APUNTES DE SISTEMAS INFORMÁTICOS

UD2 - REPRESENTACIÓN DE LA INFORMACIÓN

Módulo: 0483. Sistemas informáticos
Ciclo Formativo: IF303 - Técnico Superior en Desarrollo de Aplicaciones Web
Profesora: María Paradela García

UD1 – Representación de la información

1. Introducción

En informática, todo —absolutamente todo— se reduce a unos y ceros.

Por muy complejo que parezca un videojuego, una película o una aplicación de inteligencia artificial, en el fondo no son más que secuencias de estados eléctricos que el ordenador interpreta como datos: imágenes, sonidos o texto.

Esa capacidad de convertir el mundo real en información digital es el cimiento de toda la tecnología moderna. Cada vez que enviamos un mensaje, hacemos una foto o vemos un vídeo, lo que realmente ocurre es que un sistema informático traduce fenómenos físicos a números y los manipula con reglas matemáticas.

Antes de comprender cómo un ordenador ejecuta un programa o transmite información por una red, debemos entender cómo representa y maneja los datos.

En esta unidad aprenderás:

- Qué significa realmente “información digital”.
- Cómo se mide y se codifica la información.
- Qué son los bits y los bytes.
- Cómo funcionan los sistemas de numeración binario, decimal, octal y hexadecimal.
- Y por qué este conocimiento es esencial para detectar errores, optimizar recursos y comprender los límites de la tecnología.

2. Representación de la información en los sistemas informáticos

Un ordenador no entiende palabras, colores ni sonidos directamente: solo niveles eléctricos, que pueden estar activos o inactivos, encendidos o apagados.

Estos estados se representan mediante bits, abreviatura de binary digit, es decir, “dígito binario”.

Cada bit puede tomar solo dos valores posibles:

Valor lógico	**Estado físico**	**Significado**
0	Ausencia de señal (0 voltios aprox.)	“apagado”
1	Presencia de señal (3–5 voltios aprox.)	“encendido”

Por tanto, todo lo que un ordenador procesa se traduce finalmente a cadenas de 0 y 1.

La letra “A”, un icono de tu escritorio o una fotografía digital son, en última instancia, secuencias de bits.

Dato, información y conocimiento

Aunque en lenguaje cotidiano se usan como sinónimos, en informática no significan lo mismo:

Concepto	Descripción	Ejemplo
Dato	Valor aislado sin interpretar	36,7
Información	Dato interpretado con contexto	“Temperatura corporal: 36,7 °C”
Conocimiento	Uso de la información para tomar decisiones	“El paciente está sano.”

El ordenador procesa datos, almacena información y ayuda a generar conocimiento, pero no “entiende” su significado.

Su inteligencia radica en la precisión y velocidad con la que opera con bits.

2.1 El bit y el byte

Un solo bit no puede expresar demasiada información (solo dos estados posibles).

Por eso se agrupan en conjuntos. El más habitual es el byte, compuesto por 8 bits.

Cada bit tiene una posición y un valor distinto según su peso.

El bit situado más a la izquierda se denomina MSB (Most Significant Bit, bit más significativo), y el más a la derecha LSB (Least Significant Bit, menos significativo).

Bits	Combinaciones posibles	Valores representables
1	2	0 – 1
2	4	0 – 3
4	16	0 – 15
8	256	0 – 255
16	65.536	0 – 65.535
32	4.294.967.296	0 – 4.294.967.295

Con 8 bits (un byte) podemos representar 256 combinaciones distintas, desde 00000000 hasta 11111111.

Esa capacidad basta para codificar todos los caracteres básicos del alfabeto inglés (código ASCII).

Ejemplo 1:
Un carácter ASCII (por ejemplo, la letra “A”) ocupa 1 byte.
A → 01000001
Cada bit tiene un peso distinto:
(0×128) + (1×64) + (0×32) + (0×16) + (0×8) + (0×4) + (0×2) + (1×1) = 65
El valor decimal 65 identifica el carácter “A” en la tabla ASCII.

Ejemplo 2:
Una imagen de 1024×768 píxeles en blanco y negro (1 bit por píxel) ocupa:
1024 × 768 bits = 786.432 bits = 98.304 bytes ≈ 96 KB.
Si esa misma imagen fuese en color con 24 bits por píxel (8 por cada canal RGB), ocuparía 2.359.296 bytes ≈ 2,25 MB.
Por eso, el número de bits por píxel (profundidad de color) determina la calidad y el tamaño de la imagen.

