Ventanas

Gráficas por Computadora (CC2018) - 2026

Ventanas

Semestre 02, 2026

De renderizado estático a tiempo real

En la clase anterior usamos la memoria RAM para construir una imagen y escribirla en un archivo. A ese proceso lo llamamos renderizado estático: se calcula una sola imagen y se guarda.

De ahora en adelante trabajaremos con renderizado en tiempo real: repetimos el proceso de pintar muchas veces por segundo dentro de un ciclo (`while true`) mostrado en una ventana.

El ciclo básico es:

- Pintar la pantalla.
- Mostrarla.
- Limpiarla y volver a pintar con ligeras modificaciones.

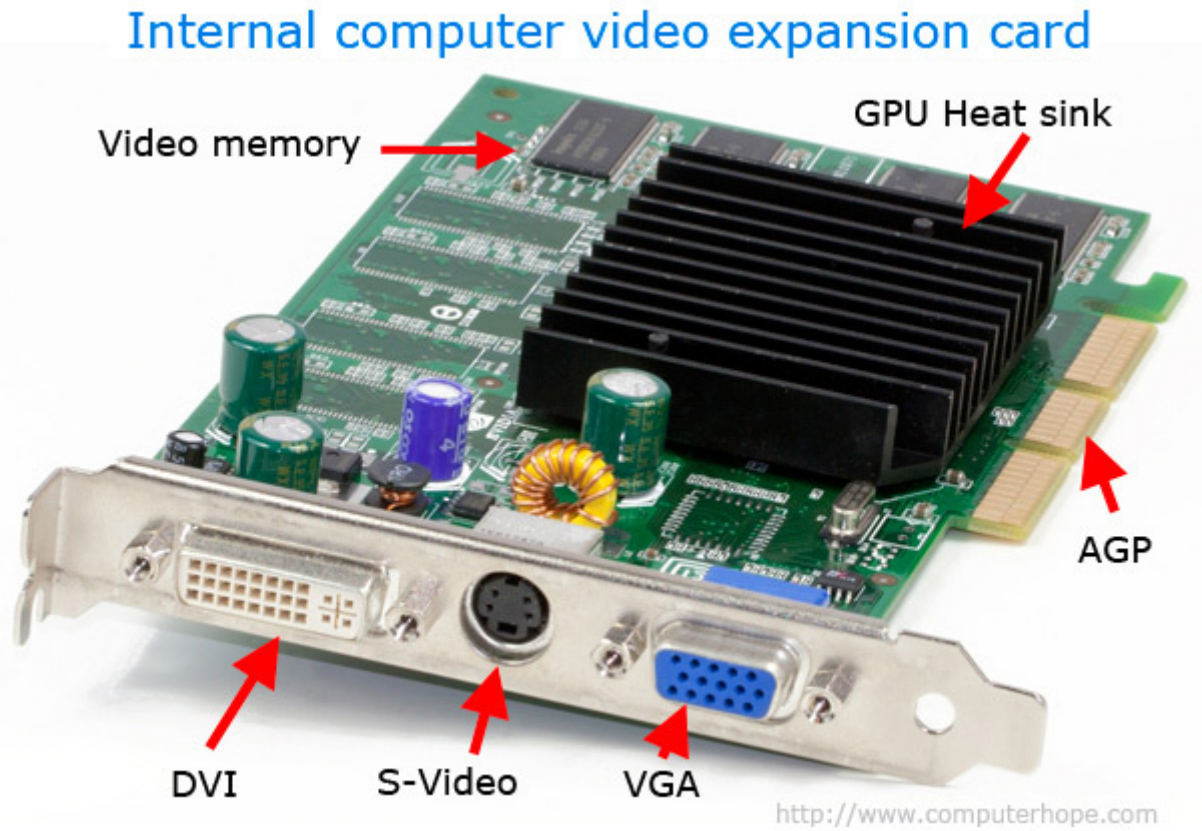
Al repetir esto rápidamente se crea la ilusión de movimiento.

Framebuffer y GPU

Su computadora tiene un framebuffer que funciona igual que el del ejercicio anterior: una región de memoria con el color de cada píxel.

Si tienen una tarjeta de video (GPU), el framebuffer vive en la memoria de video y lo gestiona la GPU.

Si no, el framebuffer está en la memoria RAM y lo gestiona la CPU (software rendering).



La tarjeta de video tiene la interfaz directa con el monitor: convierte los bytes del framebuffer en una señal eléctrica que el monitor puede mostrar.

