PRÁCTICAS AMPLIACIÓN DE ANÁLISIS NUMÉRICO
FACULTAD DE INFORMÁTICA

Práctica 1. Librerías para manejar imágenes. Descargar de la página web de la asignatura las librerias *ami_bmp.h* y *ami.h* .

Apartado 1.1. Implementar la función

void aan_unir_canales_unsigned_char(unsigned char *canal1, unsigned char *canal2, unsigned char **canal_output, int width, int height)

que toma los canales *canal1* y *canal2*, de tamaño (*width x height*) y genera el canal *canal_output* como unión de los dos canales en uno de tamaño *((width*2+4) x height)*, dejando una franja de 4 píxeles en negro entre ambos. Para comprobar el correcto funcionamiento de la función, aplicar el algoritmo a imágenes reales (en formato bmp), teniendo en cuenta que hay que aplicarlo a cada uno de los canales (rojo, verde y azul) que forman la imagen. Implementar la misma función para imágenes en precisión flotante, es decir:

void aan_unir_canales_float(float *canal1, float *canal2, float **canal_output, int width, int height)

Para leer y escribir las imágenes de/en disco, se utilizarán las funciones:

```
int ami_read_bmp(char name[200], unsigned char **red, unsigned
char **green, unsigned char **blue, int *width, int *height)

int ami_write_bmp(char name[200], unsigned char *red, unsigned
char *green, unsigned char *blue, int width, int height)
```

donde *name* es el nombre del fichero, *red*, *green* y *blue* son los 3 canales de la imagen, y *width* y *height* son las dimensiones.

Apartado 2.1. Implementar la función

void aan_normalizar_canal_unsigned_char(unsigned char *canal_input, unsigned char *canal_output, int width, int height)

que normaliza el vector canal_input entre los valores 0 y 255 y lo vuelca en el canal_output utilizando la fórmula

canal_output[i]=255*(canal_input[i]-min)/(max-min)

donde min y max son el máximo y el mínimo del vector canal_input[].

Utilizar esta función para implementar la función

void aan_normalizar_imagen_unsigned_char(unsigned char *red, unsigned char *green, unsigned char *blue, int width, int height)

que normaliza los valores de los 3 canales de una imagen.

Para comprobar el correcto funcionamiento de las funciones, aplicar el algoritmo de normalización a imágenes reales. Utilizar la función aan_unir_canales_unsigned_char() para crear una imagen que sea la original unida a la imagen después de normalizarla.

Implementar también las correspondientes versiones de estas funciones en precisión flotante.

Práctica 2. Implementar convolución con una máscara 3x3. Implementar la función

void aan_mascara_canal(float *canal_input, float *canal_output, int width, int height, float m[3][3])

que aplica una máscara 3x3 a un canal de una imagen. Utilizar esta función para implementar la función

void aan_mascara_imagen(float *red_input, float *green_input, float *blue_input, float *red_output, float *green_output, float *blue_output, int width, int height, float m[3][3])

para aplicar una mascara a una imagen.

Para comprobar el correcto funcionamiento de las funciones, aplicarlas a una imagen para calcular las derivadas parciales y el laplaciano.

1/4

$-(2-\sqrt{2})$	0	$(2-\sqrt{2})$
$-2(\sqrt{2}-1)$	0	$2(\sqrt{2}-1)$
$-(2-\sqrt{2})$	0	$(2-\sqrt{2})$

1/4

$-(2-\sqrt{2})$	$-2(\sqrt{2}-1)$	$-(2-\sqrt{2})$
0	0	0
$(2-\sqrt{2})$	$2(\sqrt{2}-1)$	$(2-\sqrt{2})$

1/3	1/3	1/3
1/3	-8/3	1/3
1/3	1/3	1/3

Utilizar las funciones de la práctica 1 para que el resultado de las pruebas sea la imagen original y al lado la imagen después de aplicar la máscara y normalizarla entre 0 y 255.

Práctica 3. Implementar la ecuación del calor. Implementar la función

void aan_ecuacion_calor_metodo_explícito(float *red_input, float *green_input, float *blue_input, float *red_output, float *green_output, float *blue_output, int width, int height, float dt, int Niter)

que realiza Niter iteraciones de la discretización explícita de la ecuación del calor tomando como incremento temporal dt. Es decir hay que implementar para cada canal el esquema numérico:

$$u_{i,j}^{n+1} = u_{i,j}^{n} + \frac{dt}{3}\left(u_{i+1,j+1}^{n} + u_{i-1,j-1}^{n} + u_{i+1,j-1}^{n} + u_{i-1,j+1}^{n} + u_{i+1,j}^{n} + u_{i-1,j}^{n} + u_{i,j+1}^{n} + u_{i,j-1}^{n} - 8u_{i,j}^{n}\right)$$

