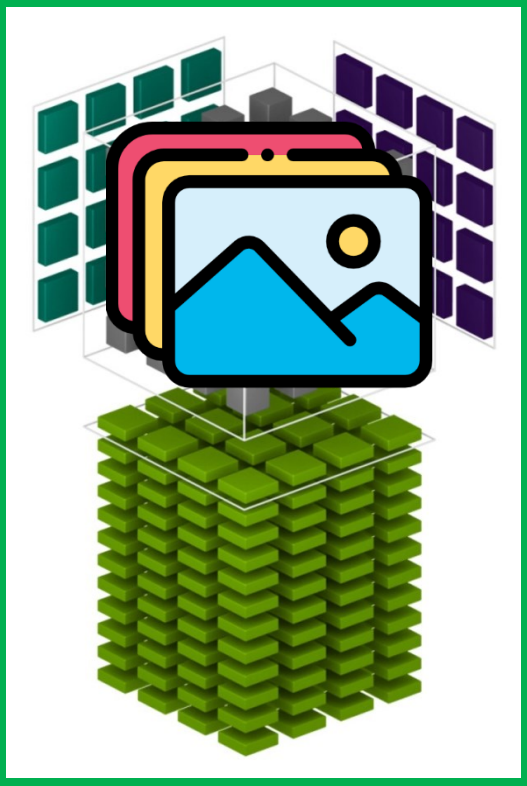
Universidad de Alcalá

Paradigmas Avanzados de Programación

Enunciado PECL1

Manipulación de Imágenes



CURSO 2024/2025 – CONVOCATORIA ORDINARIA

Versión: 2 (última revisión, 11 de marzo de 2025).

Índice

1. Introducción

En esta PEC de laboratorio se explora la manipulación de imágenes. Como un caso de uso representativo de la programación en GPU mediante CUDA en la vida real. La manipulación y procesamiento de imágenes es una tarea computacionalmente intensiva que se beneficia enormemente del paralelismo masivo que ofrecen las tarjetas gráficas.

2. Trabajo a realizar.

2.1. Consideraciones iniciales.

Algunas consideraciones a tener en cuenta:

- Salvo ciertas acciones (p.ej., leer/guardar imágenes de/en disco, gestión del menú de opciones), que puede ser realizado desde la parte de la CPU del programa, las diferentes partes solicitadas obligatoriamente deberán realizarse en la GPU. Mediante programación en CUDA C.
- Como excepción, si alguien hiciera uso de la máquina virtual con CUDA, se le permite realizar alguna acción desde el host. Esto es debido a que la versión de CUDA de la máquina virtual no tiene implementación de algunas funciones de CUDA. Antes de proceder, consultar con los profesores.
- Las salidas principales del programa serán vía consola y vía creación de ficheros (imágenes) en el host.
    - No se sobrescribirá la imagen original.
    - Entenderemos como ruta base donde se encuentre la imagen original. En esa ruta (carpeta) es donde se generarán las diferentes imágenes modificadas.
    - La salida vía consola deberá incluir trazas para seguir la ejecución del programa.
- El acceso a la imagen de memoria global de la GPU será obligatoriamente realizado mediante acceso linealizado (1D).
- Las modificaciones sobre las imágenes se realizarán mediante código C propio, escrito en kernels de CUDA. No se permite la utilización de librerías externas, como OpenCV o similares.
- Todo el código entregado deberá estar perfectamente comentado. Aquella práctica que no tenga el código bien documentado puede ser suspendida.
- Se recomienda comenzar la codificación con imágenes con unas dimensiones iniciales que permitan ejecutar el programa en un único bloque (Stream multiprocesador, SM) y el uso de la memoria global. Para facilitar la codificación. La cantidad máxima de hilos por bloque es dependiente del hardware, pero en general serán imágenes con unas dimensiones muy reducidas. Una vez superado este paso, avanzar en lanzar imágenes que requieren múltiples bloques.