Nota técnica
Los procesadores actuales operan internamente con grupos de 32 o 64 bits.
Esto define el tamaño de palabra del sistema, es decir, la cantidad de datos que puede procesar simultáneamente.
También determina los límites numéricos que puede manejar cada tipo de dato.
Por ejemplo, un número entero sin signo de 8 bits no puede superar el valor 255. Si intentas guardar 256, se produce un overflow (desbordamiento).

2.2 Unidades de medida de información

Para manejar cantidades mayores, se utilizan múltiplos del byte:

Unidad	Equivalencia (binaria)	Equivalencia (decimal aproximada)
1 Kilobyte (KB)	1.024 bytes	1.000 bytes
1 Megabyte (MB)	1.024 KB	1.000.000 bytes
1 Gigabyte (GB)	1.024 MB	1.000.000.000 bytes
1 Terabyte (TB)	1.024 GB	1.000.000.000.000 bytes

Advertencia:
En almacenamiento digital (discos duros, SSD) los fabricantes suelen usar el sistema decimal, mientras que los sistemas operativos usan el binario. Por eso un disco de “500 GB” aparece en el ordenador como unos 465 GB.

Ejemplos prácticos:

Tipo de archivo	Tamaño aproximado
Documento de texto (3.000 caracteres)	3 KB
Canción MP3 (3 minutos, 128 kbps)	3 MB
Foto HD (12 MP)	4–5 MB
Película HD (2 horas)	3–4 GB
Juego moderno	50–100 GB

Comprender estas magnitudes ayuda a dimensionar el almacenamiento y a entender el consumo de datos en redes.

3. Sistemas de numeración

Un sistema de numeración define cómo representamos los números mediante símbolos y una base determinada.

El sistema decimal (base 10) es natural para los humanos porque usamos diez símbolos (0–9).

Sin embargo, los ordenadores trabajan con otras bases, principalmente binaria (base 2), aunque también se usan octal (base 8) y hexadecimal (base 16).

Sistema	Base	Símbolos	Ejemplo
Decimal	10	0–9	235_{10}
Binario	2	0, 1	11101011_{2}
Octal	8	0–7	353_{8}
Hexadecimal	16	0–9, A–F	EB_{16}

3.1 El sistema binario

Cada dígito binario (bit) representa una potencia de 2:

Posición	Potencia	Valor
0	2^{0}	1
1	2^{1}	2
2	2^{2}	4
3	2^{3}	8

4	2^4	16
5	2^5	32
6	2^6	64
7	2^7	128

Por tanto:

$(1101)_2 = 1×8 + 1×4 + 0×2 + 1×1 = 13_{10}$

3.2 Octal y hexadecimal

El octal se usaba históricamente por su relación directa con el binario: cada dígito octal equivale a 3 bits.

El hexadecimal, en cambio, es el más habitual en informática moderna porque agrupa 4 bits por símbolo, lo que lo hace más legible y compacto.

Base 2	Base 8	Base 16
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	10	8
1001	11	9
1010	12	A
1011	13	B
1100	14	C
1101	15	D

1110	16	E
1111	17	F

> **Ejemplo:**
> $(1111\ 1111)_2 = (377)_8 = (FF)_{16}$

4. Conversiones entre sistemas

4.1 De binario a decimal

Suma de potencias de 2 activas:

> **Ejemplo:**
> $(10110)_2 = 1\times16 + 0\times8 + 1\times4 + 1\times2 + 0\times1 = 22_{10}$

4.2 De decimal a binario

División sucesiva por 2, anotando los restos de abajo a arriba.

> **Ejemplo:**
> $22_{10} \div 2 \rightarrow$ restos: $0,1,1,0,1 \rightarrow (10110)_2$

División	Cociente	Resto
22 ÷ 2	11	0
11 ÷ 2	5	1
5 ÷ 2	2	1
2 ÷ 2	1	0
1 ÷ 2	0	1

4.3 De binario a octal/hexadecimal

Agrupar los bits de tres en tres (octal) o de cuatro en cuatro (hexadecimal), desde la derecha.

