Imágenes, píxeles y colores

Eduardo Graells-Garrido

16 de marzo de 2026

1 ¿Qué es el color?

Figura 1. El arte de Yoshitaka Amano (en este caso, para *Final Fantasy VI*) se caracteriza por utilizar colores intensos y de contrastes marcados. El color le da expresividad y significado a su arte.

En esta unidad hablaremos del resultado final de cualquier proceso de Computación Gráfica: la imagen. En nuestro caso, las imágenes están compuestas de píxeles, que, a su vez, contienen un color cada uno. ¿Qué es una imagen? ¿Qué es un color? ¿Qué es un píxel? En el arte, el color es primordial para la expresividad. La imagen que inaugura esta sección es una obra de Yoshitaka Amano para el videojuego *Final Fantasy VI* (Figura 1). Cada vez que veo sus obras me sorprende el uso de colores vivos y formas orgánicas, algo caóticas también. Hoy, en tiempos en que la Inteligencia Artificial genera imágenes de mucho colorido pero poca expresividad, me pregunto cuánto de humano hay en el uso del color. Y, en el contexto del curso, nos preguntamos: ¿Cómo dialoga la importancia del color con los aspectos técnicos necesarios para representar una imagen en pantalla?

Wikipedia nos puede dar una respuesta enciclopédica:

> *El color es la impresión producida por un tono de luz en los órganos visuales, o más concretamente, es una percepción visual que se genera en el cerebro, –específicamente en el lóbulo occipital, corteza visual– de los humanos y otros animales al interpretar las señales nerviosas que le envían los fotorreceptores en la retina del ojo, que a su vez interpretan y distinguen las distintas longitudes de onda que captan de la parte visible del espectro electromagnético. Es estudiado por la colorimetría o ciencia del color.*

Esta definición nos da un punto de partida, aunque no parece tan cercana a la computación, ¿no es así? Podemos estudiarla por partes para entenderla y acercarla a CG.

Primero, habla de *impresión producida*. El color es parte de como vemos el mundo, y no solo nosotres, también otros seres vivos. Las aves lo utilizan para identificarse y elegirse: un bello plumaje es señal de buen estado de salud y de buena genética (Figura 2). Es curioso cómo la evolución hizo que el color adquiriera tal importancia en sus vidas.

Figura 2. Las plumas de los picaflores son iridiscentes, es decir, varían la luz que reflejan dependiendo del ángulo en que se vea. Fuente: WikiCommons.

Después la definición habla de un *tono de luz*. En efecto, cuando vemos una pantalla, estamos viendo luz. La luz que emiten las microampolletas de una pantalla les permiten leer esto. Ahora bien, todes tenemos una noción de cómo es el color azul, pero diferentes dispositivos muestran diferentes azules, incluso para la misma imagen (Figura 3). ¿Por qué pasa esto? Un factor importante es que cada máquina utiliza *hardware* distinto. Puede ser el mismo tipo de tecnología, pero con piezas de distinto origen y calidad, y sin una arquitectura idéntica. A nivel de *software* tienen configuraciones de fábrica distintas. En esta unidad hablaremos de estas diferencias.

Figura 3. Filas de televisores, como en los malls. Si todos muestran la misma imagen, ¿por qué tienen colores diferentes? Fuente: CMU.

Un aspecto fascinante de la percepción del color es que depende del contexto. El famoso debate sobre "el vestido amarillo/azul" que dividió internet en 2015 (Figura 4) ilustra este fenómeno: una misma imagen fue percibida como azul y negro por algunas personas, mientras otras la veían blanca y dorada. ¿De cuál color lo ves tú?

Figura 4. ¿Cuál es el color del vestido?

Una de las explicaciones para las diferentes interpretaciones es la "constancia del color", un mecanismo por el cual nuestro cerebro intenta mantener constante el color percibido de un objeto a pesar de las variaciones en la iluminación. La imagen no proporciona suficiente contexto sobre las condiciones de iluminación, por lo que cada cerebro hace suposiciones diferentes, resultando en percepciones distintas.

Un ejemplo notable en esa dirección es la ilusión de las frutillas de Akiyoshi Kitaoka (Figura 5). En esa imagen, aunque parezca que las frutillas son rojas, la imagen no contiene píxeles rojos: el cerebro "completa" el color rojo basándose en su cono-

cimiento previo de cómo lucen las fresas y en el contexto cromático circundante.

Estos ejemplos muestran que la percepción del color no es solo una medición objetiva de la realidad física, sino una interpretación construida por nuestro cerebro para facilitar la interacción con el entorno. Esta comprensión de la naturaleza subjetiva del color es necesaria para el diseño de sistemas gráficos que buscan simular o reproducir la experiencia visual humana.

Figura 5. ¿De qué color son las frutillas? La respuesta te sorprenderá. Fuente: Akiyoshi Kitaoka.

Luz y color

La unidad comenzó resaltando la importancia de la expresividad del color. Pero también la luz ha sido importante en la expresión, y está vinculada al concepto de color. Veamos qué dice sobre la luz el siguiente poema[1]:

> Hay una cierta inclinación de la luz
> en las tardes de invierno,
> que nos oprime, como el peso
> de las melodías de una catedral.
>
> Nos inflinge una herida celestial;
> que no deja marcas,
> solo la interna constancia
> donde los significados están.
>
> Nadie puede enseñarle nada,
> porque es el sello, la desesperanza:
> una aflicción imperial
> que nos envía el aire.
>
> Cuando llega, el paisaje escucha;
> las sombras contienen el aliento;
> cuando se aleja, es como la distancia
> en la mirada de la muerte.

Hay una cierta inclinación de la luz (1890),
Emily Dickinson
Traducción por El Espejo Gótico.

[1] ¿Por qué arte, por qué un poema en un curso de Computación Gráfica? Estudiamos computación para modelar el mundo y tener un impacto positivo sobre él. Solo podremos encontrar respuestas a los problemas que se nos presenten si sabemos observar, escuchar y sentir.

El poema no solo habla de cómo la luz (o su falta) incide en lo que siente su autora, también nos muestra que la luz tiene significados. En el lenguaje, la luz también se asocia al conocimiento y a encontrar el camino.

Desde un punto de vista físico, la luz es radiación electromagnética oscilante (Figura 6). Para entender qué significa esto, imaginemos una cuerda tensa que alguien agita en un extremo: se forma una onda que viaja a lo largo de la cuerda, con crestas y valles que se repiten a intervalos regulares. La distancia entre

dos crestas consecutivas es la *longitud de onda* (λ), y la cantidad de crestas que pasan por un punto por segundo es la *frecuencia* (f). Ambas magnitudes están relacionadas por la velocidad de propagación: en el vacío, la luz siempre viaja a $c \approx 3 \times 10^8$ m/s, de modo que $\lambda = c/f$.

La diferencia con una onda en una cuerda es que la luz no necesita un medio físico para propagarse: lo que oscila son campos eléctrico y magnético perpendiculares entre sí y perpendiculares a la dirección de viaje. El color que percibimos es precisamente esa frecuencia de oscilación: la luz violeta oscila más rápido (longitud de onda corta, ~400 nm) que la luz roja (longitud de onda larga, ~700 nm).