Los cables VGA, DVI, HDMI, etc. son también protocolos que la GPU usa para comunicarse con el monitor.

Doble buffer y tearing

El monitor lee el framebuffer muchas veces por segundo para refrescar la imagen. Si dibujamos sobre ese mismo framebuffer mientras el monitor lo está leyendo, puede mostrar una parte del frame anterior y otra del nuevo al mismo tiempo. A ese artefacto lo llamamos tearing (desgarro).

La solución es usar dos framebuffers:

- Front buffer: el que el monitor está mostrando.
- Back buffer: donde la aplicación dibuja el siguiente frame.

Cuando el frame está listo, se intercambian (swap): el back pasa a ser front y viceversa. El monitor nunca ve un frame a medio dibujar.

El vsync sincroniza ese intercambio con el refresco del monitor (por ejemplo, 60 Hz) para que el swap ocurra en el momento seguro y se elimine el tearing.

El driver de video

Escribir en la memoria de video es trabajo del driver de video, específico de cada fabricante:

- Nvidia usa CUDA.
- AMD usa GCN.
- Intel usa sus propias arquitecturas.

El driver abstrae los detalles del hardware y expone APIs para que el sistema operativo pueda gestionar los gráficos sin conocer cada modelo de tarjeta.

Window System

El sistema operativo ejecuta varios programas que quieren escribir en la pantalla al mismo tiempo.

Para pintar, una aplicación no necesita preocuparse por coordenadas absolutas de la pantalla: el Window System traduce las coordenadas relativas de la aplicación a posiciones dentro del framebuffer.

También provee una interfaz gráfica básica: ventanas, cursores, eventos de teclado y ratón, etc.

El problema que resuelve

Los sistemas antiguos (como DOS) solo ejecutaban un programa a la vez, así que no había conflicto por la pantalla.

Con multitarea necesitamos un Window System que organice:

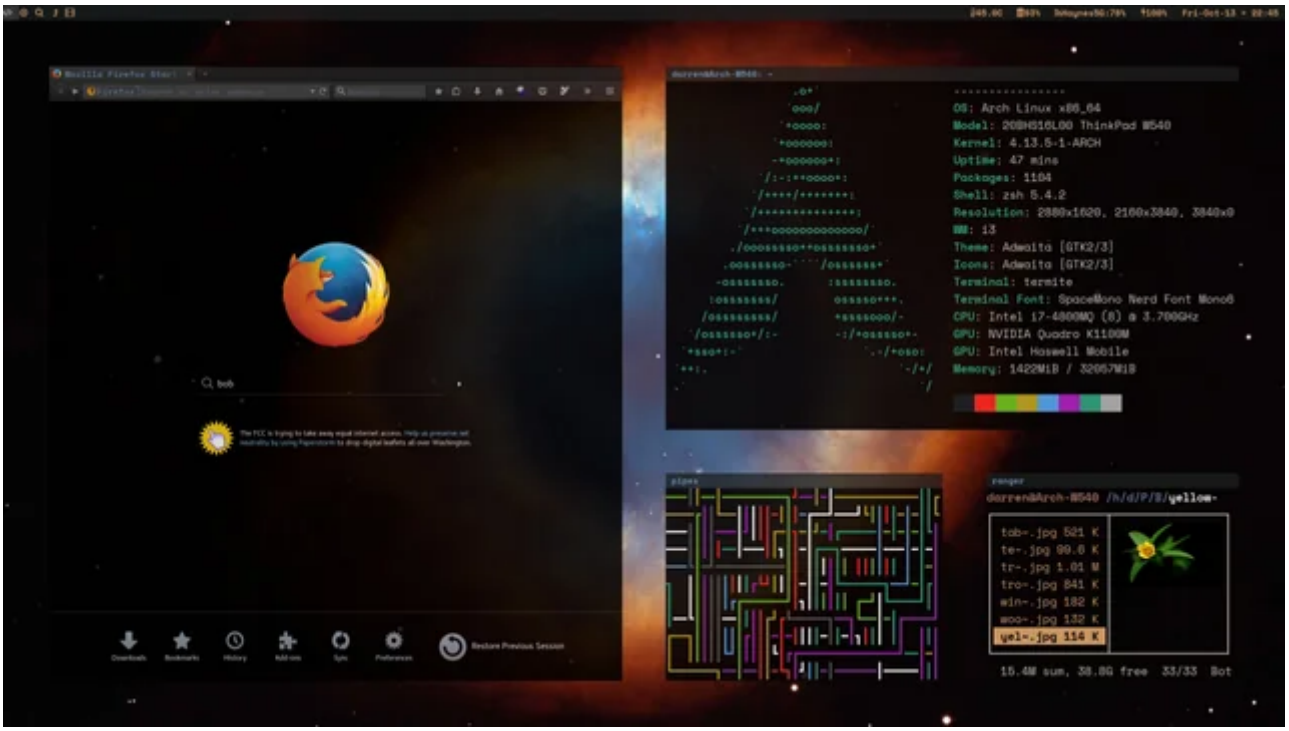
- Qué píxeles pertenecen a cada programa.
- Cuándo y cómo se debe dibujar cada uno.

