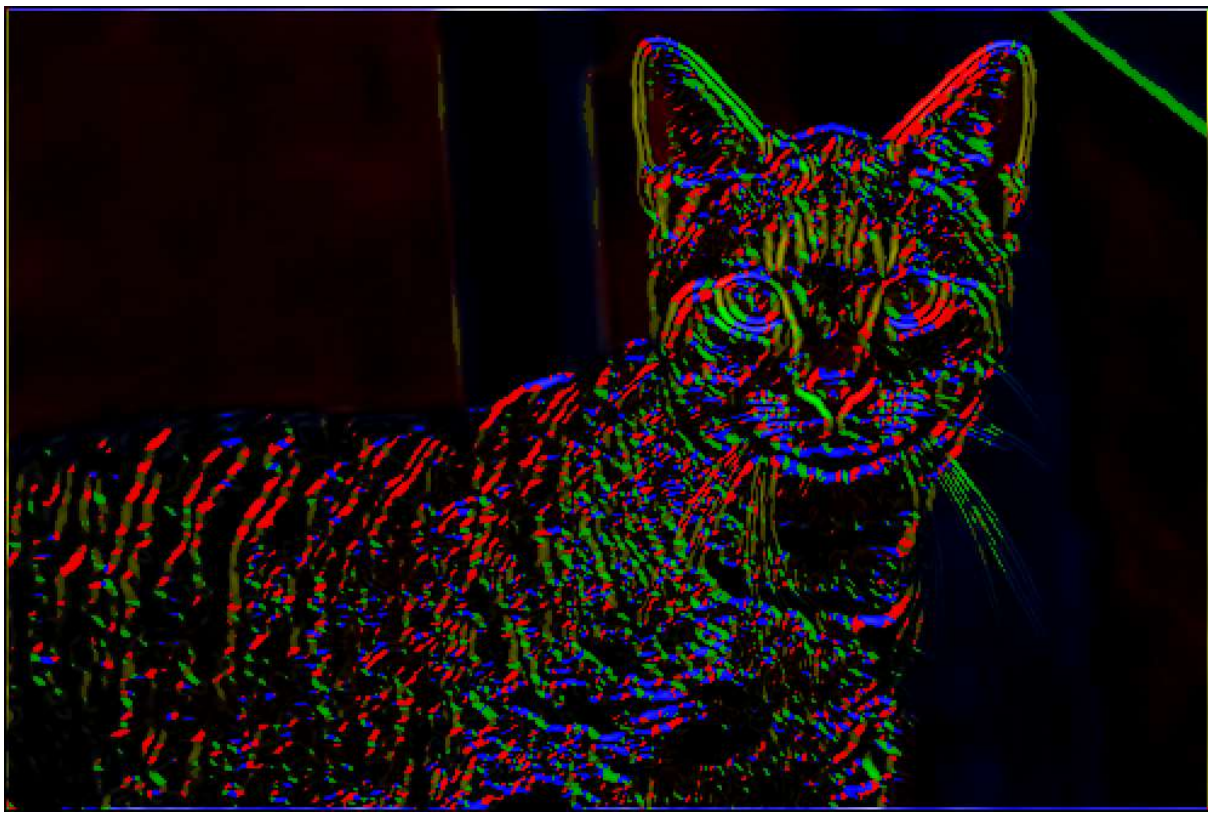
Embedded Systems Design

Project 1

Mazarakis Periklis A.M.: 57595

Papadopoulos Aristeidis A.M.:57576



Program before optimization

Initially, some parts of the code given were used. More specifically, the functions read() and write() were used, to read the images given and display them in the end, as well as the code for armulator.