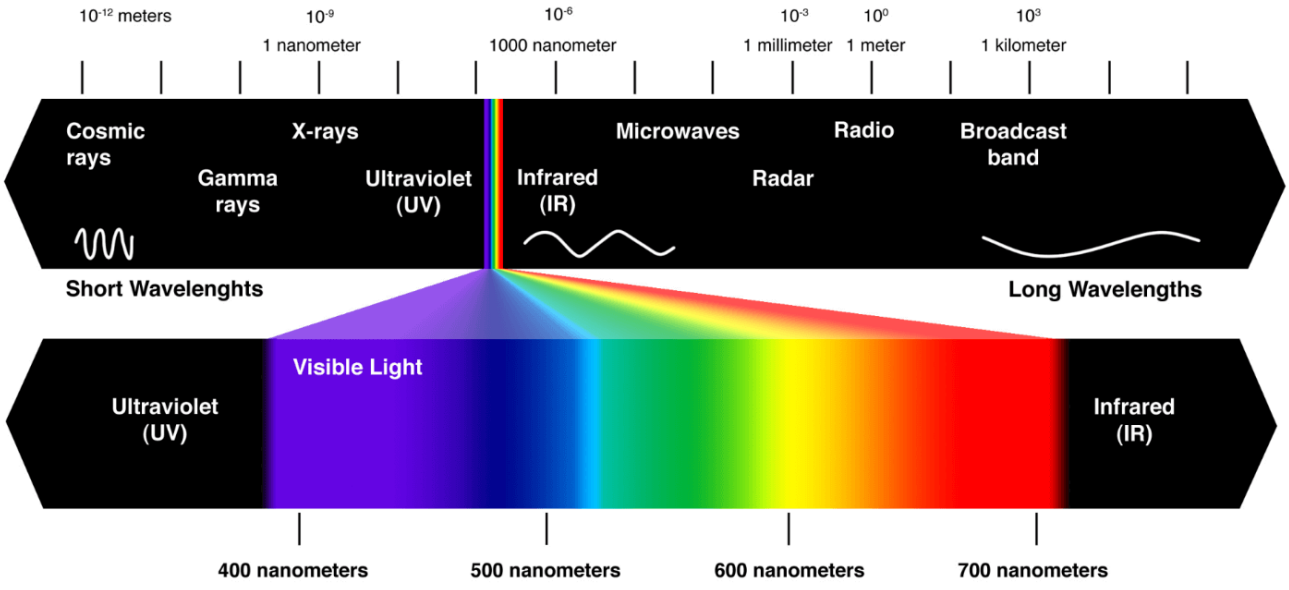


Figura 6. La luz es radiación electromagnética oscilante. La frecuencia en la que oscila determina su color.

El espectro visible humano abarca longitudes de onda entre 400 nm (violeta) y 700 nm (rojo); todo lo demás (infrarrojo, ultravioleta, microondas) existe, pero nuestros ojos no lo detectan. Este rango no es universal: varía entre especies según sus necesidades evolutivas. Las aves, por ejemplo, deben reconocer sus propios huevos entre los de otras especies, identificar a sus crías y encontrar alimento en entornos muy variados, todo desde el aire y en distintas condiciones de iluminación. Esas presiones evolutivas derivaron en una ventaja notable: las aves pueden ver luz ultravioleta (Figura 8), lo que amplía considerablemente su capacidad para distinguir plumajes, frutos maduros y señales que para nosotres son invisibles (Figura 7).

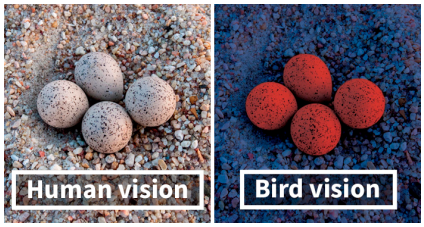


Figura 7. A la izquierda: cómo vemos un nido. A la derecha: cómo las aves ven ese nido. Fuente: BoredPanda.

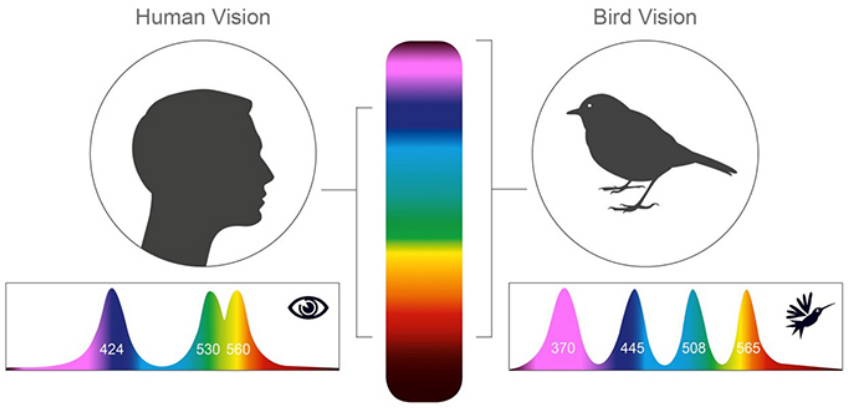


Figura 8. El espectro visible de humanos y aves es diferente. No solo vemos distintos colores, la gama del espectro accesible a las aves es mayor. ¡Hay colores que ni siquiera podemos imaginar!

Las máquinas, si están equipadas con sensores adecuados, pueden ver más que eso: rayos X, infrarojo, ondas de radio. Las imágenes satelitales no son solo fotos tal como las vemos nosotres, sino que capturan otros lugares del espectro que están relacionados con atributos biológicos y químicos de cada lugar (Figura 9). Por eso son un apoyo importante en estudios de agricultura y de escasez hídrica, de contaminación, distribución de población, entre otros temas.

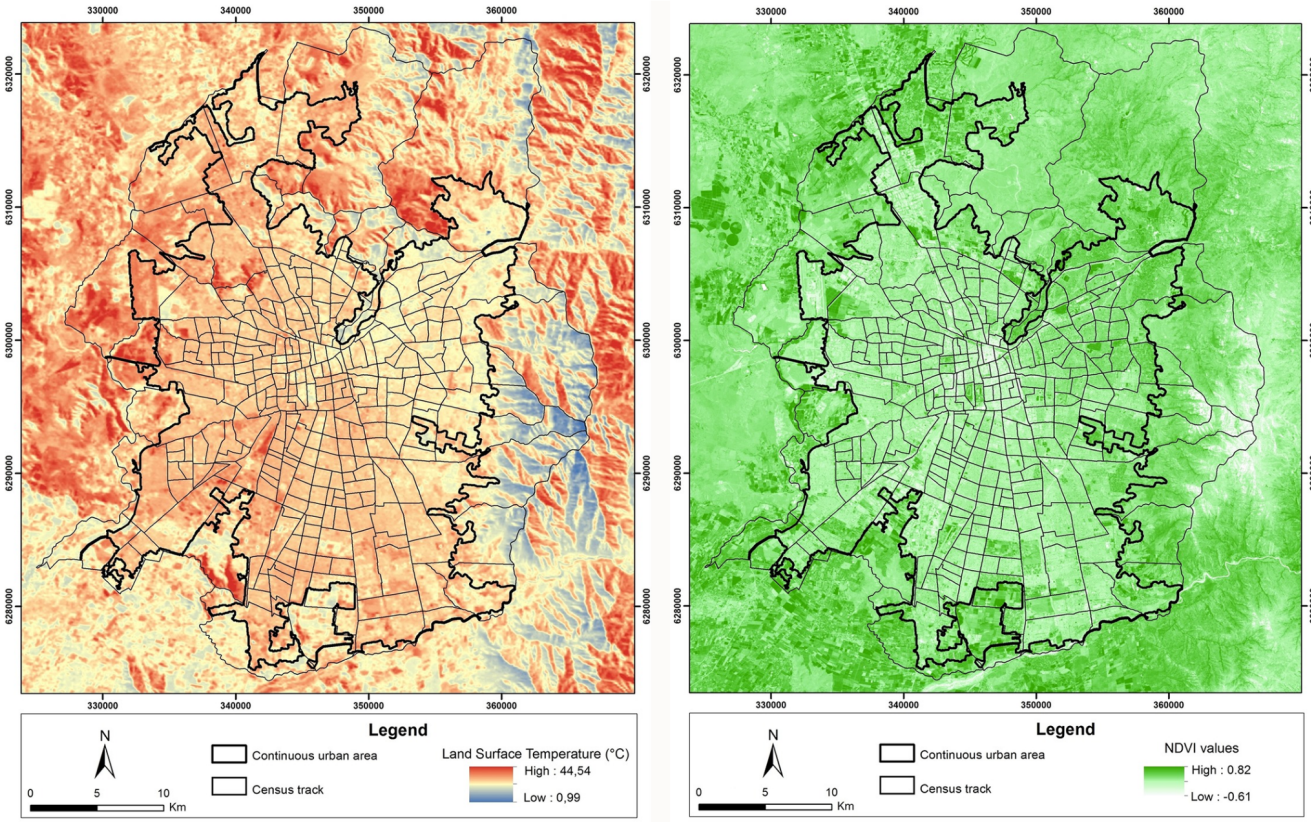


Figura 9. ¡Las bandas que no observamos se pueden estudiar! Por ej., en las imágenes satelitales se utilizan para determinar la temperatura y el índice de vegetación de un lugar (de un pixel de la imagen). Fuente: Referencia [1].