> **Ejemplo:**
> $(11001111)_2 = (317)_8 = (CF)_{16}$

Binario	Octal	Hexadecimal
00000000	000	00

101010	52	2A
11111111	377	FF

5. Los bits importan

La representación binaria no es solo una curiosidad teórica: condiciona toda la informática moderna.

Errores en la conversión o en los límites de representación han causado desde fallos en videojuegos hasta pérdidas millonarias.

- El cohete Ariane 5 (1996) explotó por un desbordamiento de 16 bits.
- El videojuego Pac-Man se colgaba al alcanzar el nivel 256 porque su contador solo tenía 8 bits.
- En 2014, el vídeo Gangnam Style desbordó el contador de YouTube (32 bits) al superar los 2.147 millones de visitas, por lo que Google tuvo que reescribir su sistema en 64 bits.

> **Reflexión:** comprender cómo se representan los datos es la diferencia entre usar un ordenador y entenderlo.

6. Operaciones en binario

Una vez comprendido cómo se representan los números en binario, podemos realizar operaciones aritméticas básicas siguiendo las mismas reglas que en el sistema decimal.

La diferencia es que solo existen dos símbolos (0 y 1), por lo que los mecanismos de llevar o pedir prestado se simplifican y se repiten con mayor frecuencia.

6.1. Suma binaria

La suma es la operación más elemental y, a la vez, la más importante: todas las demás operaciones (resta, multiplicación y división) pueden implementarse como una secuencia de sumas o restas.

Las cuatro combinaciones posibles son:

Operación	Resultado	Acarreo
0 + 0	0	0
0 + 1	1	0
1 + 0	1	0
1 + 1	0	1

Cuando el resultado es 10_2, se escribe 0 y se lleva un 1 a la posición siguiente.

Ese “1” es el acarreo (carry), y se suma junto con los siguientes bits.

Ejemplo 1 (suma sin acarreo):

```
   1010
+  0101
----------
  11111
```

Ejemplo 2 (suma con acarreo)

```
     111
    1011₂
 +  1101₂
 ----------
  11000₂
```

Paso	Bits sumados	Resultado	Acarreo
1	1 + 1	0	1
2	1 + 0 + 1	0	1
3	0 + 1 + 1	0	1
4	1 (acarreo final)	1	–

Curiosidad técnica:
En los procesadores, las sumas se realizan mediante circuitos llamados sumadores completos (full adders), que calculan simultáneamente el bit de resultado y el acarreo.
Cientos de millones de estos sumadores trabajan en paralelo dentro de una CPU moderna.

6.2. Resta binaria

En el sistema binario, la resta se basa en la misma lógica que en decimal, pero con solo dos dígitos.

Cuando el bit superior (minuendo) es menor que el inferior (sustraendo), se pide prestado a la posición siguiente. Ese préstamo equivale a 2 en binario.

Las combinaciones básicas son:

Operación	Resultado
0 - 0	0
1 - 0	1
1 - 1	0

0 - 1	1 (prestamos 1 del bit siguiente)

Ejemplo 3 (resta simple):

```
    110
-    11
----------
     11
```

Ejemplo 2 (resta con préstamo)

```
    1000
 -  0111
 ----------
    0001
```

6.3. Multiplicación binaria

La multiplicación se realiza aplicando el mismo principio que en decimal, pero con la ventaja de que los únicos factores posibles son 0 y 1:

Operación	Resultado
0 × 0	0
0 × 1	0
1 × 0	0
1 × 1	1

Cada fila parcial de la multiplicación se desplaza una posición a la izquierda por cada nuevo bit.

```
   101₂
×  11₂
--------
   101
+ 1010
--------
 1111₂
```

> **Nota histórica:**
> Los primeros ordenadores realizaban multiplicaciones mediante sucesivas sumas y desplazamientos. El circuito capaz de hacer esto de forma automática se llama multiplicador binario.

6.4. División binaria

La división binaria es análoga a la decimal: se resta repetidamente el divisor del dividendo y se cuentan las veces que “cabe”.

Se usa principalmente en algoritmos de desplazamiento (divisiones por 2, 4, 8…).