Hacer pruebas combinando los siguientes parámetros

Nt=100	dt=1/4
Nt=200	dt=1/8
Nt=400	dt=1/16
Nt=800	dt=1/32
Nt=50	dt=1/2

Para presentar los experimentos se creará una película que contenga todas las imágenes que se van generando según avanzan las iteraciones. Para la creación de la película se realizarán los siguientes pasos. En primer lugar, para cada *n* se salvará en disco la imagen correspondiente $u_{i,j}^{n}$. Dicha imagen se salvará en un fichero que tiene por nombre *imagen_m.bmp* donde *m* es el string que corresponde al número *100000+n*. La función en C necesaria para crear ese string es

sprintf(name,"imagen_%d.bmp",100000+n); /* crea el string name deseado */

Una vez generada en disco la secuencia de imágenes se utilizará un software de creación de videos para generar un fichero de vídeo con la secuencia de imágenes. Para hacerlo, se puede utilizar por ejemplo el software libre *VirtualDub* que se puede coger de http://prdownloads.sourceforge.net/virtualdub/VirtualDub-1.5.10.zip?download

Tomar un incremento temporal negativo equivale a intentar retroceder en el tiempo en la ecuación del calor, dicho proceso es altamente inestable pero si se aplica a una imagen desenfocada o difuminada, puede mejorar la nitidez. Hacer un vídeo tomando la evolución de la ecuación con un incremento temporal negativo dt= - 0.01.

Práctica 4. Implementar el método implícito para la ecuación del calor.
Implementar la función

void aan_ecuacion_calor_metodo_implícito(float *red_input, float *green_input, float *blue_input, float *red_output, float *green_output, float *blue_output, int width, int height, float dt, int Niter, float TOL)

que realiza Niter iteraciones de la discretización implícita de la ecuación del calor tomando como incremento temporal dt. TOL se utiliza para detener las iteraciones para resolver el método de Gauss-Seidel. Es decir hay que implementar para cada canal el esquema numérico :

$$u_{i,j}^{n+1} = u_{i,j}^{n} + \frac{dt}{3}\left(u_{i+1,j+1}^{n+1} + u_{i-1,j-1}^{n+1} + u_{i+1,j-1}^{n+1} + u_{i-1,j+1}^{n+1} + u_{i+1,j}^{n+1} + u_{i-1,j}^{n+1} + u_{i,j+1}^{n+1} + u_{i,j-1}^{n+1} - 8u_{i,j}^{n+1}\right)$$

o, despejando $u_{i,j}^{n+1}$:

$$u_{i,j}^{n+1} = \frac{1}{1+8\lambda}\left[u_{i,j}^{n} + \lambda\left(u_{i+1,j+1}^{n+1} + u_{i-1,j-1}^{n+1} + u_{i+1,j-1}^{n+1} + u_{i-1,j+1}^{n+1} + u_{i+1,j}^{n+1} + u_{i-1,j}^{n+1} + u_{i,j+1}^{n+1} + u_{i,j-1}^{n+1}\right)\right]$$

donde $\lambda = \frac{dt}{3h^2}$

Hacer pruebas combinando los siguientes parámetros

Nt=10, dt=1. TOL=0.01
Nt=10, dt=1. TOL=0.1
Nt=10, dt=1. TOL=0.5

Nt=10 dt=10 TOL=0.1
Nt=100 dt=100 TOL=0.1
Nt=100 dt=10000 TOL=0.1

Se realizarán algunas películas para ilustrar los resultados

Práctica 5. Implementar la ecuación del calor con un término externo. Implementar la función

void aan_ecuacion_calor_fuerza_externa metodo_explícito(float *canal_input, float *canal_output, int width, int height, float dt, int Niter,float A, float T0, float T1, float T2)

que realiza Niter iteraciones de la discretización explícita de la ecuación del calor con un término externo dada por

$$u_{i,j}^{n+1} = u_{i,j}^{n} + \frac{dt}{3}\left(u_{i+1,j+1}^{n} + u_{i-1,j-1}^{n} + u_{i+1,j-1}^{n} + u_{i-1,j+1}^{n} + u_{i+1,j}^{n} + u_{i-1,j}^{n} + u_{i,j+1}^{n} + u_{i,j-1}^{n} - 8u_{i,j}^{n}\right)$$

$$- A \cdot dt\,(u_{i,j}^{n} - T_0)(u_{i,j}^{n} - T_1)(u_{i,j}^{n} - T_2)$$

Aplicar la ecuación a imágenes reales para observar que las imágenes tienden a estabilizarse entorno a los colores T0 y T2. Quedando como resultado una segmentación de la imagen real en como máximo 8 niveles.