El color no es una propiedad estática de los objetos. Por ejemplo, ¿por qué se pone rojo el neón cuando se calienta (Figura 10)? ¿Por qué es posible la vista infrarroja, como en *Depredador* (Figura 11)?

Figura 10. Una de las tantas maneras en que se produce luz es el calor.

Figura 11. El calor puede verse (si se tiene un sensor con la sensibilidad adecuada para el espectro). Esa característica es utilizada como recurso narrativo en la película *Depredador*.

Según la teoría electromagnética de Maxwell[2], el movimiento de partículas cargadas genera campos electromagnéticos. Por otro lado, la termodinámica establece que las partículas de la materia están en constante movimiento, cuya intensidad depende de la temperatura. Esta conexión es fundamental: todo objeto con temperatura emite radiación electromagnética (luz), cuya frecuencia (y por tanto, su color) está determinada por su temperatura. Cada objeto a nuestro alrededor emite su propio color, aunque no siempre sea visible para nosotros.

[2] James Clerk Maxwell formuló la teoría clásica de radiación electromagnética que unificó electricidad, magnetismo y luz como manifestaciones distintas de un mismo fenómeno.

¿Cómo interactúa la luz con los objetos de una escena?

¿Podemos decir con seguridad que una pelota es blanca? Pregúntate lo siguiente: ¿De qué color ves la pelota blanca bajo una luz azul? Para entender la interacción entre luz y objetos, utilizamos dos modelos complementarios: el modelo aditivo, que describe cómo se combinan diferentes fuentes de luz en el espacio, y el modelo sustractivo, que explica cómo los materiales absorben o reflejan ciertas longitudes de onda. Estos modelos no clasifican a los objetos, sino que describen los fenómenos físicos que ocurren cuando la luz encuentra superficies en una escena.

Cada color tiene una "firma" característica, representada como la variación de su identidad según el espectro de frecuencias (Figura 12). Cuando la luz interactúa con un objeto, ocurren dos procesos (Figura 13). Por un lado, la función $f(v)$ representa la distribución espectral de la luz incidente, donde v es la longitud de onda medida en nanómetros (nm). Esta función muestra la intensidad de cada longitud de onda presente en la fuente de luz.

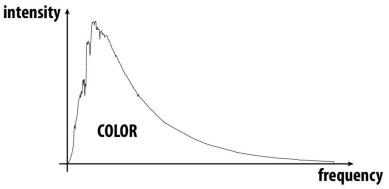


Figura 12. El color que percibimos puede entenderse como una distribución de intensidad a lo largo del espectro de frecuencias. Fuente: CMU.

Por otro lado, $g(v)$ representa la reflectancia del objeto para cada longitud de onda, indicando qué porcentaje de luz es reflejada en cada frecuencia del espectro. El color resulta del producto de ambas funciones: $f(v)g(v)$, que muestra qué frecuencias sobreviven después de la interacción luz-objeto.

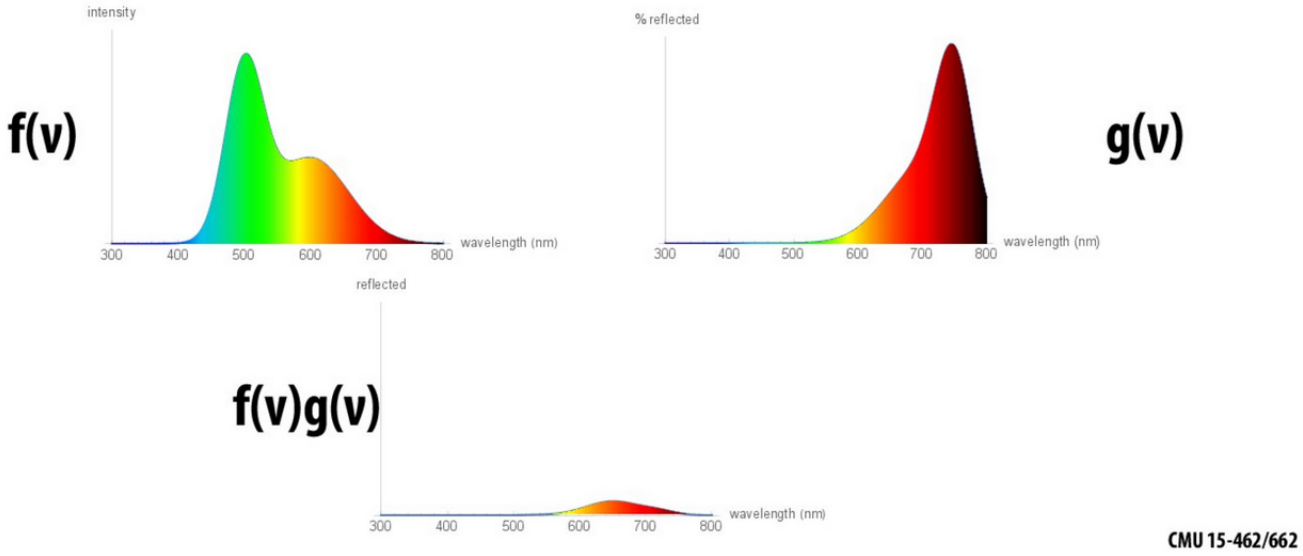


Figura 13. El color resultante de la interacción luz-objeto es el producto de dos funciones de interacción: una aditiva (relacionada con la emisión de luz) y una sustractiva (relacionada con la reflexión de luz).

Recordemos la siguiente parte de la definición: *una percepción visual que se genera en el cerebro*. Una cosa es el color físico de la superficie (el resultado del producto de funciones); una muy distinta es el color que percibes. Nuestros ojos captan esta luz reflejada y envían señales al cerebro, que son interpretadas en el color que vemos. Por tanto, el color no es una propiedad intrínseca del objeto, sino el resultado de esta interacción tridireccional entre la fuente de luz, el objeto y nuestro sistema visual (Figura 14).

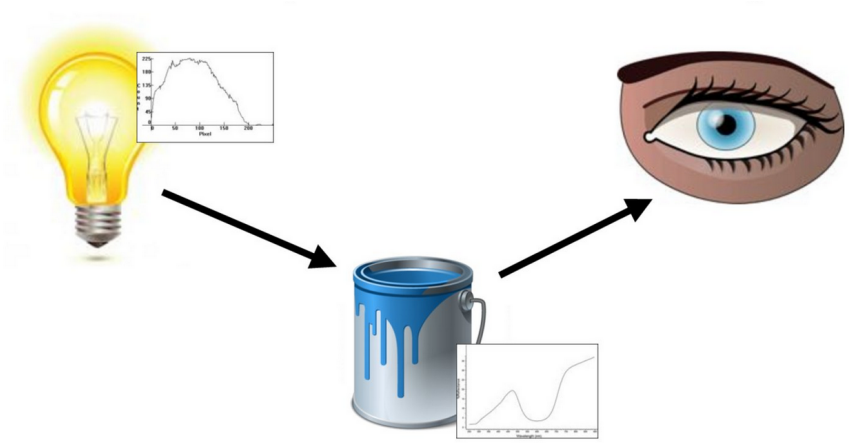


Figura 14. Cuando iluminamos un tarro de pintura azul, la luz emitida por la ampolleta (con su distribución espectral característica) interactúa con la pintura, que absorbe ciertas frecuencias y refleja otras. Nuestros ojos capta esta luz reflejada y envía señales al cerebro, que interpreta la distribución resultante como "azul". Por eso el color que eliges para pintar tu habitación en la ferretería no luce igual cuando lo aplicas en tus paredes.

La percepción del color comienza en el ojo humano[3] (Figura 15). La luz que llega a nuestros ojos atraviesa la córnea y el cristalino antes de llegar a la retina. En esta capa se encuentran los fotorreceptores: aproximadamente 120 millones de bastones (responsables de la visión en condiciones de baja luminosidad) y entre 6 y 7 millones de conos (responsables de nuestra percepción cromática). Estos fotorreceptores transforman la energía lumínica en señales nerviosas que viajan al cerebro a través del nervio óptico.