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

Image 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);
}
```

Image 1. 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);
}
```

Image 2. Write() function

Afterwards, the some global variables were set up to be used in the main function. The reason for that is that in a modern compiler (such as Atom, that was used in this project), the definitions of variables can happen anywhere in the program, but in the CodeWarrior compiler this method produces an error. This is due to the fact the CodeWarrior compiler is older generation and the variable definitions must be made in the begin of each block, before another command is written. That's why global variables are used, plus any changes can happen more easily when alternating between Atom and CodeWarrior.

```
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() {
```

Image 3. Variable Initialization

In the main function, the Gaussian filter and the Sobel filters are defined in a way that no errors are caused when using CodeWarrior. Firstly, the

convolution of the grayscale image is calculated, where in the YUV color space, Y is the grayscale channel, as it maps the image's luminosity. Afterwards, Sobel filters are used to find the edges in each direction (vertical and horizontal) and in each direction the intensity of each pixel is calculated through the gradient. The angle theta is then found, needed in the color grading for each direction and degrees are converted to radians. Two values are set, maximum and minimum, with which the maximum and the minimum value of the gradient are found. Then, a rescaling is applied throughout the picture, to convert the values in the range of 0 and 255, as such:

$$new\ value = \frac{old\ value - old\ scale\ minium}{old\ scale\ maximum - old\ scale\ minimum} * 255$$

Finally, the values of the Y channel are replaced by the newly calculated ones and the coloring limits are defined, firstly by eliminating some pixels that have their luminosity higher than a limit, defined by us and for each image that limit is different. So, for pixels considered edge pixels, their angle theta is checked to find its orientation and then set its U,V channel values accordingly.

```
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];
      }
    }
  }
```

Image 4. 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];
```

Image 5. 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
    }
  }}
```

Image 6. 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;
}
```

Image 7. Rescale magnitude, and color accordingly

Optimized Program

- **Simple Optimizations**
    - Pow command replacement
    - If-statements conversion to switch-case
    - Function creation
    - Small code changes
    - Zero padding

Firstly, the pow function was replaced by its mathematical equation, i.e., multiplying the variable with itself. By doing this, a substantial amount of clock cycles were removed, since now the program does not need to access the math library to calculate the square power. Similarly, we tried to accomplish the same thing by replacing the square root calculation with two approximative mathematical types but we were not successful in this try. Also,

we tried to reduce the total clock cycles by replacing the atan function, but due to limited accuracy, we chose to use the function from the header math.h.

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

Image 8. Pow

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

Image 9. Manual power

We used as starting point the initial clock cycles of our program before optimizing and then we decided to show the clock cycles after each change we made.

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

Image 10. 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

Image 11. 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

Image 12. 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

Image 14. Manual pow and sqrt(second method)

```
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;
}
```

Image 15. Sq_rt function

Image 16. Sq_rt function results

While a lot is to be earned by replacing the square root function, the end result is not what it is required, so in the end, this idea did not move forward. As such, the only thing that was replaced was the pow command.

Then, any if statement that could be converted to switch-case statements was converted. More specifically, we found out that by placing the case that we expected it to be the most common at the top, we could reduce the cycles by a few, since less checks were made.

```
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];
```

Image 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;}
```

Image 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

Image 19. Cycles with the switch implementation

Also, we created functions for the filtering operations, for the rescaling and the coloring, having in mind to have as few lines of code as possible inside the main function.

```
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];
      }
    }
  }
```

Image 20. Initial code in the main

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

Image 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

Image 22. New cycles with clear main

Some small code changes were followed. One of those, is the change of the ending condition of some for loops, where instead of for(i=0; i<N; i++), it was changed to for(i=N; i--), because now, it is not necessary to check i, since it is set as integer, it will never drop below zero. Since less checks are being made, some clock cycles were redacted, as the checks on the end of one loop iteration are skipped. Furthermore, the angle calculation and the conversion to radians are being made in the same line. Finally, by modifying the following if-else-if statement and by shifting some variable definitions we managed to skip a few clock cycles.

```
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;}
```

Image 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
      }
```

Image 24. New if-else-if loop

Zero-padding is a special kind of optimization. Using it, what is saved is the check made in every convolution, of whether or not the pixels used are the ones on the edges of the picture. Essentially, a new matrix is created, with added columns and rows filled with zeros, so the convolution may happen as it would normally on the edges of the image. However, more loops are required, so we did try to use zero-padding, but in our implementation it did not work and it was not used in the end.

