



Ciencia y Tecnología

Secretaría de Ciencia, Humanidades, Tecnología e Innovación



CIMAT
UNIDAD MÉRIDA

PROCESAMIENTO DE IMÁGENES USANDO CUDA

Dr. Francisco J. Hernández López
SECIHTI – CIMAT-Mérida
fcoj23@cimat.mx, www.cimat.mx/~fcoj23



LEER IMAGEN USANDO OPENCV

```
13  #include <opencv2/opencv.hpp>
14
15  int main(void){
16      cv::Mat Image;
17
18      //desde el disco duro
19      Image = cv::imread("../images/flowers8.png");
20
21      int nCols=Image.cols;
22      int nFils=Image.rows;
23
24      printf("\nLeyendo imagen de %d x %d pixeles...\n", nFils, nCols);
25
26      return(0);
27  }
```

Para compilar:

```
g++ `pkg-config --cflags opencv` LeerImagen.cpp -o LeerImagen `pkg-config --libs opencv`
```

```
g++ `pkg-config --cflags opencv4` LeerImagen.cpp -o LeerImagen `pkg-config --libs opencv4`
```

MODIFICAR IMAGEN A COLOR

```
4  /*CUDA*/
5  #include "cuda_runtime.h"
6
7  /*OpenCV*/
8  #include <opencv2/highgui/highgui.hpp>
9
10 #include "ModificarImagen.h"
11
12 int main(int argc, char **argv){
13     cudaSetDevice(0);
14     /*Host Variables*/
15     int imageW, imageH;      //Tamaño de la imagen
16     cv::Mat frame_Original;  //Imagen original
17     cv::Mat frame_Modified;  //Imagen modificada
18     /*Device Variables*/
19     uchar3 *Image_dev;
20
21     //cv::namedWindow("Original Frame",CV_WINDOW_AUTOSIZE);
22     //cv::namedWindow("Modified Frame",CV_WINDOW_AUTOSIZE);
23
24     /*Load Image*/
25     frame_Original = cv::imread("../images/Estacion_MR.jpg",1);
26
27     //cv::imshow("Original Frame", frame_Original);
```



MODIFICAR IMAGEN A COLOR (C1)

```
29      /*Size of Image*/
30      imageW=frame_Original.cols;
31      imageH=frame_Original.rows;
32      size_t size=3*imageW*imageH;
33
34      //Create Host memory
35      frame_Modified.create(imageH, imageW,CV_8UC(3));
36
37      /*Create device memory*/
38      cudaMalloc((void **)&Image_dev,size);
39
40      /*Copy Memory (Host-->Device)*/
41      cudaMemcpy(Image_dev,frame_Original.data,size,cudaMemcpyHostToDevice);
42
43      /*Define the size of the grid and thread blocks*/
44      dim3 threads(512,1,1);
45      int N=imageW * imageH;
46      dim3 grid(N/threads.x + (N%threads.x == 0 ? 0:1),1,1);
47      /*Launch the Kernel Function*/
48      CUDA_Modificar_Imagen(Image_dev,N,grid,threads);
```

MODIFICAR IMAGEN A COLOR (C2)

```
50      /*Copy Memory (Device-->Host)*/
51      cudaMemcpy(frame_Modified.data, Image_dev, size, cudaMemcpyDeviceToHost);

53      //cv::imshow("Modified Frame", frame_Modified);
54      //cv::waitKey(0);
55      //Guardar el resultado
56      cv::imwrite("../images/Estacion_MR_Modificada.jpg", frame_Modified);

58      /*Clean Memory*/
59      /*Host*/
60      //cv::destroyWindow("Original Frame");
61      //cv::destroyWindow("Modified Frame");
62      frame_Modified.release();
63
64      /*Device*/
65      cudaFree(Image_dev);
66
67      return(0);
68  }
```