[3]Puedes aprender más sobre el ojo en Wikipedia: https://es.wikipedia.org/wiki/Ojo_humano.

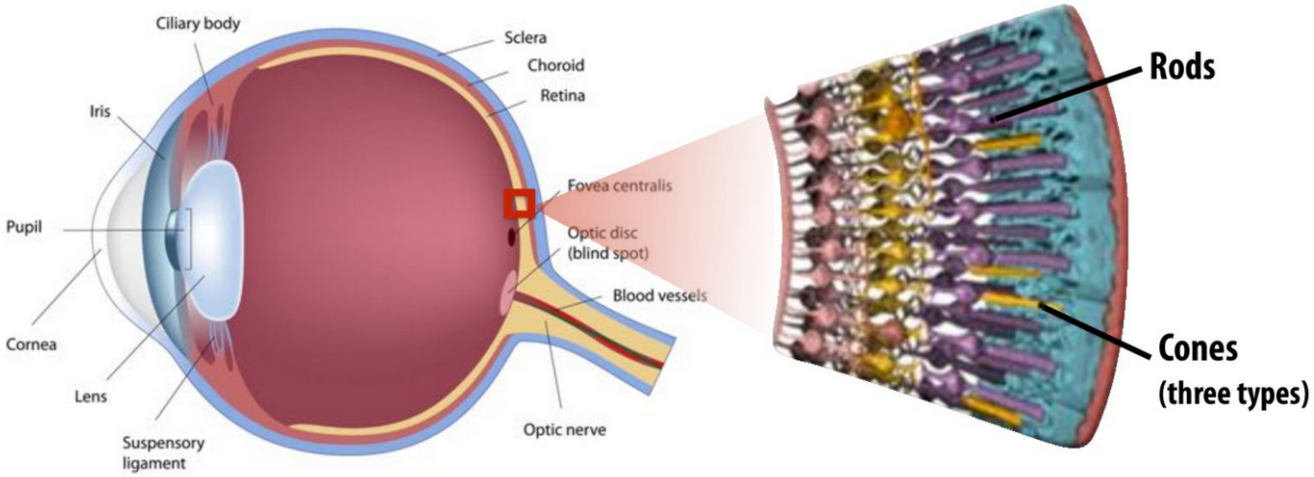


Figura 15. Nuestro ojo posee una arquitectura compleja. La luz atraviesa la córnea y el cristalino antes de llegar a la retina. Fuente: CMU.

Los conos se dividen en tres tipos, cada uno sensible a diferentes regiones del espectro visible: conos S (sensibles a longitudes de onda cortas, percibidas como azul), conos M (sensibles a longitudes de onda medias, percibidas como verde) y conos L (sensibles a longitudes de onda largas, percibidas como rojo). Estos no están distribuidos de manera uniforme en la retina (cerca del 64% son conos L y el 32% son conos M, dejando solo un 4% de conos S). Esta distribución desigual tiene implicaciones importantes en nuestra sensibilidad a diferentes colores, como veremos más adelante en la unidad de Visualización Científica[4].

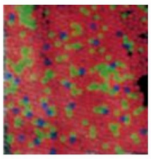
Cada tipo de cono responde a diferentes longitudes de onda de la luz (Figura 16). Los conos S tienen su techo de respuesta alrededor de los 445 nm (azul), los conos M alrededor de 535 nm (verde), y los conos L cerca de 565 nm (amarillo-rojo). Matemáticamente, la respuesta de cada tipo de cono puede expresarse mediante integrales que combinan la distribución espectral de la luz incidente $\Phi(\lambda)$ con la función de sensibilidad de cada tipo de cono ($S()$, $M(\lambda)$, $L(\lambda)$).

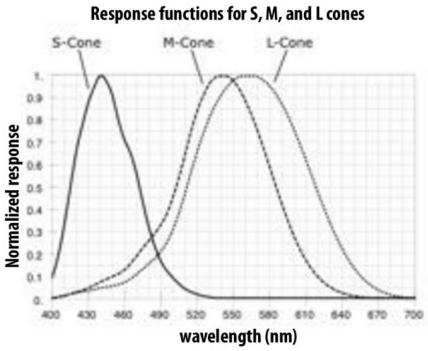
[4]En esta área se grafican fenómenos que no son visibles al ojo humano, lo que implica asignarles colores para que alguien interprete el fenómeno y tome una decisión. Pero, ¿por qué asumir que esa persona puede ver todos los colores? Esta pregunta también tiene relación con la percepción, particularmente con la "ceguera de colores", una condición importante que estudiaremos al final del curso.

$$S = \int_{\lambda} \Phi(\lambda) S(\lambda) d\lambda$$

$$M = \int_{\lambda} \Phi(\lambda) M(\lambda) d\lambda$$

$$L = \int_{\lambda} \Phi(\lambda) L(\lambda) d\lambda$$





Uneven distribution of cone types in eye
~64% of cones are L cones, ~ 32% M cones

CMU 15-462/662

Figura 16. Respuesta espectral por tipo de fotorreceptor. Fuente: CMU.

La teoría tricromática de la visión[5] explica cómo la combinación de señales de estos tres tipos de conos es suficiente para percibir todo el espectro de colores visibles. Esta base biológica es el fundamento sobre el cual nuestro cerebro construye la compleja experiencia subjetiva del color, incluyendo fenómenos de constancia del color e ilusiones cromáticas como las que vimos anteriormente.

[5] Puedes ver su descripción en Wikipedia. Fue propuesta por Thomas Young, desarrollada por Hermann von Helmholtz y demostrada por Maxwell.

2 Modelos de color

Hemos explorado cómo la luz interactúa con los objetos y cómo nuestro sistema visual procesa esas señales, pero en Computación Gráfica necesitamos representar estos fenómenos complejos de manera que las máquinas puedan trabajar con ellos [3]. ¿Cómo traducimos la riqueza continua del espectro electromagnético a valores discretos que una computadora pueda manipular?

Los modelos de color son abstracciones matemáticas que permiten representar colores con precisión y realizar operaciones entre ellos. Aunque existen decenas de modelos especializados para diferentes aplicaciones (fotografía, impresión, televisión, colorimetría científica), nos enfocaremos en tres modelos emblemáticos: RGB (aditivo), HSV (perceptual) y CMYK (sustractivo).

RGB: El color como adición de luz

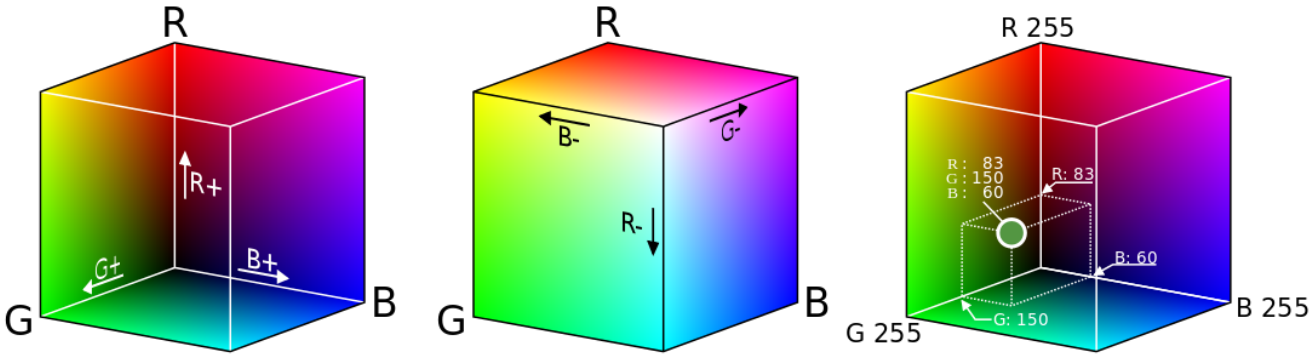


Figura 17. RGB se suele representar como un cubo.

El modelo RGB es aditivo: parte de la ausencia de luz (negro) y añade cantidades variables de los tres colores primarios de luz: rojo (R), verde (G) y azul (B). ¡No es coincidencia que sean los tres colores de la teoría tricromática de la visión! Las pantallas funcionan en base a esta aproximación: cada píxel contiene tres diminutas fuentes de luz correspondientes a estos colores primarios [4].