```
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];}
}}}
```

Image 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

Image 26. Zero-pad results

- **Loop optimizations**
  - Loop fission/fusion
  - Loop unrolling

- Loop interchange
- Loop collapsing
- Loop inversion
- Loop skewing
- Loop tiling

We started out with loop fission, where one loop is split into two smaller ones, in order to parallelize up to a point the original loop and achieve smaller times. This method is better for multi-core systems where one process can be parallelized into smaller ones. Loop fusion was not used, because this method requires that the two loops that are to be fused have the same limits, but in our case the images have not the same dimensions.

```
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;}
```

Image 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;}
```

Image 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

Image 29. Cycles with fission.

Afterwards, with loop unrolling, what happens is that more steps of the loop are written, with the counter incrementing less times and thus executing less branches. 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. So, the total clock cycles are reduced substantially. In our case, we used this method two times, because when we tried to use it a third time, ultimately the clock cycles were increased. Because in this project the data is in 2D, loop unrolling can only happen on the nested loop, where one of the two matrix's dimensions is looped. It is worth noting that if if-statements exist in the unrolled loops, whatever gain is to be made by loop unrolling is lost on the multiple checks that happen. Finally, applying loop unrolling in the given write-read functions, the end results is optimized even more.

```
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];
```

Image 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];
```

Image 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

Image 32. First unroll

Debugger Internals

Internal Variables | Statistics

Refere...	Instr...	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	352840206	497932002	394757473	77103632	91397523	0	563258628

Image 33. Second unroll

Then, loop interchange was tried wherever it was possible, i.e., changing the nested loop and the outer loop. But, changing the current code as this optimization requires would result in accessing the elements column wise and not row wise, as is now, resulting in slower code and slower reading times, even though for this project we consider the memory to be ideal. However, for the sake of experimentation, we applied this method and the results can be seen in the following image.

Debugger Internals

Internal Variables | Statistics

Refere...	Instru...	Core_C...	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	352845481	497943802	394762476	77108168	91399784	0	563270428

Image 34. Loop interchange

Indeed, loop interchange did not help our implementation, since the total clock cycles were increased by almost 20.000 cycles.

Furthermore, loop collapsing could not be applied to our code, since it requires accessing a matrix's elements one by one. On the contrary, in all our loops, multiple accessing is being made in elements stored in matrices, so loop collapsing could not be implemented.

Also, loop inversion was used wherever while-loops are, converting the while-loop into a mix of if-statements and do while, having in mind to avoid the waiting stages. However, in our implementation, no while-loop is used, so this method was not implemented.

One more optimization that was tried was loop skewing, with which the way of indexing is changed during the loop. However, results were disappointing and ultimately it was not used.

```
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
```

Image 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;
```

Image 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

Image 37. Loop skewing

Finally, one optimization that was not used is loop tiling. According to this method, the data are accessed as blocks, with the size of the block being one of this method's parameters. So, the block size can be made equal to that of cache's memory, creating constant cache hits and increasing the program's speed. But since for this project we consider the memory to be ideal, we do not expect to improve the clock cycles. It was implemented where it was possible but the clock cycles were worsened.

```
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;
    }}
}
```