MODIFICAR IMAGEN A COLOR (C3)

```
ModificarImagen.h*  ModificarImagen.cpp  ModificarImagen.cu
Miscellaneous Files  (Global Scope)

1
2 void CUDA_Modificar_Imagen(uchar3 *Image_dev,int N,dim3 grid,dim3 threads);

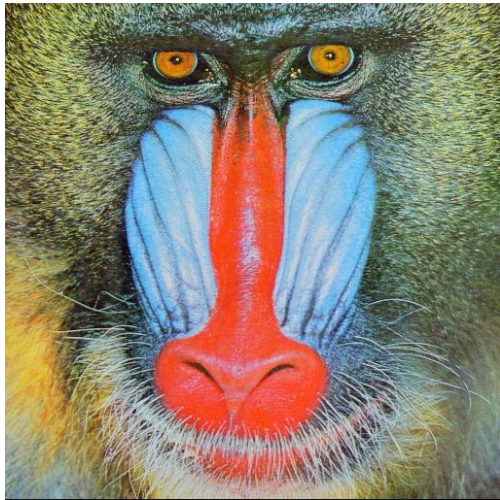
ModificarImagen.h*  ModificarImagen.cpp  ModificarImagen.cu*  ModificarImagen.cu
Miscellaneous Files  (Global Scope)

1
2 #include "ModificarImagen.h"
3 // Kernel Function which is executed in the Device.
4 __global__ void Modificar_Imagen(uchar3 *Imag,int N)
5 {
6     int idx = blockIdx.x * blockDim.x + threadIdx.x;
7
8     if (idx<N){
9         if((Imag[idx].x+Imag[idx].y+Imag[idx].z)/3 > 250){
10             Imag[idx].x=255; //B
11             Imag[idx].y=0;    //G
12             Imag[idx].z=0;    //R
13         }
14     }
15 }
16 void CUDA_Modificar_Imagen(uchar3 *Image_dev,int N,dim3 grid,dim3 threads){
17     Modificar_Imagen<<<grid,threads>>> (Image_dev,N);
18 }
```

Para compilar:

```
nvcc `pkg-config --cflags opencv4` ModificarImagen.cu ModificarImagen.cpp -o ModificarImagen
`pkg-config --libs opencv4`
```

COMPOSICIÓN DE IMÁGENES



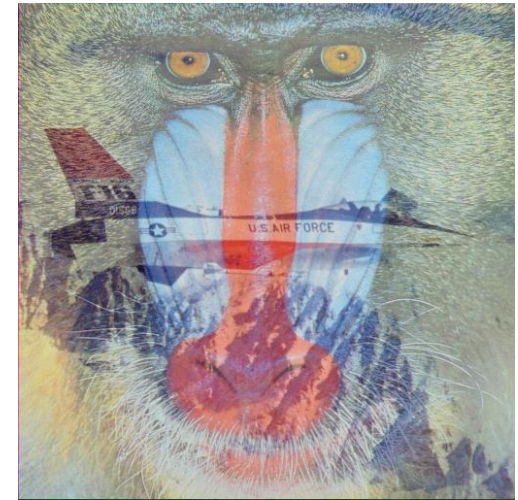
αI_1

+



$(1 - \alpha) I_2$

=



O con $\alpha = 0.5$

COMPOSICIÓN DE IMÁGENES (C1)

```
3 | #include <stdio.h>
4 | /*CUDA*/
5 | #include <cuda_runtime.h>
6 | /*OpenCV*/
7 | #include <opencv2/highgui/highgui.hpp>
8 | #include "ImageComposition.h"
9 |
10 | //alpha parameter
11 | int alpha_int=75;
12 | float alpha= (float)alpha_int / 100; //[0,1]
13 | void Change_alpha(int,void *){
14 |     alpha=(float)alpha_int/100;
15 | }
16 |
17 | int main(int argc, char **argv){
18 |     cudaSetDevice(0);
19 |     /*Host Variables*/
20 |     int imageW, imageH;           //Size of Image
21 |     cv::Mat frame1_Original;      //Original Image
22 |     cv::Mat frame2_Original;      //Original Image
23 |     cv::Mat frame_Composed;
```