Nota: Hacer diferentes pruebas tomando diferentes valores de T0, T1 y T2, por ejemplo T0=0, T1=128, T2=256, o por ejemplo los valores calculados en clase en función del histograma h[i] del canal, es decir tomar para cada canal los valores

$$T_1 = \frac{\sum_{i=0}^{255} i \cdot h[i]}{\sum_{i=0}^{255} h[i]} \qquad T_0 = \frac{\sum_{i=0}^{T_1} i \cdot h[i]}{\sum_{i=0}^{T_1} h[i]} \qquad T_2 = \frac{\sum_{i=T_1}^{255} i \cdot h[i]}{\sum_{i=T_1}^{255} h[i]}$$

Se realizarán algunas películas para ilustrar los resultados

Práctica 6. Implementar la ecualización de histogramas. El objetivo de esta práctica implementar funciones que permitan modificar el histograma de una imagen. Ello se hará canal por canal. En principio tenemos el histograma de una imagen h[.] y tenemos un histograma objetivo dado por e[.] que es el histograma que nos gustaría que tuviese la imagen. Para pasar de un histograma a otro tenemos que definir un vector de paso f[.], de tal manera que el cambio que habría que hacer en la imagen es

canal_output[m]=f[canal_input[m]].

El criterio que se utiliza para definir f[.] es que se cumpla para todo k que

$$\sum_{i=0}^{f[k]} h[i] = \sum_{i=0}^{k} e[i]$$

En clase se vio el algoritmo para calcular f[.] siguiendo el anterior criterio. La práctica consiste en implementar la funciones

void aan_ecualizar_histograma(float *h, float *e, int *f) donde h[.],e[.] y f[.] tienen una dimensión de 256, h[.] es el histograma original, e[.] es el histograma objetivo (tanto h[.] como e[.] son conocidos) f[.] es el cambio de histograma que hay que hacer para pasar de h[.] a e[.].
Nota: Se supone que los histogramas h[.] y e[.] están normalizados, es decir que su suma vale 1.

void aan_ecualizar_histograma_canal(float *canal_input, float *canal_output, int width, int height, int *f) aplica el cambio de histograma dado por f[.] al canal_input[.] y lo devuelve en canal_output[.]

Hacer pruebas de cambio de histograma en imágenes con estas funciones. En los tests deberá aparecer una imagen compuesta por la imagen original a la izquierda, y la imagen después del cambio de histograma a la derecha.

Utilizar los siguientes valores para los histogramas objetivos

1. e[i]=1./256

2. e[i] viene definida entre i=0 e i=127 por una recta que en i=0 vale 1/512 y en i=127 vale 1/128. Entre i=128 e i=255, e[i] viene definida por una recta que en i=128 vale 1/128 y en i=255 vale 1/512. Tengase en cuenta que después de definir e[.] de esta manera habrá que normalizar el vector para que su suma sea 1.

Práctica 7. Implementar la ecuación de ondas. Implementar la función

void aan_ecuacion_ondas_metodo_explícito(float *red_input, float *green_input, float *blue_input, float *red_output, float *green_output, float *blue_output, int width, int height, float dt, int Niter)

que realiza Niter iteraciones de la discretización explícita de la ecuación de ondas tomando como incremento temporal dt. Es decir hay que implementar para cada canal el esquema numérico :

$$u_{i,j}^{n+1} = 2u_{i,j}^{n} - u_{i,j}^{n-1} + (dt)^2\left(u_{i+1,j+1}^{n} + u_{i-1,j-1}^{n} + u_{i+1,j-1}^{n} + u_{i-1,j+1}^{n} + u_{i+1,j}^{n} + u_{i-1,j}^{n} + u_{i,j+1}^{n} + u_{i,j-1}^{n} - 8u_{i,j}^{n}\right)$$

Inicialmente conocemos $u_{i,j}^{0}$ y $u_{i,j}^{1}$. Para los experimentos supondremos que $u_{i,j}^{0} = u_{i,j}^{1}$.

Las pruebas que se harán son las siguientes :

Test 1: Tomar dt=0.1, y 1000 iteraciones tomar una imagen formada por varios cuadrados (o círculos) de distintos colores sobre un fondo de nivel de gris 128. (tomar una imagen de dimensión pequeña (128x128) para que tengamos espacio en el disco para salvar la secuencia. Crear la secuencia de imágenes y crear la película. Hacer lo mismo para dt=0.01 y 1000 iteraciones. Hay que tener en cuenta que en este caso, las oscilaciones producidas por la ecuación de ondas pueden provocar que los niveles de gris superen el 255 o bajen por debajo de 0. Por tanto para pasarla al tipo unsigned char antes de guardarlas en disco tendremos que “cortar” las imágenes de tal manera que si el valor del nivel de gris es mayor de 255, le asignamos el valor 255, y si está por debajo de 0, le asignamos 0.