Un color se representa como un vector (r, g, b) donde cada componente indica la intensidad de su correspondiente color primario. Estos valores se normalizan entre 0 y 1 (para cálculos en punto flotante) o entre 0 y 255 (para almacenamiento en enteros de 8 bits). Así, es un espacio euclidiano con forma de cubo (Figura 17). La diagonal desde $(0, 0, 0)$ a $(1, 1, 1)$ representa la escala de grises; y sus esquinas; "colores puros": negro, rojo, verde, azul, cian, magenta, amarillo y blanco.

Ahora bien, la distancia euclidiana entre dos colores no necesariamente corresponde a la diferencia perceptual entre esos colores. Por eso, si bien este modelo es idóneo para Computación Gráfica, no lo es para las personas.

HSV: Un modelo interpretable

Mientras RGB es excelente para representación computacional, no resulta intuitivo para la selección de colores por parte de una persona. Por ejemplo, ¿cómo saber qué valores RGB producirán un "naranja apagado"? El modelo HSV (*Hue*, *Saturation*, *Value*) reorganiza el espacio de color RGB para alinearlo con nuestra forma de pensar sobre el color: hay un **matiz** (H), el tipo de color, representado como un ángulo en la rueda cromática (0° a 360°); **saturación** (S), la pureza o intensidad del color (0 a 1); y un **valor** (V), el brillo (0 a 1).

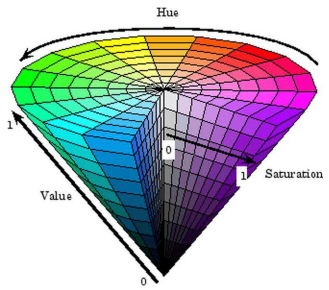


Figura 18. El cono HSV.

HSV se representa como un cono (Figura 18). Visto desde arriba, se observa una “rueda de color”. A pesar de su apariencia, es una transformación de RGB, por tanto, no es un modelo con colores distintos, sino una reorganización del espacio cromático. El color se codifica relativo a la diagonal del cubo RGB: el matiz se representa como un ángulo alrededor de este eje, la saturación como la distancia desde este eje central, y el valor (o brillo) como la altura en este espacio. Esta transformación geométrica conserva toda la información del espacio RGB original, pero la estructura de manera que resulta mucho más intuitiva para seleccionar y modificar colores.

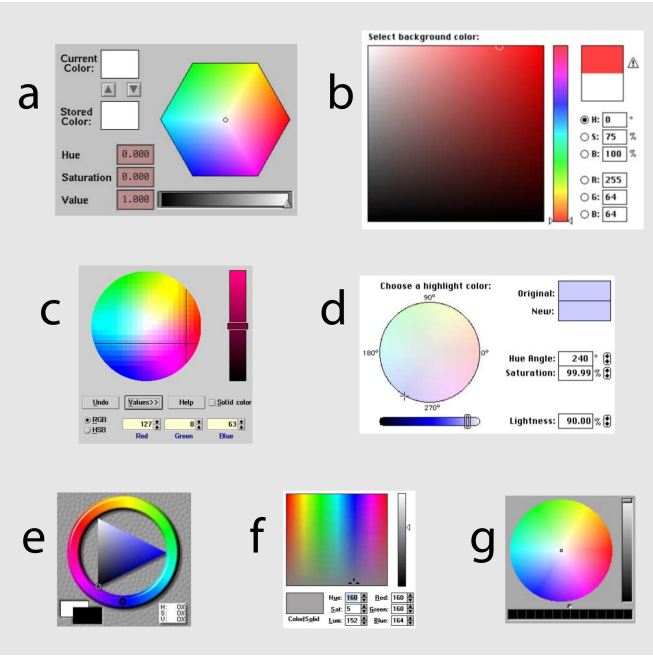


Figura 19. La historia del diseño de interfaces para selección de color nos muestra la evolución hacia estos modelos más intuitivos. Desde los años 90, prácticamente todo *software* de diseño ha incorporado *color pickers* o selectores basados en HSV o su variante HSL. Fuente: Wikipedia.

Que ambos modelos sean equivalentes permite que se puedan elegir colores de manera intuitiva (en HSV, Figura 19) para luego trabajar con ellos de manera eficiente (RGB). En este aspecto, existen herramientas modernas como Adobe Color, que facilitan no solo la selección de colores individuales sino la creación de paletas armónicas.

Las fórmulas para la conversión bidireccional entre estos espacios, aunque complejas, son fundamentales para aplicaciones que combinan manipulación algorítmica y selección humana de colores. Para convertir de RGB a HSV:

$$
\begin{aligned}
R' &= R/255 \\
G' &= G/255 \\
B' &= B/255 \\
C_{\text{max}} &= \max(R', G', B') \\
C_{\text{min}} &= \min(R', G', B') \\
\Delta &= C_{\text{max}} - C_{\text{min}}
\end{aligned}
$$

$$
H = \begin{cases}
0, & \text{if } \Delta = 0 \\
60^\circ \times \left(\frac{G' - B'}{\Delta} \bmod 6 \right), & \text{if } C_{\text{max}} = R' \\
60^\circ \times \left(\frac{B' - R'}{\Delta} + 2 \right), & \text{if } C_{\text{max}} = G' \\
60^\circ \times \left(\frac{R' - G'}{\Delta} + 4 \right), & \text{if } C_{\text{max}} = B'
\end{cases} \tag{1}
$$

$$
S = \begin{cases}
0, & \text{if } C_{\text{max}} = 0 \\
\frac{\Delta}{C_{\text{max}}}, & \text{otherwise}
\end{cases}
$$

$$
V = C_{\text{max}}
$$

Para convertir de HSV a RGB:

$$
\begin{aligned}
C &= V \times S \\
X &= C \times (1 - |((H/60^\circ) \bmod 2) - 1|) \\
m &= V - C
\end{aligned}
$$

$$
(R', G', B') = \begin{cases}
(C, X, 0), & \text{if } 0^\circ \leq H < 60^\circ \\
(X, C, 0), & \text{if } 60^\circ \leq H < 120^\circ \\
(0, C, X), & \text{if } 120^\circ \leq H < 180^\circ \\
(0, X, C), & \text{if } 180^\circ \leq H < 240^\circ \\
(X, 0, C), & \text{if } 240^\circ \leq H < 300^\circ \\
(C, 0, X), & \text{if } 300^\circ \leq H < 360^\circ
\end{cases} \tag{2}
$$

$$
\begin{aligned}
R &= (R' + m) \times 255 \\
G &= (G' + m) \times 255 \\
B &= (B' + m) \times 255
\end{aligned}
$$

CMYK: El color como sustracción de luz

A diferencia de RGB, que parte de la oscuridad y añade luz, el modelo CMYK (*Cyan*, *Magenta*, *Yellow*, *Key*/*Black*) parte de una superficie blanca (típicamente papel) que refleja todas las longitudes de onda, y utiliza tintas para sustraer ciertas frecuencias de la luz reflejada.

Modelo que sustrae colores de la luz: cada color bloquea a su opuesto (son los colores complementarios a los de RGB). Por

tanto, una impresora de inyección de tinta lo que lee es la cantidad de tinta que debe insertar en una posición dada del papel (Figura 20). No trabajaremos con este modelo, pero es un buen ejemplo de cómo una aplicación real requiere modelar el color.

Cada tinta funciona como un filtro: el cian bloquea el rojo, el magenta bloquea el verde, y el amarillo bloquea el azul. Teóricamente, la combinación de CMY a máxima intensidad produciría negro, pero en la práctica produce un marrón oscuro, por lo que se añade una tinta negra separada (K) para lograr negros más profundos y reducir el consumo de tintas de color.

Aunque no trabajaremos con CMYK en este curso, entender sus principios es valioso para comprender la diferencia entre sistemas aditivos (pantallas) y sustractivos (impresión), y cómo un mismo color puede requerir representaciones distintas según el medio de reproducción. De hecho, CMYK no es equivalente a RGB, por lo no puedes imprimir una imagen con los mismos colores de una pantalla; sí puedes acercarte lo suficiente para que perceptualmente sean similares (aunque, ¿a cuál pantalla nos estamos refiriendo? ¡En pantallas diferentes, el color se verá diferente!).