COMPOSICIÓN DE IMÁGENES (C2)

```
25  /*Device Variables*/
26  uchar3 *Image1_dev,*Image2_dev,*ComposedImage_dev;
27
28  /*cv::namedWindow("Original Frame1",CV_WINDOW_AUTOSIZE);
29  cv::namedWindow("Original Frame2",CV_WINDOW_AUTOSIZE);
30  cv::namedWindow("Composed Frame",CV_WINDOW_AUTOSIZE);
31  cv::createTrackbar("alpha", "Composed Frame",&alpha_int,100,Change_alpha,0);*/
32
33  char name_imag[500];
34  /*Load Image*/
35  sprintf(name_imag,"../images/baboon.jpg");
36  frame1_Original = cv::imread(name_imag,1);
37
38  sprintf(name_imag,"../images/airplane.jpg");
39  frame2_Original = cv::imread(name_imag,1);
40
41  /*Size of Image*/
42  imageW=frame1_Original.cols;
43  imageH=frame1_Original.rows;
44  size_t size=imageW*imageH*sizeof(uchar3);
```

COMPOSICIÓN DE IMÁGENES (C3)

```
46      /*Create host memory*/
47      frame_Composed.create(imageH,imageW,CV_8UC(3));
48
49      /*Create device memory*/
50      cudaMalloc((void **)&Image1_dev,size);
51      cudaMalloc((void **)&Image2_dev,size);
52      cudaMalloc((void **)&ComposedImage_dev,size);
53
54      /*Copy Memory (Host-->Device)*/
55      cudaMemcpy(Image1_dev,frame1_Original.data,size,/*Complete*/);
56      cudaMemcpy(Image2_dev,frame2_Original.data,size,/*Complete*/);
57
58      /*Define the size of the grid and thread blocks*/
59      dim3 threads(/*Complete*/,1,1);
60      int N=imageW * imageH;
61      dim3 grid(N/threads.x + (N%threads.x == 0 ? 0:1),1,1);
62
63      /*cv::imshow("Original Frame1",frame1_Original);
64      cv::imshow("Original Frame2",frame2_Original);*/
```

COMPOSICIÓN DE IMÁGENES (C4)

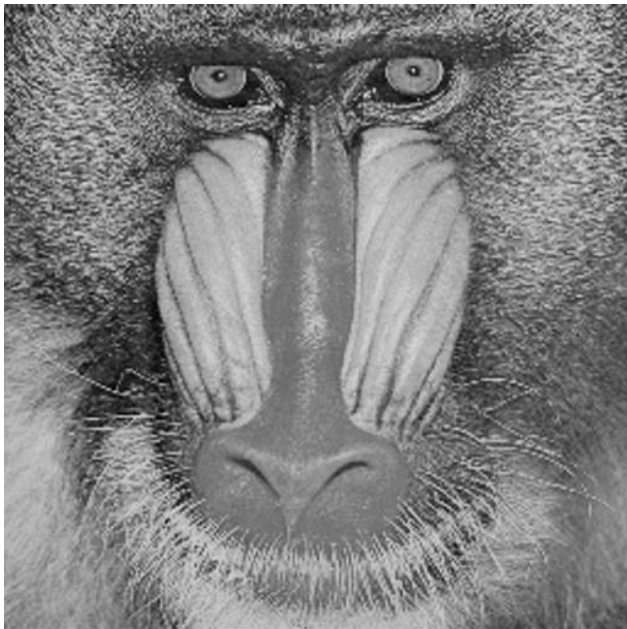
```
66 //while(1){
67     /*Launch the Kernel Function*/
68     CUDA_Compose_Images(ComposedImage_dev,Image1_dev,Image2_dev,alpha,N,grid,threads);
69
70     /*Copy Memory (Device-->Host)*/
71     cudaMemcpy(frame_Composed.data,ComposedImage_dev,size,/*Complete*/);
72
73     //cv::imshow("Composed Frame",frame_Composed);
74
75     /* char key=cvWaitKey(2);
76        if(key==27) {
77            break;
78        }
79    */
81     //Save the result
82     cv::imwrite("../results/Compose_babon_airplane.jpg", frame_Composed);
83
84     /*Clean Memory*/
85     /*Host*/
86     /*cv::destroyAllWindows();*/
87     frame_Composed.release();
88     /*Device*/
89     cudaFree(Image1_dev);
90     cudaFree(Image2_dev);
91     cudaFree(ComposedImage_dev);
92
93     return(0);
94 }
```