Operación	Equivalente decimal
Desplazar 1 bit a la derecha	÷2
Desplazar 2 bits a la derecha	÷4
Desplazar 3 bits a la derecha	÷8

> 1100_2 >> 1 → 110_2

7. Representación de números negativos (complemento a dos)

Hasta ahora hemos trabajado con números sin signo (solo positivos).

Pero un ordenador también debe representar valores negativos.

Para ello se utilizan distintos sistemas de codificación: el más extendido es el complemento a dos (C2).

7.1. Representación con bit de signo

El método más simple reserva el bit más a la izquierda para el signo:

Bit de signo	Valor
0	positivo
1	negativo

Por ejemplo, en 8 bits:

00001010_2 → +10

10001010_2 → -10

El problema de este método es que genera dos ceros distintos (00000000 y 10000000), lo cual complica las operaciones aritméticas.

Por eso se prefiere el uso de complementos.

7.2. Complemento a uno (C1)

Para obtener el complemento a uno, se invierten todos los bits: los 0 se convierten en 1 y los 1 en 0.

Valor positivo	C1 (negativo)
00001010 (+10)	11110101 (-10)

Aunque sencillo, este método también tiene dos ceros distintos (+0 y –0), por lo que fue reemplazado por el complemento a dos.

7.3. Complemento a dos (C2)

Es el sistema estándar para representar enteros con signo en la mayoría de los procesadores.

Para calcular el complemento a dos:

1. Se invierten todos los bits (como en C1).
2. Se suma 1 al resultado.

Valor	Invertido (C1)	+1 (C2)	Significado
00001010 (+10)	11110101	11110110	–10

Por tanto, 11110110_2 representa –10 en complemento a dos.

Ejemplo 7. Suma con números negativos (C2)

```
  00001010 (+10)
+  11110110 (-10)
-----------------
   00000000
```

El resultado es 0, lo que demuestra que el sistema funciona de forma coherente.

Ventaja clave:
El complemento a dos permite sumar y restar números negativos sin cambiar las reglas de la suma binaria.
El hardware no necesita distinguir si el número es positivo o negativo: basta con interpretar el bit más significativo como signo.

Ejemplo 8. Overflow en complemento a dos
En un sistema de 8 bits, el rango de valores posibles va de –128 a +127.

```
  01111111 (+127)
+ 00000001 (+1)
----------------
  10000000 (overflow → -128)
```

El resultado se “desborda” y se interpreta como –128. Este tipo de errores son la causa de muchos fallos informáticos históricos.

Curiosidad moderna*:*
El bug del Ariane 5 se debió precisamente a un error de conversión en un sistema con datos de 16 bits con signo.
El valor se desbordó al convertirse en un entero de 64 bits, provocando que el cohete perdiera el control a los 37 segundos de vuelo.

8. Representación de caracteres (ASCII y Unicode)

Hasta ahora hemos visto cómo se representan los números en el ordenador.

Sin embargo, los ordenadores también deben almacenar letras, signos de puntuación, espacios, emojis y símbolos especiales (como el símbolo del euro € o una ñ).

Para ello se utilizan los códigos de caracteres, que asignan a cada símbolo un número entero.

Ese número, a su vez, se traduce a binario para que el ordenador pueda almacenarlo y procesarlo

8.1. El código ASCII

ASCII (American Standard Code for Information Interchange) fue creado en 1963 para facilitar la comunicación entre diferentes dispositivos.

Utiliza 7 bits (un octavo bit se reservaba originalmente para control o paridad) y permite representar 128 caracteres distintos: letras mayúsculas y minúsculas, números, signos de puntuación y caracteres de control (no imprimibles, como Enter o Tabulación).

Tipo de carácter	Rango ASCII	Ejemplo
Control (no imprimibles)	0–31	Enter, ESC, Tab

Espacio	32	
Signos de puntuación y operadores aritméticos	33-47	!, “, #...
Números (0–9)	48–57	0, 1, 2…9
Letras mayúsculas (A–Z)	65–90	A, B, C…Z
Letras minúsculas (a–z)	97–122	a, b, c…z

Carácter	Código ASCII	Binario (7 bits)	Decimal
A	01000001	65	65
B	01000010	66	66
a	01100001	97	97
Espacio	00100000	32	32

> **Observa:**
> Cada carácter ocupa 1 byte.
> Por eso un documento de 1.000 caracteres ocupa aproximadamente 1 KB**.**
>
> **Ejemplo:**
> “Hola” → 72 111 108 97 → `01001000 01101111 01101100 01100001`

8.2. Limitaciones de ASCII

ASCII fue suficiente durante décadas, pero solo cubre el alfabeto inglés y algunos símbolos básicos.