Image 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;}
    }}
}}
```

Image 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

Image 41. Loop tilling results

- **Final program and results**

If all the optimizations are applied to our initial code, the total new cycles are as follows:

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

Image 42. Optimized code results for car image

So, in total, the reduced cycles are:

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

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

The cyclers, while a lot with a first look, considering the nature of the algorithm (several convolutions, accesses of large dimensioned matrices, condition checking etc.), the optimization made is considered successful.

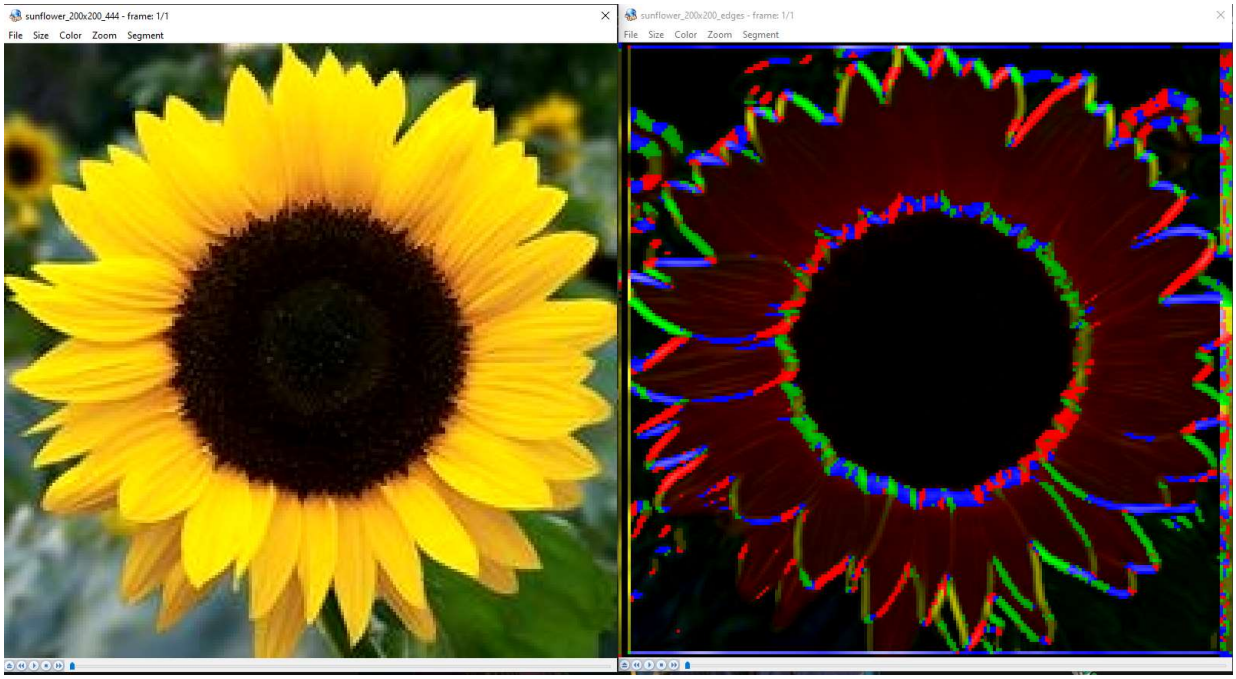


Image 43. Image sunflower before and after the algorithm