COMPOSICIÓN DE IMÁGENES (C5)

```
ImageComposition.h* X ImageComposition.cpp ImageComposition.cu
Miscellaneous Files (Global Scope)
1
2 void CUDA_Compose_Images(uchar3 *ComposedImage_dev,uchar3 *Image1_dev,uchar3 *Image2_dev,
3 float alpha,int N,dim3 grid,dim3 threads);
```

```
ImageComposition.h ImageComposition.cpp ImageComposition.cu X
1 #include "ImageComposition.h"
2
3 __global__ void Compose_Images(uchar3 *ComposedImage_dev,uchar3 *Image1_dev,uchar3 *Image2_dev,
4 float alpha,int N){
5     int idx = blockIdx.x * blockDim.x + threadIdx.x;
6     if (/*Complete*/){
7         ComposedImage_dev[idx].x=alpha/*Complete*/+(1-alpha)/*Complete*/;
8         ComposedImage_dev[idx].y=alpha/*Complete*/+(1-alpha)/*Complete*/;
9         ComposedImage_dev[idx].z=alpha/*Complete*/+(1-alpha)/*Complete*/;
10    }
11 }
12
13 void CUDA_Compose_Images(uchar3 *ComposedImage_dev,uchar3 *Image1_dev,uchar3 *Image2_dev,
14 float alpha,int N,dim3 grid,dim3 threads){
15     Compose_Images<<</*Complete*/>>>(ComposedImage_dev,Image1_dev,Image2_dev,alpha,N);
16 }
```

APLICAR FILTROS A UNA IMAGEN



I_Original

$$\otimes \frac{1}{49} \begin{pmatrix} 1 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 1 \end{pmatrix}_{7 \times 7} =$$

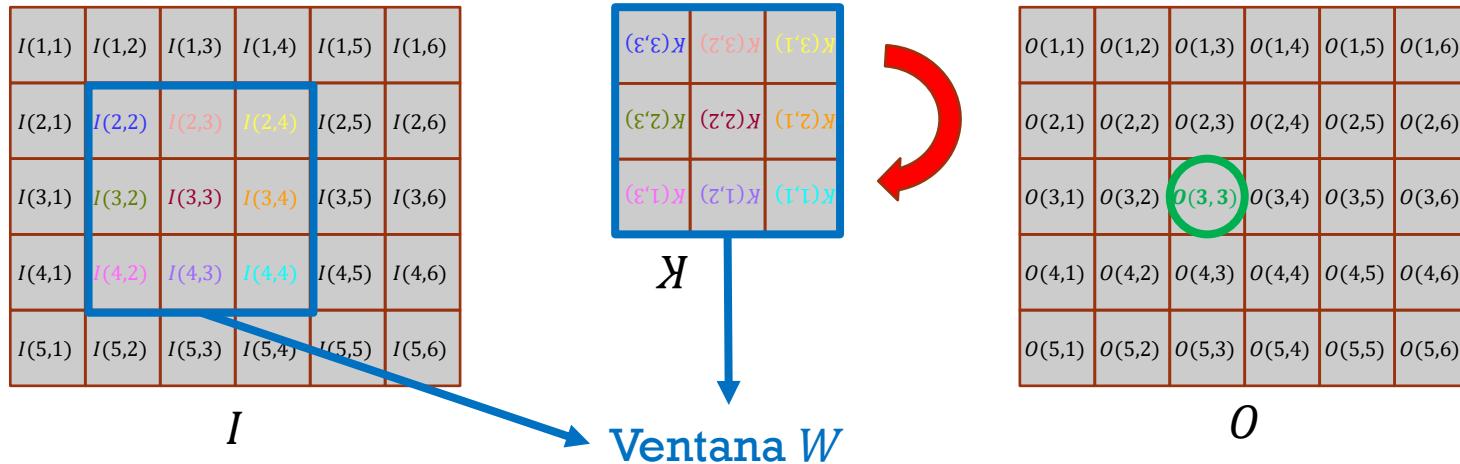
Kernel de convolución
o correlación



I_Filtrada

CONVOLUCIÓN

correlación con kernel girado



$$O(3,3) = I(4,4) * K(1,1) + I(4,3) * K(1,2) + I(4,2) * K(1,3) + \\ I(3,4) * K(2,1) + I(3,3) * K(2,2) + I(3,2) * K(2,3) + \\ I(2,4) * K(3,1) + I(2,3) * K(3,2) + I(2,2) * K(3,3)$$

filter2D

Convolve an image with the kernel.