2.2. Consideraciones sobre la manipulación de imágenes en C.

Si bien formatos gráficos como JPEG (o JPG) y PNG son formatos de imágenes bastante estandarizados y difundidos hoy en día, para la manipulación de imágenes a nivel de píxel (que es el enfoque de esta PECL) no son adecuados. Estos formatos aplican compresión y codificación, más o menos compleja. Para esta PECL, **recomendamos la utilización del formato BMP** (*Bitmap Image File*). No aplica compresión a los píxeles. Esto facilita la lectura, escritura y manipulación de los datos en bruto sin necesidad de decodificación. De forma rudimentaria, una vez pasada la cabecera de datos, los píxeles que conforman la imagen son una matriz de datos con codificación RGB (un canal por cada color).

Lo más sencillo será buscar imágenes en internet, y convertirlas a BMP para utilizar en el código. La herramienta de conversión recomendada es esta: https://imagen.online-convert.com/es/convertir-a-bmp. Los ejemplos de las manipulaciones a realizar se mostrarán sobre la Ilustración 1.

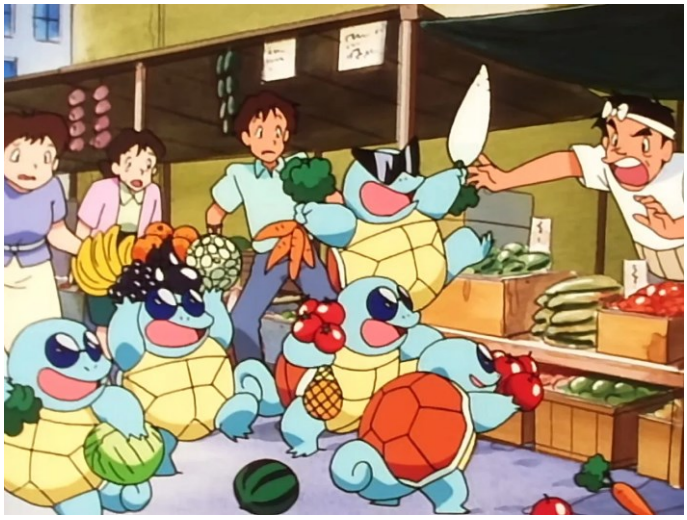


Ilustración 1. Imagen que usaremos para ilustrar las diferentes modificaciones solicitadas.

Para la lectura/escritura de ficheros BMP, hay muchos tutoriales en internet como, por ejemplo (no han sido probados, pero parecen correctos 👍):

- https://elcharolin.wordpress.com/2018/11/28/read-and-write-bmp-files-in-c-c
- https://gist.github.com/senior-sigan/67645671ff8c4d0c88599fb3ee41308b

Para la manipulación, se recomienda que la matriz convertida de la imagen sea una matriz en la que cada celda sea un struct de C (salvando las distancias, como un objeto de Java), en la que tenga los atributos de la codificación RGB:

```
typedef struct {
      unsigned char r, g, b;
} Pixel;
```

Los structs se le pueden pasar como tipo de dato a los kernels de CUDA sin problemas. Siempre y cuando estén definidos en el código antes que el kernel en cuestión. Sino, es probable que el compilador muestre algún error.

2.3. Menú de inicio.

La aplicación pedirá, al arrancarse, la ruta donde se encuentra la imagen BMP base para el resto de las funciones disponibles. Para facilitar las ejecuciones, se puede disponer de una ruta ya precargada para utilizar por defecto.

```
 PECL 1 PAP
Introduzca la ruta base de la imagen:
(pulse Intro para usar por defecto: C:\PAP\img00.bmp)
```

Ilustración 2. Ejemplo de inicio de ejecución.

Se mostrará un típico menú. Con las opciones disponibles. Estas opciones son las diferentes manipulaciones presentadas en los próximos apartados.

```
Opciones
        (1) Conversi¾n a Blanco y Negro.
        (2) Pixelar.
        (3) Filtrar colores.
        (..) .........

        (X) Salir.
```

Ilustración 3. Ejemplo de menú y sus opciones.