Test 2: Hacer lo mismo tomando una imagen real a ver que es lo que sale.

Práctica 8. La práctica consiste en calcular el movimiento que se produce en una secuencia de imágenes de tráfico que se puede encontrar en http://serdis.dis.ulpgc.es/~lalvarez/teaching/aan/aan_secuencia_trafico.zip
Para ello se utilizará la librería donde se implementa la correlación a ventanas que se encuentra en http://serdis.dis.ulpgc.es/~lalvarez/teaching/aan/aan_correlacion.zip

Se trata de coger 2 imágenes consecutivas de la secuencia y para cada canal de color calcular el movimiento existente tanto en la dirección horizontal como vertical. La manera de presentar los resultados será, por un lado presentar las dos imágenes de la secuencia elegidas, y por otro 2 imágenes que corresponden a los desplazamientos en horizontal y vertical (estas imágenes se normalizarán entre 0 y 255 para poder verlas). Probar con diferentes valores para los tamaños de las ventanas de correlación. Hacer pruebas también tomando como imagen de salida y de llegada la misma, con lo cual, en "teoría" no debería detectarse movimiento.

Práctica 9. La práctica consiste en calcular el desplazamiento "predominante" de los objetos presentes en una secuencia de imágenes. Se implementará una función que en primer lugar calcule todos los posibles desplazamientos utilizando la función de correlación. Posteriormente para aglutinar la información de los 3 canales de color, para cada punto, se quedará con el desplazamiento del canal que mayor valor de correlación haya dado. A continuación, se seleccionarán los puntos donde el módulo del desplazamiento encontrado haya sido mayor que un cierto umbral (por ejemplo umbral=2) que será un parámetro de la función. Si el número de puntos seleccionados es más pequeño que un cierto umbral (parámetro de la función), la función devuelve el vector (0,0), es decir, no ha encontrado ningún objeto en movimiento. Finalmente del conjunto de puntos seleccionados se calcularán los valores medianos de los desplazamientos en horizontal y vertical y la función devolverá esos valores medianos (un vector de 2 valores) que determinarán el desplazamiento "predominante" encontrado en la imagen. Las funciones para calcular la mediana de un conjunto de valores se pueden encontrar en
http://serdis.dis.ulpgc.es/~lalvarez/teaching/aan/aan_mediana.h

Hacer pruebas con la secuencia de tráfico de la práctica anterior. Hacer pruebas también tomando como imagen de salida y de llegada la misma, con lo cual, en "teoría" no debería detectarse movimiento.

Se deberá también incluir en la función el criterio de utilizar para calcular el desplazamiento sólo los puntos que pertenecen a los bordes, es decir que el módulo de su gradiente es suficientemente alto.

Práctica 10. Implementar la ecuación de Perona-Malik. Implementar el modelo de Perona-Malik, dado por la ecuación diferencial

$$\frac{\partial u}{\partial t} = div\left(g(\|\nabla u\|)\nabla u\right)$$

donde

$$g(\|\nabla u\|) = e^{-\lambda\|\nabla u\|}$$

siguiendo el esquema numérico visto en clase. Dicho esquema consiste en lo siguiente: Si llamamos

$$k_{i,j} = e^{-\lambda\|\nabla u_{i,j}\|}$$

entonces para pasar de la iteración $u_{i,j}^{n}$ a la iteración $u_{i,j}^{n+1}$, hay que hacer

$$u_{i,j}^{n+1} = u_{i,j}^{n} + \frac{\delta t}{2(\delta h)^2} M(u_{i,j}^{n})$$

donde $M(u_{i,j}^{n})$ es el resultado de aplicar en cada punto (i,j) la máscara que tiene por coeficientes

$$\begin{bmatrix} 0 & \left(k_{i,j} + k_{i,j+1}\right) & 0 \\ \left(k_{i,j} + k_{i-1,j}\right) & \left(-k_{i+1,j} - k_{i-1,j} - 4k_{i,j} - k_{i,j+1} - k_{i,j-1}\right) & \left(k_{i,j} + k_{i+1,j}\right) \\ 0 & \left(k_{i,j} + k_{i,j-1}\right) & 0 \end{bmatrix}$$

Sugerencia: La manera más fácil de implementarlo es utilizar el código de la función que aplica una máscara (ya implementada) y modificarlo para que en cada punto se aplique una máscara distinta.

Realizar películas con las iteraciones del proceso para diferentes valores del parámetro lambda. Tomar como valores de λ, λ=0.01, λ=0.001, λ=0.0001