Figura 20. Ejemplo de tintas sobre el papel. Fuente: Wikipedia.

Para quienes necesiten trabajar con modelos de color, la biblioteca `spectra` ofrece una interfaz elegante para operaciones y conversiones entre espacios de color. Con ella pueden:

- Convertir valores entre RGB, HSV, CMYK y otros espacios.
- Crear paletas de colores mediante reglas de armonía.
- Interpolar entre colores para crear gradientes.
- Ajustar propiedades como saturación, brillo o temperatura de color.

Esta biblioteca simplifica tareas que de otro modo requerirían implementar las fórmulas de conversión que hemos presentado. Otra alternativa es el módulo `colorsys` de la biblioteca estándar de Python, que permite convertir entre RGB, HSV y otros espacios de color sin instalar dependencias adicionales.

Para ver en la práctica cómo la interpolación entre los mismos colores produce resultados distintos según el espacio de color utilizado, se puede ejecutar el ejemplo `degradado` de la caja de juguetes del curso:

```
python caja_de_juguetes.py degradado
```

Con la barra de espacio se alternan distintos modos de interpolación, incluyendo interpolación bilineal en RGB y en HSV a partir de las mismas esquinas.

Cada modelo tiene sus ventajas según el contexto: RGB para representación digital directa, HSV para interfaces de usuario y CMYK para impresión física. Ahora que entendemos cómo representar colores individuales, ¿cómo construimos imágenes completas?

3 ¿Cómo se representa una imagen en una computadora?

Hasta ahora hemos analizado cómo se representan colores individuales, pero un color aislado no constituye una imagen. Una imagen digital es una colección estructurada de miles o millones de colores organizados en una cuadrícula bidimensional, donde cada celda (píxel) contiene información cromática específica. ¡Incluso con pocos píxeles es posible comunicar el diseño de un personaje (Figura 21)!

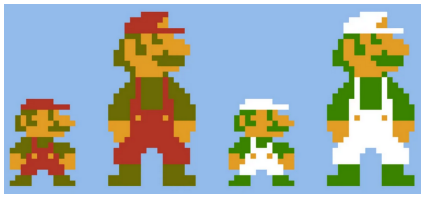


Figura 21. Mario y Luigi en *Super Mario Bros.* (1984).

Para representar imágenes digitales utilizaremos el modelo RGB, con la posible adición de un cuarto componente: el canal α para transparencia (Figura 22). Este canal α es transversal a los modelos de color y permite que ciertos píxeles sean parcial o totalmente transparentes, revelando lo que esté detrás de ellos.

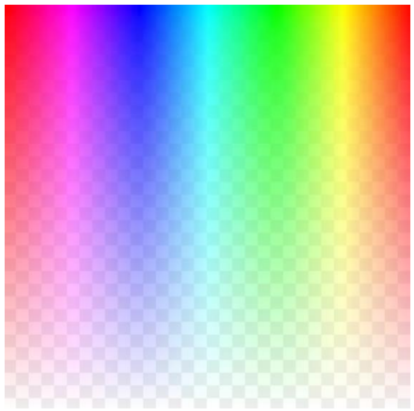


Figura 22. El canal α. La transparencia suele representarse con un patrón tipo tablero de ajedrez.

Así, el modelo que incluye transparencia se llama RGBA. En la práctica es utilizado en todo tipo de aplicaciones: los *stickers* de WhatsApp o Telegram incluyen transparencia, una ventana traslúcida en una interfaz, o un efecto de desvanecimiento en una animación son ejemplos donde necesitamos que los píxeles tengan diferentes grados de opacidad. Para calcular el color resultante de un píxel semitransparente, es necesario conocer tanto su color propio como el color del fondo sobre el que se superpone, lo que añade complejidad al *rendering*, como veremos en una unidad futura.

Formalmente, podemos definir un color en el espacio RGB como un vector tridimensional:

$$\text{Color}_{\text{RGB}} = (r, g, b) \in [0, 1]^3.$$

Y si incluimos el canal α para transparencia:

$$\text{Color}_{\text{RGBA}} = (r, g, b, \alpha) \in [0, 1]^4.$$

Donde cada componente $r, g, b, \alpha \in \mathbb{R}$ es un número de punto flotante con valores entre 0 y 1 (en caso de trabajar con números enteros, entre 0 y 255). Esto tiene implicaciones directas en el almacenamiento en memoria:

$$\text{Tamaño}(\text{Color}_{\text{RGB}}) = 3 \times \text{Tamaño}(\text{float}) = 3 \times 4 \text{ bytes} = 12 \text{ bytes},$$

$\text{Tamaño}(\text{Color}_{\text{RGBA}}) = 4 \times \text{Tamaño}(\text{float}) = 4 \times 4 \text{ bytes} = 16 \text{ bytes}.$

Una imagen se define entonces como una matriz tridimensional:

$$I \in [0,1]^{n \times m \times c},$$

donde n es el número de filas (altura de la imagen o *height*), m es el número de columnas (anchura o _width) y c es el número de canales ($c = 3$ para RGB o $c = 4$ para RGBA). La *resolución* de la imagen es el valor $r = n \times m$, que indica el número total de píxeles (celdas).

El modelo *raster*

Este enfoque de representar imágenes como matrices de píxeles se conoce como modelo *raster* o de mapa de bits. Al tener un número definido de píxeles, las imágenes *raster* pierden calidad al ampliarse mediante *zoom*, ya que no contienen información adicional para rellenar los nuevos píxeles, aunque puede utilizarse algún modelo que prediga lo que hay entre píxeles, o bien se puede interpolar con algoritmos como NoHalo [2].

Existen distintos formatos de imagen, como JPEG, que utiliza compresión con pérdida, con aplicaciones en su mayoría de fotografía; PNG, que emplea compresión sin pérdida, óptimo para gráficos con áreas de color uniforme y transparencia; o BMP, que almacena la información sin comprimir. Por ejemplo, una imagen RGBA en resolución 4K (3840×2160) contiene 8.294.400 píxels. En formato PNG, utilizando color de 8 bits, se requieren 33,18 megabytes de memoria (sin compresión).

El modelo *raster* es ubicuo en Computación Gráfica, especialmente para aplicaciones interactivas y en tiempo real, debido a que el *hardware* gráfico está optimizado para procesarlo. De hecho, el proceso de convertir una escena 3D a este formato se denomina *rasterización*, y es uno de los pilares del *renderizado* (lo discutiremos en la unidad de GPU y *Rendering Pipeline*).

El modelo vectorial

Existe otro paradigma para representar imágenes: el modelo vectorial. En lugar de almacenar una cuadrícula de píxeles, las imágenes vectoriales definen formas mediante primitivas gráficas expresadas como funciones matemáticas: líneas, curvas, polígonos y sus propiedades como color, grosor y relleno. Con creatividad, mezclando esas primitivas incluso se puede hacer un pingüino (Figura 23). Los primeros monitores que desplegaron gráficos utilizaban esta tecnología.

En contraste al modelo *raster*, las imágenes vectoriales no pierden calidad al ampliarse, ya que se recalculan para cada ni-

Figura 23. Háganle *zoom* a Tux al visitarlo en WikiCommons.

vel de *zoom*. Además, el peso de un archivo depende de la complejidad de sus primitivas, no de las dimensiones de la imagen. Debido a su uso de primitivas, es posible modificar componentes individuales de la imagen sin afectar al resto. Los formatos más comunes incluyen SVG (*Scalable Vector Graphics*), EPS (*Encapsulated PostScript*) y PDF (que puede combinar elementos vectoriales y *raster*).

La elección entre representaciones *raster* y vectoriales depende del propósito: las imágenes *raster* son ideales para fotografías y contenido con detalles complejos y orgánicos, mientras que las vectoriales brillan en ilustraciones, tipografía, diagramas y cualquier gráfico que requiera escalabilidad y edición precisa. Es común que los elementos de una UI (*User Interface*) se diseñen de manera vectorial, pero que su graficación se realice con una versión *raster*.