Al finalizar la ejecución de cada una de las modificaciones planteadas, la aplicación volverá al menú de inicio. Hasta que el usuario quiera salir, se ofrecerán las opciones, manteniendo la imagen base introducida en el inicio de la ejecución. Opcionalmente, se puede incluir la opción de modificar la ruta de la imagen base.

2.4. Fase 01. Conversión a blanco y negro (obligatorio).

Dada la imagen original, se procederá a generar una versión en blanco y negro. La imagen resultante se guardará en disco. Por ejemplo, la Ilustración 1 tendría como resultado algo similar a lo mostrado en la Ilustración 4.

Ilustración 4. Fase 01. Imagen en blanco y negro.

Dado un píxel, conformado en codificación RGB, la conversión a blanco y negro es aplicar la siguiente fórmula a cada uno de los canales de color del píxel.

$$PGris_{(R,G,B)} \cong (0.299 \times P_{(R)}) + (0.587 \times P_{(G)}) + (0.114 \times P_{(B)})$$

Es una fórmula que permite ajustar el espectro de colores de la imagen a la percepción del ojo humano de cada canal. Los valores son aproximados. Y no son únicos. Se permite jugar con los pesos para obtener una mejor percepción. Fuentes: [1] [2].

El programa deberá comprobar cuáles son las características hardware de la tarjeta donde se va a realizar la ejecución. Dependiendo de dichas características, el problema deberá dimensionarse de una forma u otra para poder ser resuelto (pista: implementar el problema en N bloques M hilos por bloque y que estos valores no sean fijos a priori).

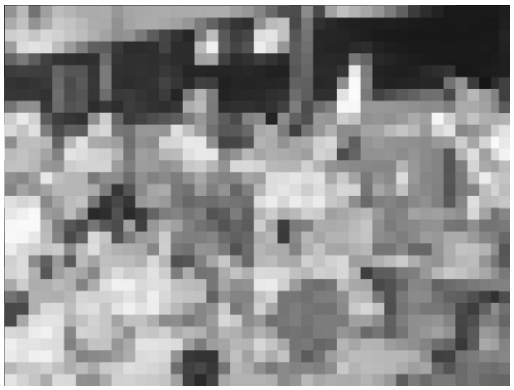
2.5. Fase 02. Pixelar imagen (obligatorio).

Dada la imagen original, se procederá a generar una versión pixelada (es decir, difuminada aplicando un filtro de cristalización con un rango determinado). La imagen resultante se guardará en disco. Por ejemplo, la Ilustración 1 tendría como resultado algo similar a lo mostrado en la Ilustración 5.

Ilustración 5. Fase 02. Imagen pixelada.

El usuario podrá elegir el rango de acción. Cuánto más grande, más distorsión. Se puede observar una comparación en la Ilustración 6.

(a) Diámetro de 15 píxeles.



(b) Diámetro de 30 píxeles.

Ilustración 6. Fase 02. Ejemplos de rango de pixelado.

El usuario también podrá elegir si quiere pixelar sobre la imagen original (en color) o sobre la imagen en blanco y negro.

Para cada píxel, se calcula la región que le corresponde al filtro (el filtro especifica el diámetro de un cuadrado, puede que no regular, en el que el píxel es el centro), y toma los píxeles dentro de esa región para calcular el valor promedio de cada canal de color. Luego, asigna el valor promedio a todos los canales de color del píxel. Los píxeles frontera de la imagen solo trabajan con elementos dentro de la imagen; lo que está fuera de la imagen, no suma ni aporta al valor promedio.

$$\begin{cases} V_R = \sum_{k=0}^{k<TAM_{Filtro}} \sum_{j=0}^{j<TAM_{FILTRO}} P_{R(x+k,y+j)} \\ V_G = \sum_{k=0}^{k<TAM_{Filtro}} \sum_{j=0}^{j<TAM_{FILTRO}} P_{G(x+k,y+j)} \\ V_B = \sum_{k=0}^{k<TAM_{Filtro}} \sum_{j=0}^{j<TAM_{FILTRO}} P_{B(x+k,y+j)} \end{cases}$$

$$P'_{(x,y)Canal_{Color}} = \frac{V_{Canal_{Color}}}{TOTAL_{ELEMENTOS-FILTRO}}$$

Esta manipulación requiere de la aplicación de memoria compartida para su resolución. Según las características del hardware, se puede impedir que determinados diámetros de filtro sean aplicados. Por exceso en la memoria compartida a utilizar.

