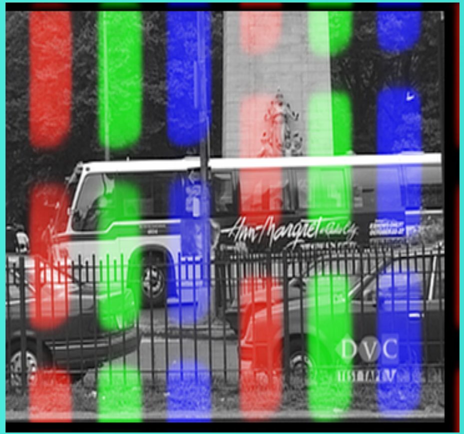
ΔΗΜΟΚΡΙΤΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΡΑΚΗΣ
DEMOCRITUS UNIVERSITY OF THRACE

Image process

Reddi,Rudim,Keshavan algorithm and Optimization

Implementation of Reddi,Rudin, Keshavan (Otsu’s extension) in image processing.



--STEP 1--
We set the thresholds to be “n” and we indicate the variables.
--STEP 2--
We find the initial thresholds
We also estimate the frequency for all the brightness levels and find the estimates ofm(ki,kj) , kj=ki+1
--STEP 3--
We calculate the error “e” and the new thresholds “ki” untill we reach {maxof}e[i]<0.5.
Last but not least, we classify the pixels taking into consideration the new thresholds. We write the photo and show the new “current_y[N][M]
Και τέλος ταξινομούμε τα pixels λαμβάνοντας υπόψην τα νέα κατώφλια.
It prints 5 different numbers which are the thresholds that we assigned.
No pixels have the brightness value of 255.

```
int main()
{
read();
printf("Arxika Katoflia\n"); //arxika katoflia
for(i=1;i<n+1;i++)
{
ki[i]=256*i/(n+1);
printf("%d\n",ki[i]);}
//ypologismos sixnotita fwteinothtas
for(f=0;f<256;f++)
{
 for(j=0;j<M;j++)
  {
        for(i=0;i<N;i++)
        {
        if(current_y[i][j]==f)
        p[f]=p[f]+1;
    }
  }
  p[f]=p[f]/(N*M);}
ki[0]=0; //prin to proto katofli
ki[n+1]=255; //meta to teleutaio katofli
for(i=0;i<n+1;i++) //exoume n+1 plithos m
{
 for(x=ki[i];x<=ki[i+1];x++)
 {
        paranom[i]=paranom[i]+p[x];
        arithm[i]=arithm[i]+x*p[x];
 }
 m[i]=arithm[i]/paranom[i];}
iter=0;
do
{
iter=iter+1;
for(i=1;i<n+1;i++)
{
e[i]=(m[i-1]+m[i])/2-ki[i];
ki[i]=ki[i]+rint(e[i]);
}
for(j=1;j<n+1;j++)
{
max=-50;
if(fabs(e[j])>max)
max=fabs(e[j]);
}
}while(max>=0.5);
printf("Finally after %d iterations:\n",iter);
```

Size of data matrix – before the optimization

Code + RO Data + ZI Data =Total RO + Total RW = 418.02kB

After Loop Unrolling and Loop Exchange we have:

Code + RO Data + ZI Data =Total RO + Total RW = 418.12kB

The number of accesses in data matrix– before the optimization

Total - Internal cycles = 623844826 – 94602482 = 529242344

After Loop Unrolling: Total - Internal cycles =623844605 – 94602455 = 529242150

After Loop Exchange: Total - Internal cycles = 540300892 – 61802804=478498088
(big change since it’s C programming)

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

printf("Arxika Katoflia\n"); //arxika katoflia
for(i=1;i<n+1;i=i+3)        //LOOP UNROLLING
{
ki[i]=256*i/(n+1);
if(i==n) {
printf("%d\n",ki[i]);
break; }
ki[i+1]=256*(i+1)/(n+1);
ki[i+2]=256*(i+2)/(n+1);
printf("%d\n%d\n%d\n",ki[i],ki[i+1],ki[i+2]);
}
```

```
 //ypologismos sixnotita fwteinothtas
for(f=0;f<256;f++)
{
 for(i=0;i<N;i++)
  {
    for(j=0;j<M;j++)
    {
    if(current_y[i][j]==f)  //LOOP EXCHANGE
    p[f]=p[f]+1;
    }
  }
  p[f]=p[f]/(N*M);
}
```

Hierarchy of data and Performance

