

# Tema 3: Introducción a la biblioteca SDL

Tecnología de la Programación de Videojuegos 1

Grado en Desarrollo de Videojuegos

Curso 2023-2024

Miguel Gómez-Zamalloa Gil con de Rubén Rubio Cuéllar  
cambios

Departamento de Sistemas Informáticos y Computación

Universidad Complutense de Madrid

# ¿Por qué SDL?

- ✦ Cada plataforma tiene su propia forma de manejar la salida gráfica, la entrada del usuario, acceder al hardware, etc.
- ✦ SDL proporciona **acceso uniforme y eficiente** a todas estas características específicas de cada plataforma (audio, teclado, ratón, joystick, gráficos 2D y 3D vía OpenGL, etc.)
  - ▶ ¡Más tiempo para centrarse en el desarrollo propio del juego!
- ✦ **Multiplataforma:** Windows, macOS, Linux, iOS y Android
- ✦ Activamente mantenido y distribuido → manuales, tutoriales, foros, etc.
  - ▶ Ver [wiki.libsdl.org](http://wiki.libsdl.org)

# Nuevo en SDL 2

- ◆ Aceleración 2D (hardware) y 3D
- ◆ Soporte para múltiples ventanas
- ◆ Soporte para entrada de tipo multi-touch
- ◆ Múltiples fuentes de audio y múltiples fuentes de entrada
- ◆ Captura de audio
- ◆ Mejoras en programación multi-hebra
- ◆ Manejo del portapapeles, rueda horizontal del ratón, etc.

# Inicialización y finalización

```
#include <SDL.h> // Esto puede cambiar dependiendo de la instalación
constexpr int WIN_WIDTH = 800;
constexpr int WIN_HEIGHT = 600;

int main(int argc, char* argv[]) {
    // Variables
    SDL_Window* window = nullptr;
    SDL_Renderer* renderer = nullptr;
    int winX, winY; // Posición de la ventana
    winX = winY = SDL_WINDOWPOS_CENTERED;
    // Inicialización del sistema, ventana y renderer
    SDL_Init(SDL_INIT_EVERYTHING);
    window = SDL_CreateWindow("First test with SDL", winX, winY,
                              WIN_WIDTH, WIN_HEIGHT, SDL_WINDOW_SHOWN);
    renderer = SDL_CreateRenderer(window, -1, SDL_RENDERER_ACCELERATED);
    if (window == nullptr || renderer == nullptr)
        cout << "Error initializing SDL\n"; // En general lanzaremos una excepción
    else { // Programa que usa SDL
        ...
    }
    // Finalización
}
```

# Inicialización y finalización

```
#include <SDL.h> // Esto puede cambiar dependiendo de la instalación
constexpr int WIN_WIDTH = 800;
constexpr int WIN_HEIGHT = 600;

int main(int argc, char* argv[]) {
    // Variables
    ...
    // Inicialización del sistema, ventana y renderer
    ...
    if (window == nullptr || renderer == nullptr)
        cout << "Error initializing SDL\n"; // En general lanzaremos excepción
    else { // Programa que usa SDL
        ...
    }
    // Finalización
    SDL_DestroyRenderer(renderer);
    SDL_DestroyWindow(window);
    SDL_Quit();
}
```

# Dibujando con SDL

```
#include <SDL.h> // Esto puede cambiar dependiendo de la instalación
constexpr int WIN_WIDTH = 800;
constexpr int WIN_HEIGHT = 600;

int main(int argc, char* argv[]) { // Tiene que ser así exactamente
    // [...] Variables
    // [...] Inicialización del sistema, ventana y renderer
    if (window == nullptr || renderer == nullptr)
        cout << "Error initializing SDL\n";
    else { // Programa que usa SDL
        SDL_SetRenderDrawColor(renderer, 0, 0, 0, 255); // RGB y alpha
        SDL_RenderClear(renderer); // Borra la pantalla
        SDL_RenderPresent(renderer); // Muestra la escena
        SDL_Delay(5000); // Espera 5 segs. antes de cerrar
    }
    // Finalización
    SDL_DestroyRenderer(renderer);
    SDL_DestroyWindow(window);
    SDL_Quit();
}
```