No incluye letras acentuadas ni caracteres de otros idiomas.

Por ejemplo:

- “Ñ” o “á” no tienen representación en ASCII puro.
- Los idiomas asiáticos (chino, japonés, árabe, etc.) requieren miles de símbolos.

Para resolver este problema surgieron variantes como ASCII extendido (8 bits), que añadía 128 caracteres más (de 128 a 255).

Sin embargo, cada fabricante o sistema operativo definía los suyos, provocando incompatibilidades (por ejemplo, el código 164 podía representar una “ñ” en Windows pero otro símbolo en Linux)

8.3. Unicode: el código universal

Para unificar todos los sistemas, se creó Unicode, un estándar internacional que asigna un número único a cada carácter de todos los idiomas y símbolos del mundo.

Unicode no define solo los números, sino también cómo se codifican en bits, según tres variantes principales:

Codificación	Bits por carácter	Características	Ejemplo de uso
UTF-8	Variable (8–32 bits)	Compatible con ASCII, eficiente para textos en inglés.	Web, Linux
UTF-16	16 o 32 bits	Más eficiente para alfabetos con muchos símbolos.	Windows, Java
UTF-32	32 bits fijos	Cada carácter ocupa 4 bytes (mayor tamaño).	Aplicaciones de procesamiento interno

Carácter	Código	Binario	Bytes
A	U+0041	01000001	1
á	U+00E1	11000011 10100001	2
🦋	U+1F98B	11110000 10011111 10100110 10001011	4

> **Curiosidad:**
> Aunque se llama “UTF-8”, puede usar hasta 4 bytes por carácter.
> Por eso los emojis o caracteres asiáticos ocupan más espacio y, en redes sociales, cuentan más “caracteres” aunque visualmente parezcan uno solo.

8.4. ASCII dentro de Unicode

Los 128 primeros caracteres de Unicode coinciden exactamente con los de ASCII.

Esto garantiza la compatibilidad hacia atrás: un archivo en ASCII es automáticamente válido en UTF-8.

Texto: “Hola”

Carácter	Código UTF-8	Binario	Bytes
H	72	01001000	1
o	111	01101111	1

l	108	01101100	1
a	97	01100001	1

Total: 4 bytes.
Así de literal: el ordenador no guarda letras, sino secuencias de bits que corresponden a esos valores.

9. Conclusión

Comprender las operaciones binarias y los sistemas de codificación no es solo una curiosidad técnica: es **la base del pensamiento informático**.

- Cada instrucción de un procesador se traduce en operaciones binarias.
- Cada carácter, número o imagen que vemos en pantalla es una secuencia de bits codificados.
- Cada error o bug que rompe un programa tiene, en última instancia, un origen binario.

> **Idea clave:**
> Un ordenador no hace magia: hace matemáticas en binario, a una velocidad que parece mágica.

10. Representación multimedia (imagen, sonido, vídeo)

Hasta ahora hemos visto cómo el ordenador representa números, caracteres y símbolos.

Sin embargo, la información digital no se limita a datos alfanuméricos: también incluye elementos visuales y sonoros que forman parte de la experiencia humana cotidiana.

Fotografías, música, películas, videollamadas o videojuegos son ejemplos de información multimedia, es decir, que combina varios tipos de medios —texto, imagen, sonido y vídeo— en un mismo entorno digital.

El tratamiento de la información multimedia es uno de los logros más complejos de la informática moderna.

A diferencia de los datos numéricos o textuales, que se almacenan de manera discreta y compacta, las imágenes y los sonidos son fenómenos continuos.

Esto significa que, para ser procesados por un ordenador, deben convertirse primero en valores discretos, muestreados a intervalos regulares y representados mediante números binarios.

Esa conversión de lo analógico a lo digital es lo que permite que una cámara, un micrófono o una tarjeta gráfica transformen el mundo físico en datos que el procesador puede interpretar.