4 Arquitectura

Una vez que entendemos cómo se representan los colores y las imágenes en memoria, es fundamental comprender el camino que recorren estos datos hasta manifestarse como luz en nuestras pantallas (Figura 24). Este proceso involucra una arquitectura especializada que ha evolucionado con el avance de la tecnología gráfica.

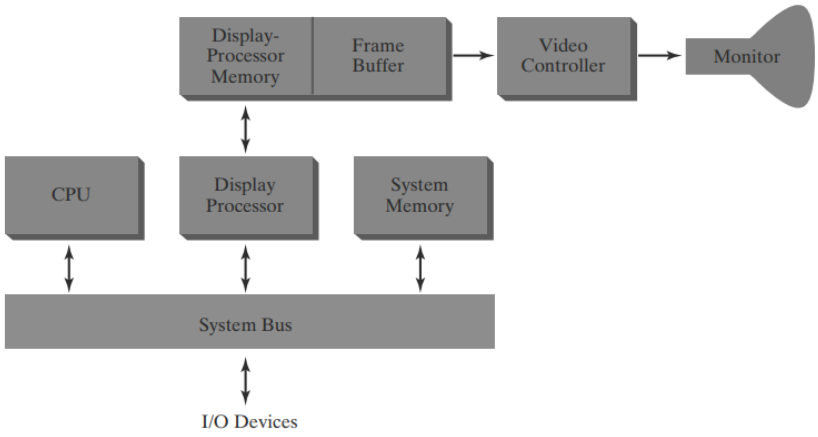


Figura 24. Arquitectura de un sistema *raster*.

En el corazón de todo sistema gráfico se encuentra el *frame buffer*, una región de memoria dedicada que almacena la representación de la imagen que se mostrará en pantalla. Cada celda de este *buffer* corresponde a un píxel específico, conteniendo su información de color (y, en caso de ser RGBA, transparencia).

En los sistemas gráficos modernos, este *buffer* reside en la memoria dedicada de la GPU (Unidad de Procesamiento Gráfico)[6]. A menudo se implementa un sistema de *buffer* doble o in-

[6] En sistemas integrados o aplicaciones de *software rendering* no se utiliza GPU.

cluso triple:

- El *front buffer* contiene la imagen que se está mostrando.
- El *back buffer* es donde se construye la siguiente imagen a mostrar.
- Cuando la nueva imagen está completa, se realiza un *swap* (intercambio) de *buffers*.

Esta técnica, conocida como *page flipping*, permite que la visualización sea fluida, evitando artefactos visuales como el *tearing* (desgarros en la imagen) que ocurrirían si se modificara el *buffer* mientras se muestra.

Un componente especializado, el *video controller* o controlador de video, es responsable de leer el contenido del *frame buffer* y enviarlo a la pantalla o proyector. Este proceso ocurre a una frecuencia específica, medida en *frames per second* (FPS, cuadros por segundo) o Hertz (Hz).

En los sistemas actuales, esta tasa de refresco suele ser como mínimo de 60 Hz, aunque en aplicaciones de alta respuesta como videojuegos competitivos o realidad virtual, se buscan frecuencias más altas (120Hz, 144Hz o incluso 240Hz) para reducir la latencia perceptible y mejorar la respuesta del juego y la interacción con él.

Por tanto, el *video controller* maneja la conversión de los datos digitales a las señales específicas requeridas por diferentes tecnologías de pantalla (LCD, OLED, etc.) y estándares de conexión (HDMI, DisplayPort, etc.).

Manipulación de matrices con `numpy`

Para entender mejor cómo se genera este contenido, veamos cómo podemos hacerlo nosotros usando Python. Utilizaremos la biblioteca `numpy` que nos permite definir una matriz y la biblioteca `matplotlib` que nos permite dibujarla en pantalla. El siguiente código inicializa una matriz con ceros (`np.zeros`) y luego ejecuta dos bucles: uno en el eje x, para aumentar la intensidad en el canal rojo, y luego uno en el eje y, para aumentar la intensidad del canal azul. Como consecuencia, se obtiene una imagen con gradiente rojo horizontal y gradiente azul cuadrático vertical (Figura 25).

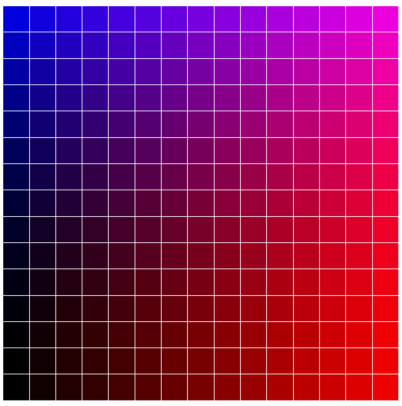


Figura 25. Un conjunto de píxeles en los que se varía linealmente el canal rojo en el eje x; mientras que el canal azul en el eje y varía de manera cuadrática.

```
import numpy as np
import matplotlib.pyplot as plt

altura, ancho = 15, 15
imagen_rgb = np.zeros((altura, ancho, 3))

# gradiente horizontal de rojo
```

```
for x in range(ancho):
    imagen_rgb[:, x, 0] = x / ancho

# gradiente vertical de azul
for y in range(altura):
    imagen_rgb[y, :, 2] = np.power(y, 2) / np.power(altura, 2)

# mostrar la imagen sin interpolación
plt.imshow(imagen_rgb, interpolation='nearest', origin='lower')

# dibujar líneas de cuadrícula blancas
for i in range(altura + 1):
    plt.axhline(i - 0.5, color='white', linewidth=0.5)
for j in range(ancho + 1):
    plt.axvline(j - 0.5, color='white', linewidth=0.5)

plt.axis('off')
```

En el código no se menciona el modelo de color: `matplotlib` trabaja con RGB de manera implícita. La instrucción `imagen_rgb[:, x, 0]` significa "todas las filas (`:`) en la columna `x`, canal rojo (`0`)". Esto podría parecer contraintuitivo, pero se debe a la convención de indexación de `numpy`. Nota que al crear la imagen, se utiliza la función `np.zeros` con los parámetros `(altura, ancho, canales)` y no `(ancho, altura, canales)`.

Generación del contenido: El *fragment program*

En sistemas gráficos (particularmente de ejecución en tiempo real), el contenido del *frame buffer* se computa mediante *fragment programs* (o *pixel shader* en terminología DirectX), un tipo de programa especializado que se ejecuta en la GPU para cada píxel (o fragmento) de la imagen. Un *fragment program* es responsable de determinar el color final de cada píxel en función de diferentes factores relacionados con la escena, su configuración y la estética buscada en la aplicación. Veremos con detalle diferentes tipos de *fragment program* a lo largo del curso.

Estos programas son parte de un pipeline más amplio de renderización, que comienza con geometría 3D y culmina con la imagen 2D en el *frame buffer*, proceso que estudiaremos en detalle en la unidad sobre la *Rendering Pipeline*.

Profundidad de color y memoria gráfica

Ya dijimos cuánto espacio requiere un color cuando se utilizan enteros de 8 bits. Pero este número es variable: la cantidad de bits utilizada para expresar colores define la "profundidad"

del color. Entonces, la cantidad de memoria requerida para el *frame buffer* depende de la resolución de la pantalla y de la profundidad de color del sistema.

En la historia, las limitaciones de memoria impusieron restricciones severas en estos parámetros. Hace décadas era común encontrar sistemas con profundidades de color limitadas:

- 1 bit por píxel: Monocromático (blanco y negro).
- 3 bits por píxel: 8 colores posibles (Figura 26).
- 8 bits por píxel: 256 colores.
- 16 bits por píxel: 65,536 colores (a veces llamado *high color*).
- 24 bits por píxel: Aproximadamente 16.7 millones de colores (*true color*).
- 32 bits por píxel: *True color* con canal α.