# Dibujando con SDL - Texturas

SDL maneja dos estructuras básicas para hacer dibujos, `SDL_Surface` y `SDL_Texture`:

- ♦ `SDL_Surface` contiene una colección de píxeles y se renderiza mediante software (no la GPU)
- ♦ `SDL_Texture` permite el renderizado acelerado por hardware (GPU). ¡Es la que usaremos!

## Creación de texturas:

```
SDL_Texture* texture; // Variable para la textura
string filename = "..."; // Nombre del fichero con la imagen .bmp
SDL_Surface* surface = SDL_LoadBMP(filename.c_str()); // Solo para bmps
texture = SDL_CreateTextureFromSurface(renderer, surface);
SDL_FreeSurface(surface); // Se borra la estructura auxiliar
// Textura lista para ser usada
```

# Texturas - ciclo de vida

Típicamente las texturas se crean, se utilizan repetidamente en el juego (repintados) y al final del programa se destruyen:

1. Creación: (ver diapositiva anterior)
2. Ejemplo de ciclo de repintados: sea `vector<SDL_Texture*> textures`

```
while (!exit){  
    // Actualizar posiciones de objetos del juego  
    SDL_RenderClear(renderer); // Borra la pantalla  
    for (int i = 0; i < textures.size(); ++i)  
        SDL_RenderCopy(renderer, textures[i], ...); // Copia en buffer  
    SDL_RenderPresent(renderer); // Muestra la escena  
}
```

3. Destrucción:

```
for (SDL_Texture* texture : textures)  
    SDL_DestroyTexture(texture); // Borra memoria dinámica
```



# Rectángulos fuente y destino

- ♦ El **rectángulo fuente** define el área que queremos copiar de la textura a la ventana SDL
- ♦ El **rectángulo destino** define el área de la ventana SDL en la que queremos copiar el rectángulo fuente
- ♦ Sean srcRect y destRect del tipo **SDL\_Rect\*** los rectángulos fuente y destino, se usan así:

```
SDL_RenderCopy(renderer, textures[i], &srcRect, &destRect);
```

- ♦ Obsérvese que se esperan punteros a **SDL\_Rect** (por eso el &)
- ♦ Si se pasa **nullptr** en un rectángulo se interpreta como el rectángulo fuente/destino que ocupa toda la textura/ventana

# Rectángulos fuente y destino

El tipo `SDL_Rect` es una estructura con cuatro campos:

```
struct SDL_Rect {  
    int x;    // Coordenada x de la esquina superior izqda.  
    int y;    // Coordenada y de la esquina superior izqda.  
    int w;    // Anchura del rectángulo  
    int h;    // Altura del rectángulo  
}
```

**Ejemplo 1:** Pintamos toda la textura pero escalada a un frame de 50x50

```
SDL_Rect destRect;  
destRect.w = destRect.h = 50; // Frame de 50x50  
destRect.x = destRect.y = 0; // Se pinta en la esquina superior izqda  
SDL_RenderCopy(renderer, texture, nullptr, &destRect);  
SDL_RenderPresent(renderer); // Muestra la escena
```