Desde el punto de vista técnico, el ordenador no distingue entre una nota musical y un píxel de color: ambos se reducen a secuencias de ceros y unos almacenados en memoria.

- La diferencia está en cómo se codifican esos bits y qué significado se les asigna.

- En una imagen digital, cada número representa el color y la intensidad de un punto de la pantalla.
- En un sonido digital, los números indican la amplitud de la onda sonora en distintos instantes.
- En un vídeo digital, los números se organizan en secuencias de imágenes sincronizadas con una pista de audio.

En todos los casos, la clave está en encontrar un equilibrio entre calidad y tamaño.

Cuantos más bits se dediquen a describir cada detalle, más fiel será la representación… pero también mayor será el espacio ocupado y el tiempo de transmisión necesario.

Por eso existen sistemas de compresión, muestreo y codificación que optimizan la cantidad de datos sin que el usuario perciba una pérdida significativa de calidad.

Comprender cómo se representan las imágenes, los sonidos y los vídeos en formato digital es esencial para todo desarrollador, diseñador o técnico de sistemas, ya que determina:

- el peso de los archivos que manejan las aplicaciones,
- la velocidad de transmisión de datos en redes,
- la calidad visual o sonora percibida por el usuario, y
- la compatibilidad entre distintos formatos o dispositivos.

En los siguientes apartados analizaremos de forma detallada cómo se lleva a cabo esta representación en los tres tipos de medios más habituales: imágenes digitales, sonido digital y vídeo digital.

Veremos cómo se codifican los colores, cómo se convierte una onda sonora en números y cómo un conjunto de fotogramas se transforma en movimiento continuo.

10.1. La imagen digital

Las imágenes son, probablemente, la forma de información más intuitiva para el ser humano… y una de las más costosas de manejar para un ordenador.
Cuando miramos una fotografía, nuestro cerebro procesa formas, colores, profundidad, contraste; pero para un ordenador, todo eso debe traducirse a números. Cada punto de la imagen se convierte en un valor digital que representa su color y su brillo.
Esa es la esencia de una imagen digital: una colección ordenada de millones de números.

La estructura básica: la matriz de píxeles

Una imagen digital no es más que una matriz bidimensional de píxeles (picture elements).
Cada píxel ocupa una posición exacta dentro de esa cuadrícula —por ejemplo, fila 120, columna 640— y contiene la información necesaria para determinar qué color debe mostrarse en ese punto.

El tamaño de la imagen viene determinado por su resolución, expresada como:

Ancho × Alto = número total de píxeles.

Por ejemplo, una imagen de 1920×1080 píxeles (resolución Full HD) contiene

1920 × 1080 = 2 073 600 píxeles, es decir, algo más de dos millones de puntos de color.

Cuantos más píxeles haya, mayor detalle tendrá la imagen. Por eso decimos que una cámara “de 12 megapíxeles” puede capturar 12 millones de puntos distintos.
Sin embargo, más píxeles no siempre implican mejor calidad, porque influyen otros factores como el tamaño físico del sensor o la óptica. En informática, interesa entender la resolución como una medida de cantidad de datos, no necesariamente de nitidez visual.

> **Ejemplo:**
> Una imagen HD (1280×720) contiene 921 600 píxeles.
> Una imagen 4K (3840×2160) tiene 8 294 400 píxeles.

La segunda no tiene el doble de píxeles, sino nueve veces más.
Por eso los vídeos 4K pesan tanto: cada fotograma tiene nueve veces más datos que uno HD.

La profundidad de color

El siguiente parámetro fundamental es la profundidad de color, que indica cuántos bits se usan para codificar el color de cada píxel.
Con 1 bit por píxel solo pueden representarse dos colores (0 = negro, 1 = blanco).
A medida que se añaden bits, crece exponencialmente el número de tonos posibles.

Profundidad	Colores posibles	Descripción
1 bit	2	Blanco y negro puros
4 bits	16	Primeros ordenadores y consolas (paletas reducidas)
8 bits	256	Escala de grises o paletas básicas
16 bits	65 536	Alta fidelidad (“High Color”)
24 bits	16,7 millones	Color real o “True Color”
32 bits	16,7 millones + canal α	Transparencias y efectos

Con 24 bits se logran 256 niveles para cada componente primaria del color (rojo, verde y azul), lo que da 256^3 = 16 777 216 combinaciones distintas.
Ese es el estándar actual en pantallas y cámaras.