The observation can be made that the edges found on the right image, and the mistakes of this implementation, as areas on the edges of the image (where no convolution was calculated) where conceived by the algorithm as edges (the continuous lines on the outskirts of the image).

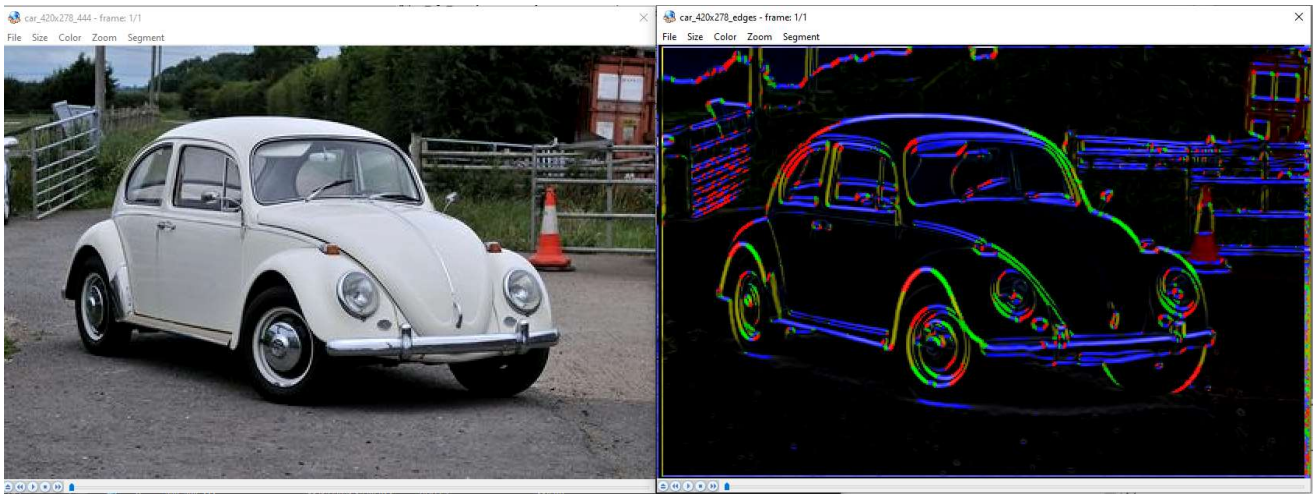


Image 44. Image car before and after the algorithm

We observed that while running the initial optimized implementation, memory violations happened for the next, larger images. The solution was to reduce the use of loop unrolling, resulting in higher clock cycles. So, in the following table, the comparison between our initial clock cycles and our optimized clock cycles can be seen.

Image	Dimensions	Initial Cycles	Final Cycles
cat	498x332	927.251.387	804.480.437
cherry	496x372	1.017.018.326	879.365.975