C++: `void filter2D(InputArray src, OutputArray dst, int ddepth, InputArray kernel, Point anchor=Point(-1,-1), double delta=0, int borderType=BORDER_DEFAULT)`

OpenCV: https://docs.opencv.org/3.4.1/d4/d86/group_imgproc_filter.html

FILTROS: PROMEDIO, GAUSSIANO Y LAPLACIANO

```
3  #include <stdio.h>
4  /*CUDA*/
5  #include <cuda_runtime.h>
6  /*OpenCV*/
7  #include <opencv2/opencv.hpp>
8
9  #include "ImageFilters.h"
10
11 //División techo.
12 int iDivUp(int a, int b){
13     return ((a % b) != 0) ?
14         (a / b + 1) : (a / b);
15 }
16
17 //PARAMETERS
18 #define BLOCKDIM_X 16
19 #define BLOCKDIM_Y 16
20
21 int main(int argc, char **argv){
22     /*Host Variables*/
23     int imageW, imageH;    //Size of Image
24     cv::Mat I_Original;    //Original Image
25     cv::Mat I_Filtrada;    //Para guardar el resultado
26     cv::Mat I_Filtrada_C1F; //Imagen de 1 Canal en Float
```

$$K_{Promedio} = \frac{1}{tW \times tW} \begin{pmatrix} 1 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 1 \end{pmatrix}_{tW \times tW}$$

$$K_{Gaussiano} = \frac{1}{16} \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

$$K_{Laplaciano} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

FILTROS: PROMEDIO, GAUSSIANO Y LAPLACIANO (C1)

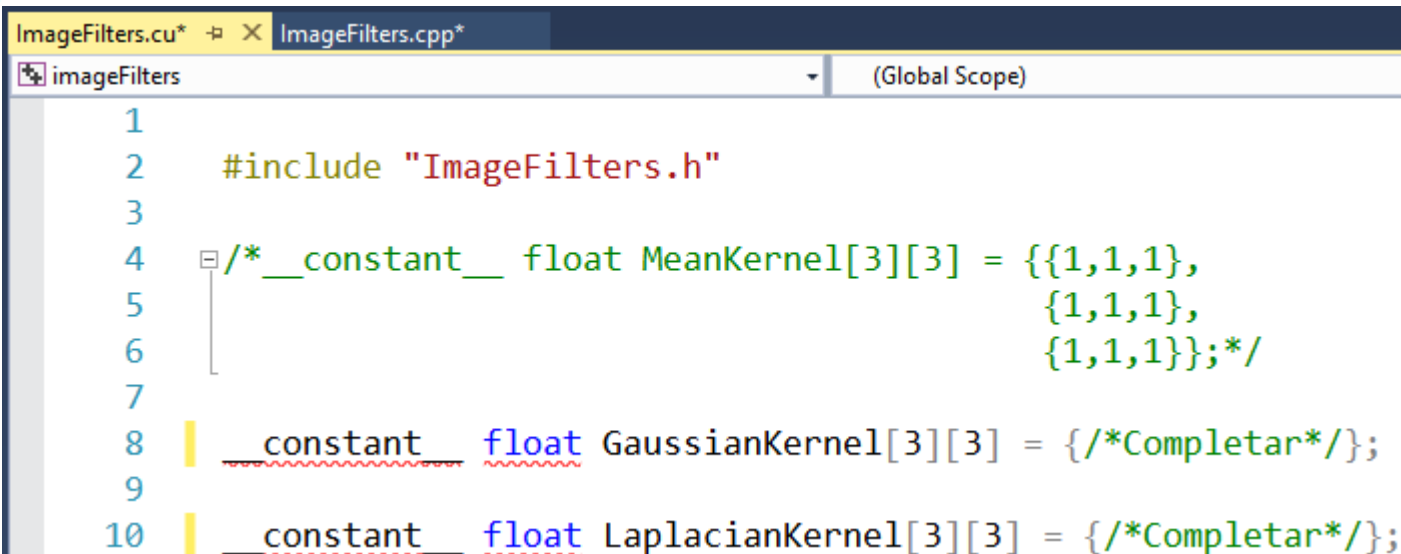
```
27     /*Device Variables*/
28     unsigned char *I_Original_dev;
29     unsigned char *I_Filtrada_dev;
30     float *I_Filtrada_C1F_dev;
31
32     char name_imag[500];
33     /*Load Image*/
34     sprintf(name_imag, /*Completar*/);
35     I_Original = cv::imread(name_imag,0);
36
37     /*Size of Image*/
38     imageW=I_Original.cols;
39     imageH=I_Original.rows;
40     size_t size=imageW*imageH*sizeof(unsigned char);
41     size_t sizef=imageW*imageH*sizeof(float);
42
43     I_Filtrada.create(imageH, imageW, CV_8UC(1));
44     I_Filtrada_C1F.create(imageH, imageW, CV_32FC(1));
45
46     /*Create device memory*/
47     cudaMalloc((void **)&I_Original_dev, size);
48     cudaMalloc((void **)&I_Filtrada_dev,size);
49     cudaMalloc((void **)&I_Filtrada_C1F_dev,sizef);
50     cudaMemset(I_Filtrada_dev, 0, size); //Inicializamos en cero
51     cudaMemset(I_Filtrada_C1F_dev, 0, sizef);
```