Modelos de color: RGB y CMYK

Para describir el color de un píxel existen distintos modelos de color.

Los dos más comunes son:

- RGB (Red-Green-Blue): modelo aditivo utilizado en pantallas, proyectores y cámaras.
  Cada canal se suma para generar colores más claros; cuando los tres alcanzan su valor máximo, el resultado es blanco.
- CMYK (Cyan-Magenta-Yellow-Key/Black): modelo sustractivo utilizado en impresión.
  Parte del blanco del papel y resta luz mediante tintas; cuando los tres pigmentos se combinan, se obtiene un tono oscuro (que se refuerza con el negro K).

Modelo	Medio	Principio	Ejemplo
RGB	Monitores, cámaras, móviles	Aditivo (sumar luz)	Rojo 255, Verde 0, Azul 0 → Rojo puro
CMYK	Impresión	Sustractivo (restar luz)	Cian 100, Magenta 100, Amarillo 0, Negro 0 → Violeta

> **Curiosidad:**
> Las discusiones entre diseñadores y programadores sobre “por qué este azul no se imprime igual que en pantalla” se deben a esta diferencia entre los modelos. RGB y CMYK no son equivalente, hay colores que pueden mostrarse en pantalla pero son imposibles de reproducir en tinta.

Ejemplo de cálculo: tamaño de archivo

Supongamos una fotografía de 1024×768 píxeles, codificada en 24 bits (3 bytes) por píxel.

[1024 × 768 × 3 = 2 359 296 bytes ≈ 2,25 MB]

Si esa misma imagen fuera en escala de grises (8 bits), ocuparía:

[1024 × 768 × 1 = 786 432 bytes ≈ 0,75 MB]

Y si fuera en blanco y negro (1 bit):

[1024 × 768 ÷ 8 = 98 304 bytes ≈ 96 KB]

> **Conclusión**:
> Reducir la profundidad de color o la resolución disminuye drásticamente el tamaño, pero también la calidad visual.
> Por eso las miniaturas y los iconos se guardan en resoluciones pequeñas y con menos bits por píxel.

La compresión: ahorrar espacio

Una imagen sin comprimir almacena la información de cada píxel de forma literal.

Sin embargo, eso genera archivos enormes, por lo que se aplican algoritmos de compresión, que pueden ser:

Tipo	Descripción	Ejemplo
Sin pérdida	El archivo puede descomprimirse y recuperarse exactamente igual al original.	PNG, BMP (RLE), TIFF
Con pérdida	Se eliminan datos redundantes o poco perceptibles para reducir el tamaño.	JPEG, WEBP

Compresión sin pérdida: ideal para logotipos o gráficos con pocos colores, donde cada píxel importa.

Compresión con pérdida: óptima para fotografías, donde pequeñas variaciones son imperceptibles.

> **Advertencia:**
> Cada vez que guardas una imagen JPEG se vuelve a comprimir.
> Repetir el proceso varias veces degrada la calidad, generando los típicos “bloques” o artefactos en zonas uniformes.
>
> **Ejemplo práctico:**
> Una foto en BMP (2,3 MB) puede comprimirse a JPEG de 200 KB manteniendo una calidad visual casi idéntica.
> En cambio, un logotipo con líneas nítidas se vería borroso si se guardara como JPEG; en ese caso conviene PNG

Metadatos y cabeceras

Además de los píxeles, un archivo de imagen incluye información adicional llamada cabecera o metadatos, donde se almacena la resolución, el modo de color, la compresión usada, la fecha o incluso la ubicación GPS en fotografías.

Ejemplo de campos típicos en metadatos EXIF de una foto:

- Cámara: Canon EOS 200D
- Resolución: 6000×4000 píxeles
- ISO: 200
- Tiempo de exposición: 1/125 s
- Apertura: f/5.6
- Localización: 41.655 N – 0.875 W

> **Curiosidad:**
> Muchos programas eliminan estos datos al exportar imágenes “optimizadas” para web, reduciendo algunos kilobytes y mejorando la privacidad (evita mostrar la ubicación).