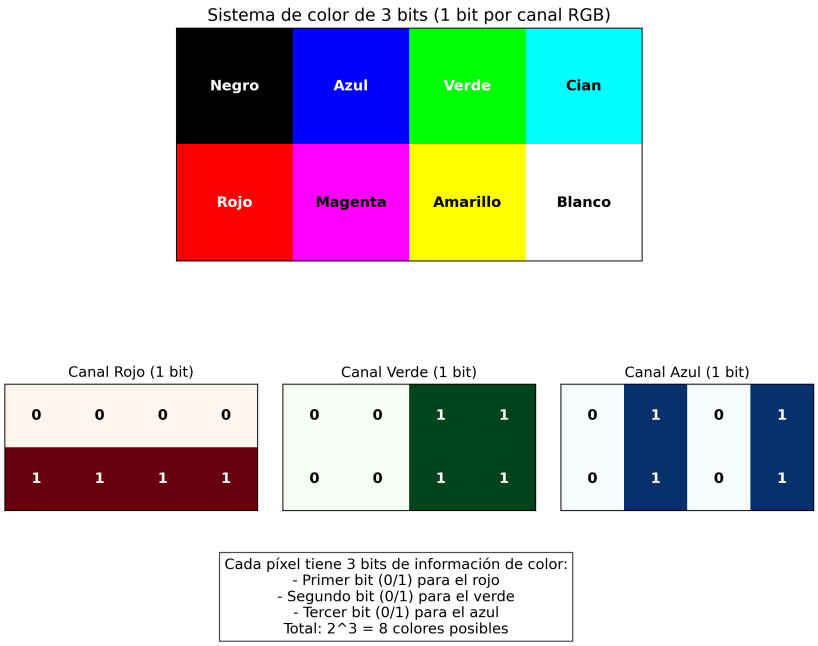


Figura 26. Un ejemplo de combinación de colores utilizando tres bits por celda: el primer bit es para el rojo; el segundo, para el verde; y el tercero, para el azul. La fila superior muestra el resultado de combinar los bits indicados en la fila inferior.

Figura 27. *Pseudo color*. Con pocos colores es posible crear imágenes realistas.

Para maximizar las capacidades visuales con recursos limitados, los sistemas gráficos antiguos implementaban técnicas ingeniosas como el *pseudo color* o color indexado (Figura 26).

En lugar de almacenar el valor RGB de cada píxel, se utilizaba una tabla de colores (paleta) y el *frame buffer* solo almacenaba índices a esta tabla. Por ejemplo, con 8 bits por píxel, se podían referenciar 256 colores diferentes de una paleta que podía incluir cualquier selección de colores RGB completos. Esta técnica no solo ahorraba memoria, sino que también permitía efectos especiales al modificar la paleta en tiempo de ejecución. Cambiar un solo valor en la tabla modificaba instantáneamente todos los píxeles que lo utilizaban, facilitando efectos como

Figura 28. Una obra de *pixel art* de Mark Ferrari.

ciclos de color, fundidos y transiciones. Las Figuras 28 y 29 son buenos ejemplos.

Los pioneros del *pixel art* y los desarrolladores de videojuegos de las generaciones de 8 y 16 bits crearon efectos visuales impresionantes a pesar de las severas limitaciones técnicas. Consolas como el NES, Game Boy, Sega Master System y Super Nintendo (que permitía manipular paletas de 256 colores seleccionados de un espacio de 32,768 posibilidades) definieron una estética que sigue influyendo el arte digital actual.

Esto también tenía usos prácticos. Por ejemplo, en *Super Mario Bros.* se puede jugar con Mario o Luigi. Ambos personajes tienen el mismo *sprite*, es decir, la misma imagen, pero su paleta es diferente (Figura 21). Esto se controla en el juego mediante un *byte* que contiene el índice de la paleta a utilizar: 0 si es Mario y 1 si es Luigi. Pero... ¿si es un *byte*, acaso no hay otros posibles valores? Resulta que si mediante un emulador se manipula el índice de la paleta y se utilizan otros valores, aparece lo que se ha llamado "invalid brother", el mismo *sprite* de Mario y Luigi pero con colores y un nombre distinto (Figura 30).

Figura 29. Otra obra de *pixel art* de Mark Ferrari. Los píxeles contienen la misma información: los índices de los colores que contiene cada uno. Lo que ha cambiado es la paleta de colores utilizada; como resultado, la imagen es diferente.

Figura 30. Fuente: Supper Mario Broth.

Para finalizar, aunque la memoria ya no es una limitación tan restrictiva, algunas de estas técnicas siguen siendo relevantes en contextos específicos como dispositivos embebidos, gráficos procedurales y, por supuesto, en el vibrante mundo del *pixel art retro*.

En las próximas unidades, profundizaremos en cómo estas estructuras de bajo nivel interactúan con algoritmos sofisticados de renderizado para producir las experiencias visuales inmersivas que caracterizan a la computación gráfica moderna.

5 Ejercicios

Verdadero/Falso

1. (Control 1, Otoño 2025). En el modelo de color RGB, un valor de (1,1,1) representa el blanco, mientras que en HSV el mismo vector representaría un tono saturado.
2. (Control 1, Otoño 2025). Las conos S, M y L en la retina humana están distribuidos uniformemente, permitiendo igual sensibilidad a todos los colores del espectro visible.
3. (Control 1, Primavera 2025). En el modelo HSV, los tres componentes tienen el mismo rango [0,1], al igual que en RGB.
4. (Examen, Otoño 2025). En el modelo CMYK, cuando $K = 1$, todos los canales CMY resultan indefinidos.
5. (Examen, Primavera 2025). Para convertir RGB a HSV, el tono o matiz (*hue*) se calcula como $H = \arctan\left(\frac{\sqrt{3}(G-B)}{2R-G-B}\right)$, y esta fórmula es simétrica para todos los colores del espectro.

(Control 1, Otoño 2025) Conversión CMYK

Las personas que trabajan en diseño gráfico suelen manipular colores en el modelo HSV debido a su intuitividad, pero necesitan convertirlos a CMYK para impresión profesional.

a) Considerando que no podemos convertir directamente de HSV a CMYK, proponga y justifique una estrategia de conversión: derive los pasos y ecuaciones necesarias para convertir valores CMYK a partir de HSV.

Recuerde que CMYK es un **modelo sustractivo** donde cada tinta absorbe su color complementario:

- C (Cian) absorbe el Rojo.
- M (Magenta) absorbe el Verde.
- Y (Amarillo) absorbe el Azul.
- K (Negro) se utiliza por razones prácticas y económicas: en teoría, la combinación de CMY a máxima intensidad debería producir negro, pero en la práctica produce un marrón oscuro; además, usar tres tintas en alta concentración puede saturar el papel, y la tinta negra es más económica que usar las tres tintas CMY en conjunto.

Derive las ecuaciones para calcular C, M, Y y K siguiendo estos pasos:

1. Determine los valores preliminares de C, M, Y sin considerar K.
2. Determine K como la "oscuridad común" presente en los tres canales CMY, que representa la cantidad máxima de negro que puede extraerse sin alterar el balance de color.
3. Calcule los valores finales de C, M, Y considerando la presencia de K. Esta parte es crucial: una vez que parte del color se producirá con tinta negra, los valores CMY deben ajustarse proporcionalmente. La fórmula de ajuste debe considerar cuánto del espacio total de color será ocupado por la tinta negra y cómo deben redistribuirse los componentes CMY en el espacio restante para mantener el mismo tono original.

Hint: para definir este modelo preste atención al significado de absorber y de "oscuridad común" proporcional.

b) Considere el color HSV(210°, 0.8, 1.0). Primero, describa qué color es. Luego conviértalo a CMYK.

Referencias

[1] Inostroza, L., Palme, M. y De La Barrera, F. 2016. A heat vulnerability index: spatial patterns of exposure, sensitivity and adaptive capacity for Santiago de Chile. *PLOS one*. 11, 9 (2016), e0162464. https://doi.org/10.1371/journal.pone.0162464.

[2] Robidoux, N., Gong, M., Cupitt, J., Turcotte, A. y Martinez, K. 2009. CPU, SMP and GPU implementations of Nohalo level 1, a fast co-convex antialiasing image resampler. *Proceedings of the 2nd Canadian Conference on Computer Science and Software Engineering* (2009), 185-195.

[3] Smith, A.R. 1978. Color gamut transform pairs. *SIGGRAPH Comput. Graph.* 12, 3 (ago. 1978), 12-19. https://doi.org/10.1145/965139.807361.

[4] Stone, M.C. 2005. Representing colors as three numbers [color graphics]. *IEEE Computer Graphics and Applications*. 25, 4 (2005), 78-85. https://doi.org/10.1109/MCG.2005.84.