```
load_ROM 0x0 0x08000000
{
ROM 0x0 0x80000
{
* ( +RO )
}
SRAM 0x80000 0x80000
{
 *(sram)
 stdio.o(+ZI,+RW)
 libspace.o(+ZI,+RW )
}

DRAM 0x100000 0x80000
{
* (dram)
}
}
```

```
00000000 00080000 ROM 4 R 100/100 100/100
00080000 00080000 SRAM 2 RW 1/1 1/1
00100000 00080000 DRAM 2 RW 250/50 250/50
```

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

#pragma arm section zidata="dram"
int i,j,t,f,x,r[n+1],ki[n+2],iter;
double e[n+1],p[255],m[255],paranom[n+1],arithm[n+1],max;
#pragma arm section
```

Memory ROM 524kB, 4x8=32bit,only for RO data
Memory SRAM 524kB, 2x8=16bit for library and sram data

Memory DRAM 524kB, 2x8=16bit for dram data

SRAM usually is smaller than DRAM and is closest to the processor(cache memory).
It’s more expensive while being faster.

It is shown that there is change only in the waiting states part. Waiting state is a delay of a processor when he wants to gain access in a memory with slowest responsiveness than him.We sent more data in a cycle so the waiting states are decreasing.

ebugger Internals

nternal Variables | Statistics

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Wait_States	Total	True_Idle_Cycles
statistics	324841960	613826626	359427515	190941023	93051076	0	2463287858	3106707472	3727908

Then we change write and read times to 150/25 150/25 in DRAM and waiting states decrease.

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Wait_States	Total	True_Idle_Cycle:
$statistics	324841960	613826626	359427515	190941023	93051076	0	2183755079	2827174693	3727908

Cycles changes before and after

ebugger Internals

nternal Variables | Statistics

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Wait_States	Total	True_Idle_Cycles
statistics	324841960	613826626	359427515	190941023	93051076	0	2015953664	2659373278	3727908

Debugger Internals

Internal Variables | Statistics

Reference Po...	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Wait_States	Total	True_Idle_Cy...
$statistics	60235228	112906384	67673256	33910132	12191446	0	333345952	447120786	903468

Instruction changes before and after

```
Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

      1952         60          0     414052       8032   Object Totals
     15896        314          0        300       6624   Library Totals
==============================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     17848        374          0     414352      14656   Grand Totals
==============================================================================

    Total RO  Size(Code + RO Data)                18222 (  17.79kB)
    Total RW  Size(RW Data + ZI Data)            414352 ( 404.64kB)
    Total ROM Size(Code + RO Data + RW Data)      18222 (  17.79kB)
==============================================================================
```

```
Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

      1656         60          0     409756       6320   Object Totals
     15964        314          0        300       6700   Library Totals
==============================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     17620        374          0     410056      13020   Grand Totals
==============================================================================

    Total RO  Size(Code + RO Data)                17994 (  17.57kB)
    Total RW  Size(RW Data + ZI Data)            410056 ( 400.45kB)
    Total ROM Size(Code + RO Data + RW Data)      17994 (  17.57kB)
==============================================================================
```

```
/* code for armulator*/
#pragma arm section zidata="sram"
int current_y[N][M],current_line[M][B],block[B][B];
#pragma arm section
```

```
//CURRENT_LINE 2o epipedo
for(x=0;x<N/B;x++)
{                         /* For all blocks in the current frame */
 for(j=0;j<M;j++)
 {                           /* Copy data from current to buffer current_line */
 for(i=0;i<B;i++) {
 current_line[i][j]=current_y[B*x+i][j];
                } }


//BLOCK 3o epipedo

for(y=0;y<M/B;y++)
{

for(k=0;k<B;k++) /* Copy data from current to buffer block */
{
 for(l=0;l<B;l++)
 {
 block[k][l]=current_line[k][B*y+l];
 }
}}

 //ypologismos sixnotita fwteinothtas
 for(f=0;f<256;f++)
 {
 for(k=0;k<B;k++)
{
 for(l=0;l<B;l++)
 {
 if(block[k][l]==f)
    p[f]=p[f]+1;
    }
   }
    p[f]=p[f]/(N*M);
    }
}
```

We we reuse the data we try to decrease the access times in the data matrix , more importantly the data of current_y.
We accomplish that by making smaller matrixes where we insert the temporarily items that we need at the specific time.

Obviously we see a rise of instructions since we added lines of code for the temporarly keeping of the data. We notice degration of cycles comparing the previous stepw where all the data were included in just one big matrix.