FILTROS: PROMEDIO, GAUSSIANO Y LAPLACIANO (C2)

```
54  /*Copy Memory (Host-->Device)*/
55  cudaMemcpy(I_Original_dev,I_Original.data,size,/*Completar*/);
56
57  /*Define the size of the grid and thread blocks*/
58  dim3 threads(/*Completar*/);
59  dim3 grid(iDivUp(imageW, /*Completar*/), iDivUp(imageH, /*Completar*/),1);
61  /*Launch the Kernel Function*/
62  //Mean Filter
63  CUDA_MeanFilter(I_Filtrada_dev, I_Original_dev,imageW,imageH,grid,threads);
64  //CUDA_GaussianFilter(I_Filtrada_dev, I_Original_dev,imageW,imageH,grid,threads);
65  //CUDA_LaplacianFilter(I_Filtrada_C1F_dev,I_Original_dev,imageW,imageH,grid,threads);
67  /*Copy Memory (Device-->Host)*/
68  cudaMemcpy(I_Filtrada.data, I_Filtrada_dev, size, /*Completar*/);
69  //cudaMemcpy(I_Filtrada_C1F.data, I_Filtrada_C1F_dev,sizef,/*Completar*/);
71  cv::imshow("Imagen Original", I_Original);
72  cv::imshow("Imagen Filtrada", I_Filtrada);
73  /*cv::normalize(I_Filtrada_C1F, I_Filtrada_C1F, 1, 0, CV_MINMAX);
74  cv::imshow("Imagen Filtrada", I_Filtrada_C1F);*/
75  cv::waitKey(0);
85  cv::imwrite("../.../images/Baboon_512_MeanFilter.png",I_Filtrada);
86  //cv::imwrite("../.../images/Baboon_512_GaussianFilter.png", I_Filtrada);
87  /*cv::normalize(I_Filtrada_C1F, I_Filtrada_C1F, 255, 0, cv::NORM_MINMAX);
88  cv::imwrite("../.../images/Baboon_512_LaplacianFilter.png", I_Filtrada_C1F);*/
```

FILTROS: PROMEDIO, GAUSSIANO Y LAPLACIANO (C3)

```
82  /*Clean Memory*/
83  /*Host*/
84  I_Filtrada.release();
85  I_Filtrada_C1F.release();
86  //cv::destroyAllWindows();
87  /*Device*/
88  cudaFree(I_Original_dev);
89  cudaFree(I_Filtrada_dev);
90  cudaFree(I_Filtrada_C1F_dev);
91
92  return(0);
93 }
```