El programa deberá comprobar cuáles son las características hardware de la tarjeta donde se va a realizar la ejecución. Dependiendo de dichas características, el problema deberá dimensionarse de una forma u otra para poder ser resuelto (pista: implementar el problema en N bloques M hilos por bloque y que estos valores no sean fijos a priori).

2.6. Fase 03. Identificación de colores (obligatorio)

En la codificación RGB los pixeles con colores primarios "*puros*" son aquellos que su componente principal es el máximo valor del rango (es decir, 255), y el resto de los componentes es cero. Por ejemplo, el color rojo es el (255,0,0), el verde es el (0,255,0), y el azul es el (0,0,255).

Sin embargo, la presencia de colores primarios "*puros*" en las imágenes es difícil de conseguir. Las combinaciones posibles de los tres colores primarios generan una paleta de colores grande. (255,0,0) ( ) es rojo, pero (249,78,78) ( ) o (240.50.30) ( ) también son considerados rojo al ojo humano. En mayor o menor medida.

De ahí que para considerar que el píxel es de color rojo, consideraremos que el píxel es principalmente rojo si su componente roja es mayor que las componentes de los colores restantes. ~~Aplicando un factor de magnitud, para determinar cuánto queremos medir esa diferencia entre componentes. Factor normalmente entre 1 y 2. Además, la componente roja debe superar un determinado umbral mínimo de color aportado. Normalmente 100.~~

~~Por tanto:~~

$$f_{esRojo}(P_{(x,y)})\begin{cases}(R > (G \times fm)) \; AND \; (R > (B \times fm)) \\ AND \\ R > u\end{cases}$$

~~Siendo, fm el factor de magnitud y u el umbral mínimo del color aportado.~~

~~Por ejemplo, para la Ilustración 1, aplicando $fm = 1.5$ y $u = 150$, tendríamos 118.390 píxeles que podemos considerar como rojos. Si generamos la imagen resultante, obtendríamos algo similar a lo mostrado en Ilustración 7.~~

~~*Ilustración 7. Fase 03. Resultado de filtrar el color rojo.*~~

~~Como puede observarse, podría aplicarse un refinamiento en el factor de magnitud y en el umbral para acercarse más a tener píxeles más cercanos al rojo, según el ojo humano. Podemos ver que la configuración usada identifica algunos píxeles que podríamos considerar como "rosa" o "naranja".~~

La fórmula anterior tiene problemas. Para el color rojo parece funcionar bien. Sin embargo, para el resto de los colores se obtienen resultados muy distorsionados según la imagen de entrada.

Se propone la siguiente fórmula:

$$f_{esRojo}()\begin{cases}100 \leq R \leq 255 \; AND \\ 0 \geq \; G < 150 \; AND \\ 0 \geq \; B < 150\end{cases}$$

$$f_{esVerde}()\begin{cases}30 < R \leq 150 \; AND \\ 50 \geq \; G \leq 255 \; AND \\ 0 \geq \; B < 75\end{cases}$$

$$f_{esAzul}()\begin{cases}0 \leq R \leq 200 \; AND \\ 0 \geq \; G < 250 \; AND \\ 100 \geq \; B \leq 255\end{cases}$$

Se recomiendan estos valores como elementos por defecto. El usuario puede especificar otro rango. Se deja abierto a los estudiantes, y se valorará, la optimización de los valores para obtener un resultado óptimo acorde al ojo humano.

Para la Ilustración 1, sobre el color rojo obtenemos 273.834 píxeles que cumplen el umbral. Se puede observar un resultado filtrado similar al de la Ilustración 8.