Ejemplo con X11 (Linux)

En X11 el servidor gestiona la pantalla y las aplicaciones son clientes:

- Firefox pide abrir una ventana de 800x600.
- X11 le asigna un área dentro del framebuffer.
- Firefox pinta sus elementos únicamente en ese espacio.

A esa área asignada la llamamos una ventana.



X11 y Wayland

X11 es un protocolo antiguo (de 1984): el servidor X hace casi todo y el compositor se añadió después, encima. Además, cualquier cliente puede leer los eventos y el contenido de otras ventanas, lo que hoy se considera un problema de seguridad.

Wayland es la alternativa moderna: el compositor y el servidor de display son lo mismo. Cada aplicación dibuja el contenido de su propia ventana en un buffer (renderizado del lado del cliente) y se lo entrega al compositor, que solo combina esos buffers. El resultado es más simple, más seguro y con menos latencia.

El camino de un evento

Cuando presionas una tecla o mueves el ratón, el evento recorre varias capas antes de llegar al programa:

- El hardware genera una señal que el kernel recibe a través de su driver.
- El servidor de display (X11 o Wayland) recibe el evento del kernel.

- El servidor decide a qué ventana pertenece (normalmente la que tiene el foco) y le entrega el evento.
- El render loop de la aplicación lo lee en su fase de `process_input()`.

Por eso una aplicación no habla directamente con el teclado: recibe eventos ya traducidos y dirigidos por el Window System.

Window Manager y Compositor

El Window Manager es un cliente especial dentro del Window System cuyo trabajo es controlar cómo se muestran las ventanas.

El Window Manager decide:

- Posición y tamaño de cada ventana.
- Decoraciones: barra de título, botones de cerrar/minimizar/maximizar.
- Comportamiento al maximizar, minimizar o mover una ventana.

Para efectos más complejos (transparencias, sombras, animaciones) se usa un compositor, que combina las ventanas en la imagen final.

Ejemplos:

- Linux: i3, GNOME Mutter, KWin.
- macOS: Quartz Compositor.
- Windows: Desktop Window Manager (DWM).

La organización por capas se ve así:

```
┌──────────────────────────────────────┐
│           Window System              │  ← X11, Wayland
│  ┌──────────────────────────────┐    │
```

```
│  │        Window Manager         │  │  ← i3, KWin, Mutter
│  └──────────────────────────────┘  │
│  ┌──────────────────────────────┐  │
│  │         Applications          │  │  ← Firefox, terminal…
│  └──────────────────────────────┘  │
└────────────────────────────────────┘

                ↓
        Framebuffer → GPU → Monitor
```

Window Toolkits

Los Window Toolkits son bibliotecas que facilitan crear interfaces gráficas al abstraer la complejidad de los Window Systems y Window Managers.

Permiten crear ventanas, botones, menús y controles sin manejar directamente el framebuffer ni los eventos del sistema operativo.

Gestionan la comunicación con el Window System (X11, Wayland, Quartz, DWM) y hacen que el código sea portable entre plataformas.

Ejemplos: Qt, GTK, wxWidgets, y a más bajo nivel, librerías como SDL o GLFW para gráficos y videojuegos.

En el proyecto de gráficas

En este curso no se habla con X11 ni con Wayland directamente. Se usa un Window Toolkit (por ejemplo minifb, raylib o SDL) que da dos cosas: una ventana y un framebuffer donde escribir los píxeles.

El raytracer calcula un color para cada píxel y lo escribe en ese buffer; el toolkit se encarga del swap hacia la pantalla dentro del render loop.

Todas las capas que vimos (driver, Window System, Window Manager, compositor) siguen ahí debajo, pero el toolkit las abstrae por ti.