Representación binaria interna

Dentro de la memoria del ordenador, los píxeles se almacenan de forma secuencial. Por ejemplo, un píxel con color RGB (128, 64, 255) se representa como:

Canal	Valor decimal	Valor binario
Rojo	128	10000000
Verde	64	01000000
Azul	255	11111111

Juntos forman 24 bits: 10000000 01000000 11111111

Cuando el sistema muestra la imagen, cada grupo de 24 bits se traduce de nuevo a un color en pantalla.

> **Cálculo mental útil:**
> Si una pantalla Full HD (1920×1080) muestra 24 bits por píxel, el número total de bits en pantalla en un solo instante es:
> 1920×1080×24 = 49 766 400 bits ≈ 6 MB.

Es decir, cada fotograma ocupa 6 MB de memoria; por eso los vídeos sin compresión son tan pesados.

Errores comunes y aliasing

La digitalización de imágenes introduce errores si la resolución es insuficiente.

El fenómeno conocido como aliasing aparece cuando los bordes oblicuos se ven “dentados” o cuando los patrones finos producen efectos de moiré.

Para mitigarlo se utilizan técnicas de antialiasing, que suavizan los bordes mediante promedios de color.

> **Analogía:**
> Si dibujas una línea diagonal con piezas de LEGO, verás escalones.
> Cuantas más piezas uses, más suave parece la línea.
> El ordenador hace lo mismo aumentando la resolución o mezclando tonos intermedios.

Formatos modernos y transparencia

El formato PNG añadió un cuarto canal: el canal α (alfa), que representa la transparencia.

Cada píxel no solo tiene valores RGB, sino también un valor entre 0 y 255 que indica su opacidad.

Esto permite superponer imágenes sin fondo, una función esencial en diseño web y videojuegos.

Canal	R	G	B	A
Valor	255	0	0	128

En este ejemplo, el píxel sería rojo al 50 % de transparencia.

10.2.

8.2 Sonido digital

El sonido se representa mediante muestras (samples) tomadas a intervalos regulares.

- **Frecuencia de muestreo:** nº de muestras por segundo (Hz).
  Ejemplo: CD de audio = 44.100 Hz.
- **Profundidad de bits:** nº de bits por muestra.
  Ejemplo: 16 bits = 65.536 niveles posibles.

> **Ejemplo de cálculo:**
> Estéreo, 44.100 Hz, 16 bits →
> 44.100 × 16 × 2 = 1.411.200 bit/s ≈ **176 KB/s**

8.3 Vídeo digital

El vídeo no es más que una secuencia de imágenes (fotogramas) con sonido sincronizado.

Sus factores clave son:

- **Resolución:** número de píxeles por fotograma.
- **FPS:** fotogramas por segundo (30, 60…).
- **Bitrate:** cantidad de datos por segundo.

> **Ejemplo:**
> Vídeo Full HD (1920×1080), 30 fps, 24 bits:
> 1920×1080×3×30 ≈ 186 MB/s (sin compresión).
> Por eso se usan códecs como H.264 o HEVC.

11. Precisión, redondeo y errores

Los ordenadores tienen precisión finita: solo pueden almacenar un número limitado de dígitos.

Cuando un resultado supera esa capacidad, se produce un overflow (desbordamiento) o underflow.

9.1 Redondeo

Ejemplo clásico:

En una hoja de cálculo, (0.1 + 0.2) puede dar 0.30000000000000004.

Esto ocurre porque 0.1 y 0.2 no tienen representación exacta en binario.

> **Advertencia:**
> Los números de punto flotante son aproximaciones. En cálculos críticos (financieros, científicos) se utilizan librerías de precisión arbitraria o tipos decimales

12. Resumen y aplicación práctica

Toda la información que maneja un ordenador —números, texto, imágenes o sonido— se reduce a secuencias de bits.
Los sistemas de numeración nos permiten entender y manipular esa codificación, mientras que los métodos de representación (ASCII, Unicode, RGB, PCM…) nos acercan al modo en que la máquina traduce el mundo real.

> **Aplicación práctica:**
> En un archivo `.txt` de 1000 caracteres, cada carácter ocupa 1 byte → 1 KB.
> Si insertamos una imagen `.jpg` de 2 MB, ese mismo documento pasa a ocupar 2000 veces más, aunque su contenido “visible” sea solo una foto.