Ilustración 8. Fase 03. Resultado de filtrar el color rojo.

Para la Ilustración 1, sobre el color verde obtenemos 141.630 píxeles que cumplen el umbral. Se puede observar un resultado filtrado similar al de la Ilustración 9.

Ilustración 9. Fase 03. Resultado de filtrar el color verde.

Para la Ilustración 1, sobre el color azul obtenemos 257.161 píxeles que cumplen el umbral. Se puede observar un resultado filtrado similar al de la Ilustración 10.

Ilustración 10. Fase 03. Resultado de filtrar el color azul.

Así pues, en esta modificación requiere que se muestre por pantalla el número de píxeles que son de color rojo, cuántos de color verde y cuántos de color azul. Se deja vía libre para buscar una configuración que sea más adecuada a la visión del ojo humano. Así mismo, se guardará la imagen resultante de filtrar cada color dado en disco.

El programa requerirá al usuario ~~el factor de magnitud y umbral~~ los umbrales de color aplicados a usar (se pueden tener valores por defecto). Y generará 3 imágenes, una por cada color. Los píxeles que no sean del color identificado se pintarán en gris o blanco.

Esta manipulación **requiere de la aplicación de operaciones atómicas** para su resolución.

También se **requiere la utilización de la memoria de constantes**, al menos para almacenar ~~el factor de magnitud y el umbral aplicado~~ los umbrales de color aplicados.

El programa deberá comprobar cuáles son las características hardware de la tarjeta donde se va a realizar la ejecución. Dependiendo de dichas características, el problema deberá dimensionarse de una forma u otra para poder ser resuelto (pista: implementar el problema en N bloques M hilos por bloque y que estos valores no sean fijos a priori).

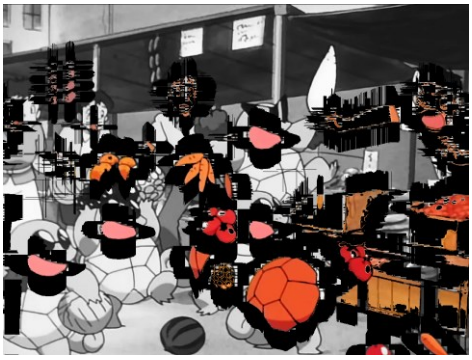
2.7. Fase 04. Filtrado y delineado de zonas de color (obligatorio).

Partiendo de la fase previa, de identificar un color principalmente primario, se generará una imagen en blanco y negro en la que aparecerán en color los píxeles que cumplan la identificación de un determinado color primario (principalmente rojo, verde o azul). Los píxeles de color seleccionado aplicaran un halo alrededor que permita delimitar la zona del color identificado. Este halo será de color negro. Partiendo de la Ilustración 1, obtendríamos un resultado similar a lo mostrado en la Ilustración 11.

Ilustración 11. Fase 04. Resultado de filtrado y delineado.

El halo simplemente indica el ancho de la línea de delineado en cada dirección. Según el valor del halo, esta línea será más o menos gruesa. Puede observarse en la Ilustración 12 unos ejemplos de variación del halo. Esto es un ejemplo, ya que puede observarse un delineado un poco tosco. Se deja libertad (y se valorará) la optimización en el suavizado del delineado a aplicar sobre la imagen.

(a) Aplicando un halo de tamaño 15



(b) Aplicando un halo de tamaño 50

Ilustración 12. Fase 04. Ejemplos de filtrado y delineado.

El programa pedirá al usuario el color primario a filtrar y el tamaño del halo de delineado a aplicar en la imagen resultante.

Esta manipulación requiere de la aplicación de memoria compartida para su resolución.

El programa deberá comprobar cuáles son las características hardware de la tarjeta donde se va a realizar la ejecución. Dependiendo de dichas características, el problema deberá dimensionarse de una forma u otra para poder ser resuelto (pista: implementar el problema en N bloques M hilos por bloque y que estos valores no sean fijos a priori).