The screenshot shows a code editor with two tabs: 'ImageFilters.cu*' and 'ImageFilters.cpp*'. The 'ImageFilters.cpp*' tab is active, showing the implementation of the 'imageFilters' function in the 'Global Scope'. The code includes the header 'ImageFilters.h' and defines three 3x3 kernels: 'MeanKernel' (a 1D kernel with all ones), 'GaussianKernel' (commented as 'Completar'), and 'LaplacianKernel' (commented as 'Completar').

```
1
2  #include "ImageFilters.h"
3
4  /*__constant__ float MeanKernel[3][3] = {{1,1,1},
5                                           {1,1,1},
6                                           {1,1,1}};*/
7
8  constant float GaussianKernel[3][3] = {/*Completar*/};
9
10 constant float LaplacianKernel[3][3] = {/*Completar*/};
```

FILTRO PROMEDIO

```
12  /*Image Filters*/
13  global void MeanFilter_kernel(unsigned char *Dst_dev, unsigned char *Src_dev,
14                                     int imageW,int imageH){
15
16      const int ix = blockDim.x * blockIdx.x + threadIdx.x;
17      const int iy = blockDim.y * blockIdx.y + threadIdx.y;
18
19      int rW = 3;//radio de la ventana
20      int tW = rW * 2+1;
21      if(ix>=rW && ix < imageW-rW && iy>=rW && iy < imageH-rW){//Dejamos un margen de tamaño rW
22          float sum=0;
23          int idx;
24          for(int k_i=0;k_i<tW;k_i++){//Para recorrer el Kernel en cada pixel (iy,ix)
25              idx = (iy + (k_i - rW))*imageW + ix;
26              for(int k_j=0;k_j<tW;k_j++){
27                  sum+=(float)Src_dev[idx + (k_j - rW)];
28                  /*sum+=(float)Src_dev[idx + (k_j - rW)]*MeanKernel[k_i][k_j];*/
29              }
30          }
31          idx = iy*imageW + ix;
32          Dst_dev[idx]=(unsigned char)(sum/(tW*tW));
33      }
34  }
```

FILTRO GAUSSIANO

```
33 __global__ void GaussianFilter_kernel(unsigned char *Dst_dev, unsigned char *Src_dev,
34                                     int imageW, int imageH) {
35
36     const int ix = blockDim.x * blockIdx.x + threadIdx.x;
37     const int iy = blockDim.y * blockIdx.y + threadIdx.y;
38
39     if (ix > 0 && ix < imageW - 1 && iy > 0 && iy < imageH - 1) { //Dejamos un margen de tamaño 1
40         float sum = 0;
41         int idx;
42         for (int k_i = 0; k_i < 3; k_i++) { //Para recorrer el Kernel en cada pixel (iy,ix)
43             idx = (iy + (k_i - 1)) * imageW + ix;
44             for (int k_j = 0; k_j < 3; k_j++) {
45                 sum += (float)Src_dev[idx + (k_j - 1)] * /*Completar*/;
46             }
47         }
48         idx = iy * imageW + ix;
49         Dst_dev[idx] = (unsigned char)(sum / 16);
50     }
51 }
```

FILTRO LAPLACIANO

```
56 __global__ void LaplacianFilter_kernel(float *Dst_dev, unsigned char *Src_dev,  
57                                     int imageW, int imageH){  
58  
59     const int ix = blockDim.x * blockIdx.x + threadIdx.x;  
60     const int iy = blockDim.y * blockIdx.y + threadIdx.y;  
61  
62     if (ix > 0 && ix < imageW - 1 && iy > 0 && iy < imageH - 1) {//Dejamos un margen de tamaño 1  
63         float sum = 0;  
64         int idx;  
65  
66         /*Completar*/  
67  
68         idx = iy*imageW + ix;  
69         Dst_dev[idx] = sum;  
70     }  
71 }
```

FILTROS: PROMEDIO, GAUSSIANO Y LAPLACIANO (C4)