Even with reducing the use of loop unrolling, a substantial reduction of clock cycles is achieved (near 13.5%).

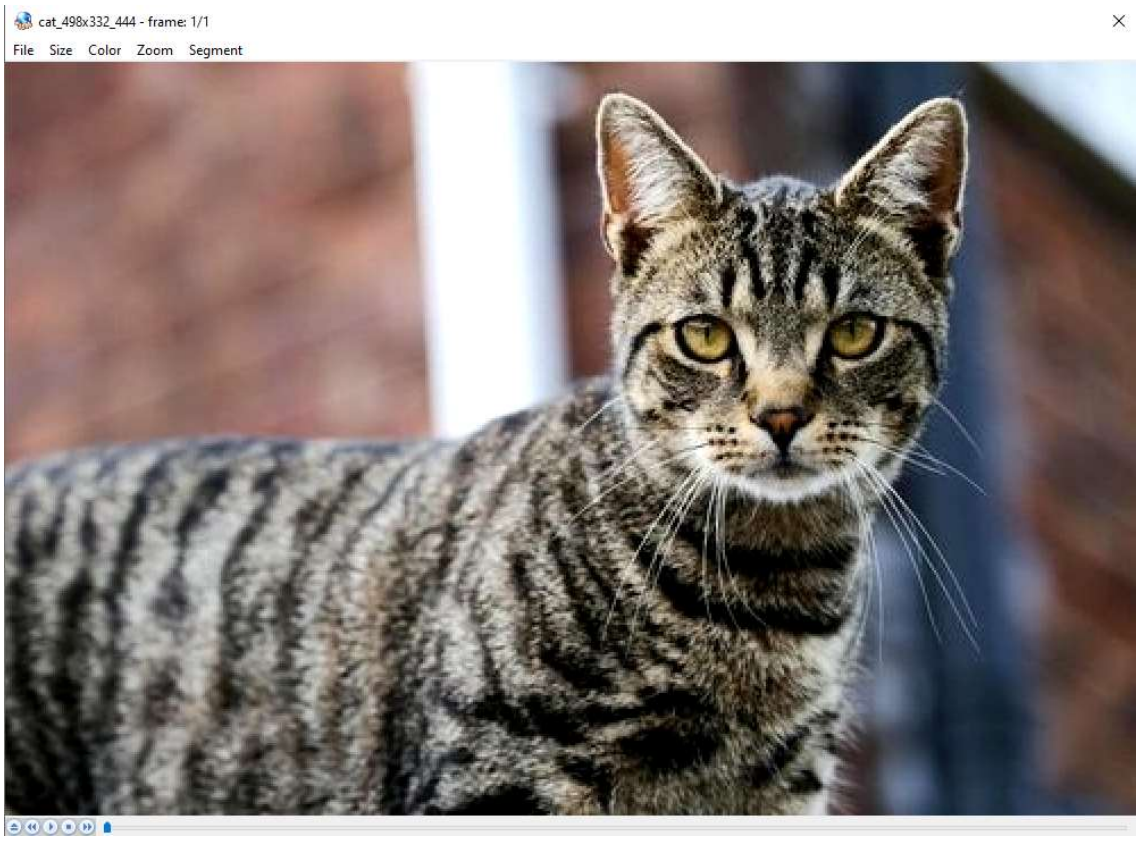


Image 45. Image cat before the algorithm

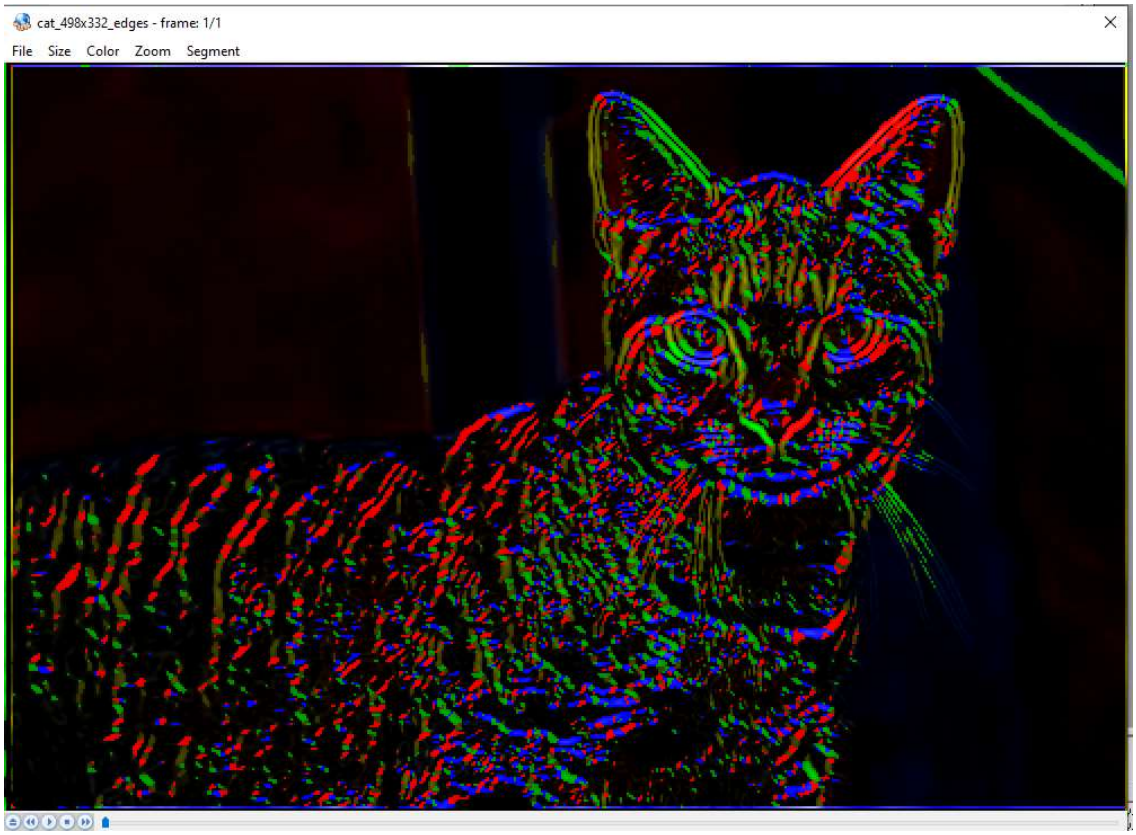


Image 46. Image cat after the algorithm

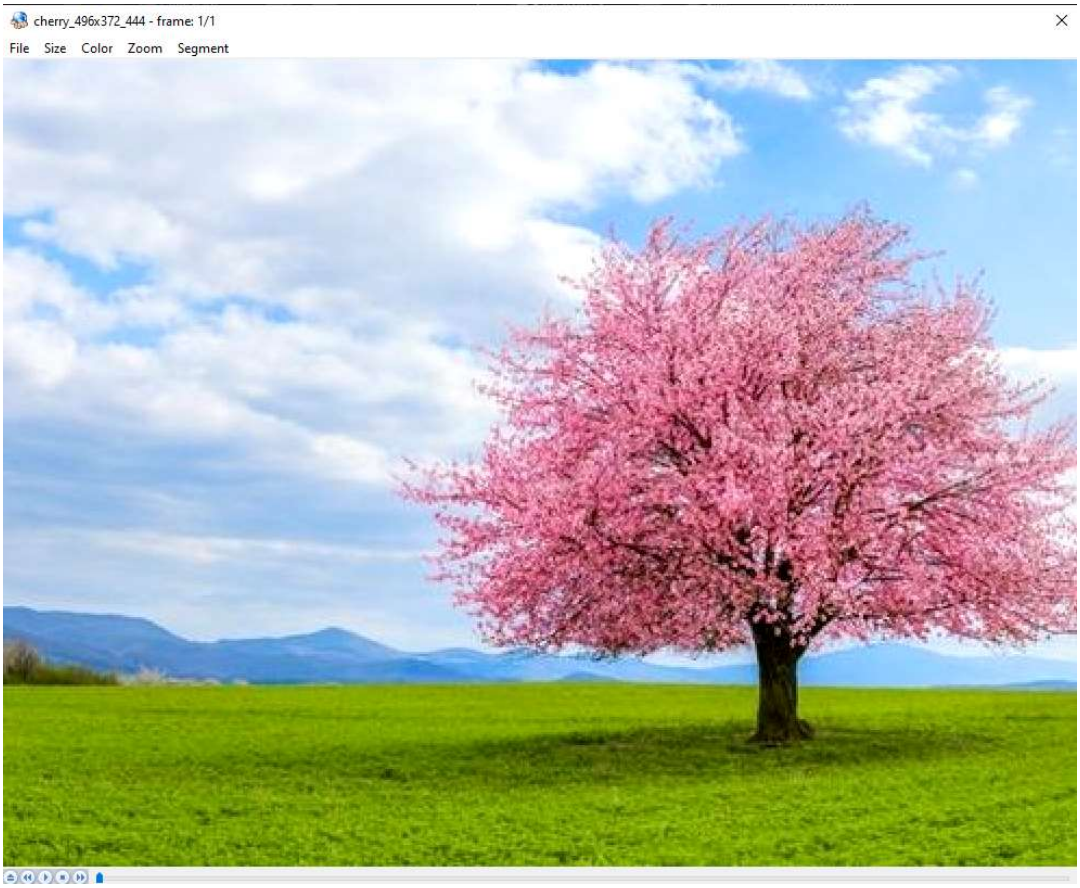


Image 47. Cherry before the algorithm

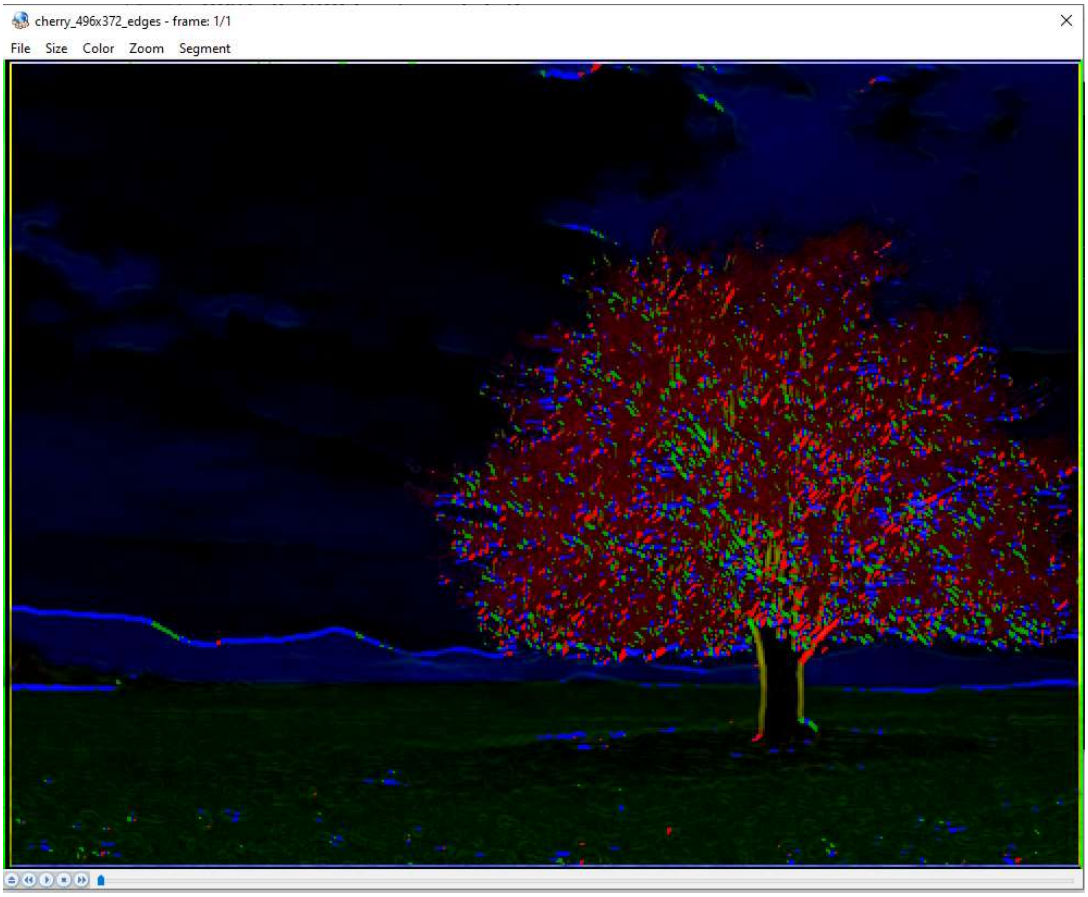


Image 48. Cherry image after the algorithm

Data matrices and memory accesses

In general, the data in this project are images, formatted as 3 matrices, with the same dimensions as the original image. Each matrix corresponds to one of the three channels of the YUV color space, with Y being luminosity and U,V being chromisity. Each matrix's values are between 0-255 with integer format (8bit). Based on those conclusions, each matrix's size is:

Image	Size
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

The memory accesses are counted with the following explanation:

Each time a matrix is used, a counter is incremented. For example, during the convolution of the initial image with the gaussian filter, the counter increases by 9, and due to loop unrolling, matrix Y is accessed 3 times. So, the results for each matrix can be seen in the next table:

Matrix	Number of accesses
current_Y	1.399.745
current_U	248.547
current_V	248.547

As expected, the memory accesses of channel Y are much more than those of the rest channels since the Y values are used almost exclusively. On the contrary, channels U,V are only accessed during the edge coloring, so those channels have the same number of memory accesses.

Σημειώσεις:

1. Unless mentioned otherwise, screenshots were taken finding the edges and checking the code using the image named car. The memory accesses and the sizes of the matrices are calculated using this image, with similar calculations for the other images.
2. All the images used have 444 encoding.
3. The final images do not contain the original colors, as all pixel values are replaced, according to step 4 of the algorithm.
4. Variable definitions are global, as implied by the C version compiler of CodeWarrior.
5. There might be some small differences between the clock cycles of the delivered files and those of this report. This is due to subtle changes made to optimize the code of this project.
6. The limit value in the scaling function is arbitrary. We found out empirically that the best visual results are obtained for setting the limit equal to 45 for the image car and 30 for the rest.