2.8. Fase 05. Cálculo de pseudo-hash (obligatorio).

Se busca obtener un código pseudo-hash para la imagen original (es decir, la imagen de entrada al programa). Para ello, en esta manipulación se aplicará el patrón paralelo de reducción para obtener un vector de números enteros.

El patrón paralelo buscará aplicar el máximo de cada pareja de valores comparado. Cómo inicialmente la imagen está formada por píxeles, se transformará el pixel en base a la siguiente función para obtener un número entero de cada píxel (se garantiza que cada valor inicial está comprendido entre 0 y 255, ambos inclusive):

$$P_{(x,y)} = \left(I_{(x,y).Rojo} \times 0{,}50\right) + \left(I_{(x,y).Verde} \times 0{,}25\right) + \left(I_{(x,y).Azul} \times 0{,}25\right)$$

La reducción, como se comentaba, busca encontrar el máximo de la pareja de valores a comparar en cada fase:

$$V = \max\left(P(x), P(x')\right)$$

Nota: Según la versión del compilador de C, puede que la función max() no esté disponible para poder utilizar y haya que codificarla manualmente.

El kernel de un patrón de reducción generará un resultado de máximos en un vector de N elementos (siendo N el número de bloques que se lance en el kernel). Queremos que el resultado sea un vector de, como máximo, 15 posiciones. Esto probablemente implicará reducciones sucesivas.

Pista: Se realiza, en un kernel, la reducción de la imagen original a un vector de valores parciales de tamaño N. Si N es superior a 15, se aplica una segunda reducción en otro kernel hasta obtener 15 valores o menos.

Para finalizar, queremos mostrar por consola el valor textual del pseudo-hash. Habrá que normalizar los valores obtenidos en la reducción (entre 0 y 255, ambos inclusive) a valores de caracteres imprimibles en la tabla ASCII (entre 35 y 125, ambos inclusive).

Nota: Los siguientes ejemplos, se han lanzado reducciones con 512 hilos/bloque. Otras configuraciones de hilos/bloque pueden generar otro pseudo-hassh.

La Ilustración 1, imagen base de este enunciado, generaría lo siguiente:

Hash (ASCII)	xv|wv|qxvvwlwqx
Hash (núm. normalizado)	120 118 124 119 118 124 113 120 118 118 119 108 119 113 120
Hash (núm. original)	243 236 253 239 236 253 222 243 236 237 239 209 239 222 241

Una imagen como la mostrada en la Ilustración 13, genera el siguiente pseudo-hash. Y, para cualquier imagen con todos sus píxeles en rojo "puro" (es decir, (255,0,0)), independiente de su tamaño, generará siempre el mismo pseudo-hash. Lo mismo, con su respectivo hash, ocurre con imágenes con todos sus píxeles con el mismo color.

Hash (ASCII)	OOOOOOOOOOOOOOO (letra "o" mayúscula)
Hash (núm. normalizado)	79 79 79 79 79 79 79 79 79 79 79 79 79 79 79
Hash (núm. original)	127 127 127 127 127 127 127 127 127 127 127 127 127 127 127

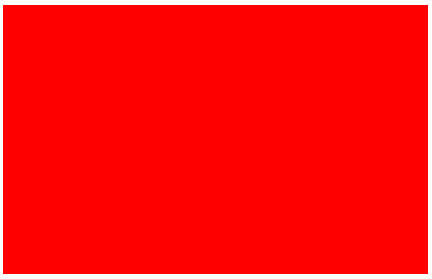


Ilustración 13. Fase 05. Imagen de muestra de color uniforme.

Por último, una imagen como la mostrada en Ilustración 14, generaría lo siguiente:

Hash (ASCII)	skloglhi`fmclfi
Hash (núm. normalizado)	115 107 108 111 103 108 104 105 96 102 109 99 108 102 105
Hash (núm. original)	227 204 207 216 195 207 196 200 174 191 212 183 207 190 201

Ilustración 14. Fase 05. Imagen de muestra 03.