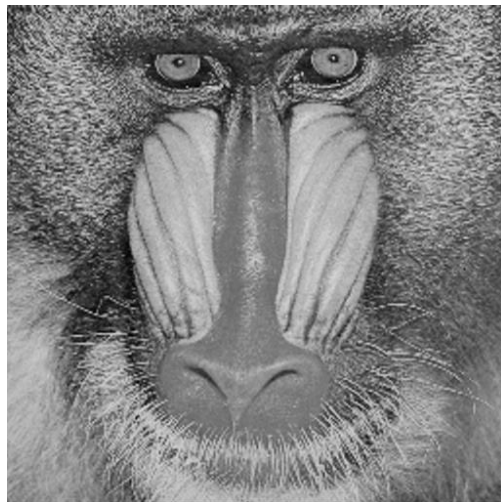
ImageFilters.h ImageFilters.cu* ImageFilters.cpp

```
73 void CUDA_MeanFilter(unsigned char *Dst_dev, unsigned char *Src_dev,int imageW,int imageH,  
74                     dim3 grid,dim3 threads){  
75     MeanFilter_kernel/*Completar*/(Dst_dev,Src_dev,imageW,imageH);  
76 }  
77  
78 void CUDA_GaussianFilter(unsigned char *Dst_dev, unsigned char *Src_dev, int imageW, int imageH,  
79                          dim3 grid, dim3 threads) {  
80     GaussianFilter_kernel/*Completar*/(Dst_dev, Src_dev, imageW, imageH);  
81 }  
82  
83 void CUDA_LaplacianFilter(float *Dst_dev, unsigned char *Src_dev,int imageW,int imageH,  
84                          dim3 grid,dim3 threads){  
85     LaplacianFilter_kernel/*Completar*/(Dst_dev, Src_dev,imageW,imageH);  
86 }
```

ImageFilters.h* ImageFilters.cu* ImageFilters.cpp

```
4 void CUDA_MeanFilter(unsigned char *Dst_dev,unsigned char *Src_dev,int imageW,int imageH,  
5                     dim3 grid,dim3 threads);  
6  
7 void CUDA_GaussianFilter(unsigned char *Dst_dev, unsigned char *Src_dev, int imageW, int imageH,  
8                          dim3 grid, dim3 threads);  
9  
10 void CUDA_LaplacianFilter(float *Dst_dev, unsigned char *Src_dev, int imageW, int imageH,  
11                          dim3 grid, dim3 threads);
```

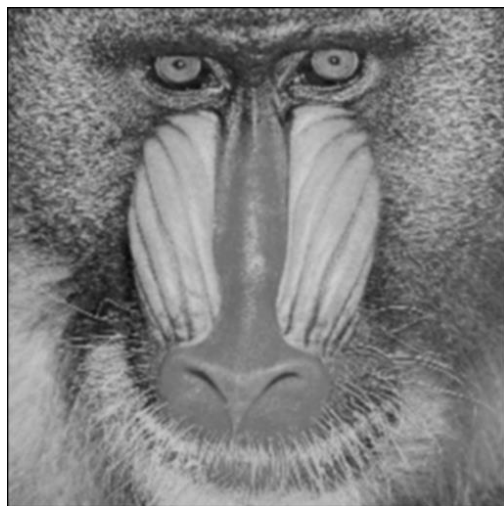
RESULTADOS



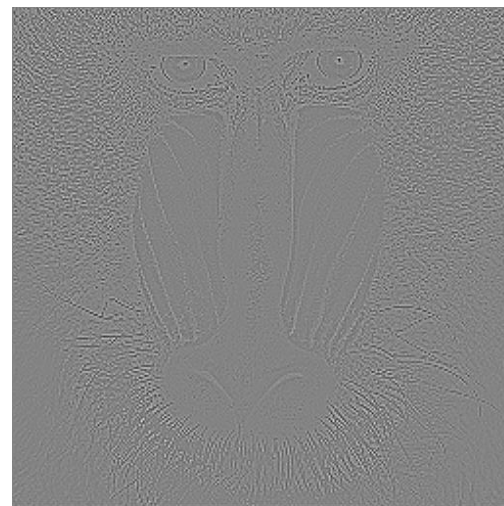
I_{Original}



$I_{\text{Filtrada}}, K_{\text{Promedio}}$



$I_{\text{Filtrada}}, K_{\text{Gaussiano}}$



$I_{\text{Filtrada}}, K_{\text{Laplaciano}}$

GRACIAS POR SU ATENCIÓN

Francisco J. Hernandez-Lopez

fcoj23@ciimat.mx

WebPage:

www.ciimat.mx/~fcoj23