Esta manipulación requiere de la aplicación de memoria compartida para su resolución. Y se valorará la aplicación del patrón con limitada divergencia de los hilos. La solución base y optimizada de este patrón está en las diapositivas de clase. Más información sobre la optimización del patrón paralelo, se puede [consultar la documentación de nVidia].

El programa deberá comprobar cuáles son las características hardware de la tarjeta donde se va a realizar la ejecución. Dependiendo de dichas características, el problema deberá dimensionarse de una forma u otra para poder ser resuelto (pista: implementar el problema en N bloques M hilos por bloque y que estos valores no sean fijos a priori).

2.9. Implementación Gráfica (extra)

Como apartado extra, se posibilita la opción realizar la implementación de la aplicación con librerías 3D. Las manipulaciones seguirán siendo obligatorias realizarse en kernels. Pero la visualización de los resultados, menús y mensajes se pueden realizar en una aplicación con apartado gráfico. En otras palabras, una interfaz de usuario gráfica. Es un apartado completamente opcional, y sirve para obtener puntos extra.

A aquellos alumnos que implementen el programa utilizando para ello las librerías gráficas OpenGL/DirectX, podrán obtener puntos extra, aunque será necesario haber realizado cada una de las fases de manipulación previas.

Nota: Este apartado no tiene soporte por parte del profesorado, debido a la diversidad de librerías gráficas e implementaciones específicas en cada GPU.

3. Documentación a entregar

La entrega de la PECL1 deberá constar de:

- Proyectos de Visual Studio con la solución codificada.
  - 1x Proyecto de implementación con todas las fases solicitadas.
  - (extra) (opcional) 1x Proyecto de implementación 3D.
- Breve video comentando la ejecución
  - 1x Video general de la aplicación.
  - El vídeo no superará los 3 minutos de duración. Deben participar ambos miembros de la pareja de trabajo. Se comentarán los detalles clave de la implementación y demostración de ejecución.
- Breve memoria PDF comentando los detalles clave de la implementación.

4. Forma de entrega

El trabajo se realizará en parejas. Se entregará mediante el correspondiente Buzón de Entrega disponible en BlackBoard. La fecha tope de entrega y fecha de Defensa se encuentra disponible en BlackBoard. Al ser un trabajo en pareja, solamente es necesario que uno de los miembros haga la entrega. Puede que el buzón se mantenga visible pasada la fecha tope de entrega. Si se reciben entregas tardías, serán penalizadas con la mitad de la nota obtenida.

Nota: No se admitirán entregas enviadas al correo electrónico (o mensajería interna de BlackBoard) de los profesores. Únicamente se permite la interacción con el Buzón de entrega.

5. Defensa

La defensa de las prácticas será en la fecha y la forma indicada por el/la profesor/a del laboratorio, pudiendo ser esta oral o escrita. En el caso de no contestarse correctamente a las cuestiones presentadas, la práctica presentada podrá considerarse como suspensa.

Es obligatorio que todos los miembros de la práctica acudan a la Defensa. Salvo causa justificada. De no ser así, la práctica presentada podría considerarse como suspensa para los miembros que no acudan a la Defensa.

6. Evaluación

La nota máxima a la que puede acceder se pondera en función de:

a) Calidad del material entregado por el alumno.
b) Documentación de seguimiento presentada durante su realización
c) Realización e implementación de la práctica.
d) Manejo del software utilizado para el desarrollo de la práctica.
e) Código desarrollado.
f) Defensa e implementación de las modificaciones solicitadas.

La puntuación de cada apartado es el siguiente:

Sección	Nota
Fase 01 *	1 pt.
Fase 02 *	2 pts.
Fase 03 *	1 pt.
Fase 04 *	2 pts.
Fase 05 *	2 pts.
Implementación Gráfica (extra)	(máximo 2 puntos extra)
Documentación entregada *	2 pts.
Defensa *	0% - 100% de la nota de la PCL1

* Elementos de obligada realización. Partes entregadas manifiestamente erróneas supondrá el suspenso de la PECL en su conjunto.