# Rectángulos fuente y destino

**Ejemplo 2:** Imaginamos una matriz virtual de 8x8 en la escena y pintamos el primer frame (fantasma rojo) escalado a la celda (0,1) y el frame 11° (primer pacman) a la celda (4,3):

```
int textW, textH; // Para saber el tamaño de la textura
SDL_QueryTexture(texture, nullptr, nullptr, &textW, &textH);
SDL_Rect srcRect, destRect;
srcRect.w = textW/14; srcRect.h = textH/4; // Tamaño frame textura
uint cellW = WIN_WIDTH/8; uint cellH = WIN_HEIGHT/8;
destRect.w = cellW; destRect.h = cellH; // Tamaño celda salida
srcRect.x = srcRect.y = 0; // Frame fantasma rojo
destRect.x = 1*cellW; destRect.y = 0*cellH; // Celda (0,1)
SDL_RenderCopy(renderer, texture, &srcRect, &destRect);
srcRect.x = 10 * textW/14; // Frame pacman
destRect.x = 3*cellW; destRect.y = 4*cellH; // Celda (4,3)
SDL_RenderCopy(renderer, texture, &srcRect, &destRect);
SDL_RenderPresent(renderer); // Muestra la escena
```



# Rectángulos fuente y destino

**Ejemplo 3:** Creando una animación a partir de un "sprite-sheet"



Cada `TIME_PER_FRAME` milisegundos movemos el rectángulo fuente una posición a la derecha

```
int textFrameW = textW / 6;
int textFrameH = textH / 1;
SDL_Rect srcRect;
srcRect.x = srcRect.y = 0;
srcRect.w = textFrameW;
srcRect.h = textFrameH;
while (!exit) {
    srcRect.x = textFrameW * int(((SDL_GetTicks() / TIME_PER_FRAME) % 6));
    SDL_RenderClear(renderer);
    SDL_RenderCopy(renderer, texture, &srcRect, nullptr);
    SDL_RenderPresent(renderer); // Muestra la escena
}
```

# El paquete SDL\_Image

- ♦ SDL por defecto solo nos permite usar ficheros **bmp**
- ♦ El paquete **SDL\_Image** permite la carga de otros formatos de ficheros de imágenes (**png**, **jpeg**, **tiff**, etc.)
- ♦ Una vez instalado e incluido mediante:

```
#include <SDL_Image> // Puede variar según la instalación
```

- ♦ Simplemente hay que sustituir el

```
SDL_Surface* surface = SDL_LoadBMP(filename.c_str());
```

por la llamada análoga

```
SDL_Surface* surface = IMG_Load(filename.c_str());
```

# Ficheros png

- ♦ Los ficheros **png** a parte de tener un tamaño muy pequeño tienen soporte para el **canal alfa**
- ♦ Esto permite fundir imágenes con el fondo. Por ejemplo, nos permitiría en lugar de ver la imagen así:



verla de esta forma:



# Manejo básico de eventos

- ✦ SDL mantiene una cola con los eventos pendientes de ser procesados
- ✦ La función `SDL_PollEvent` `int SDL_PollEvent(SDL_Event* event)` devuelve 1 o 0 si hay o no algún evento pendiente y el primer evento pendiente en la variable `event`
- ✦ La estructura `SDL_Event` tiene dos campos, `type` y `timestamp`. Ver documentación de SDL para ver los distintos tipos de eventos
- ✦ Algunos ejemplos de tipos de eventos: `SDL_QUIT`, `SDL_WINDOWEVENT`, `SDL_KEYDOWN`, `SDL_KEYUP`, `SDL_MOUSEBUTTONDOWN`, `SDL_MOUSEMOTION`, `SDL_FINGERDOWN`, `SDL_CONTROLLERBUTTONDOWN`, `SDL_APPDIDENTERBACKGROUND`, ...

# Manejo básico de eventos

**Ejemplo:** bucle que trata todos los eventos pendientes y procesa algunos de ellos

```
SDL_Event event;
...
while (!exit) {
    ...
    while (SDL_PollEvent(&event) && !exit) {
        if (event.type == SDL_QUIT)
            exit = true;
        else if (event.type == SDL_KEYDOWN) {
            if (event.key.keysym.sym == SDLK_DOWN) ...
            else if (event.key.keysym.sym == SDLK_UP) ...
        } else if (event.type == SDL_MOUSEBUTTONDOWN) {
            if (event.button.button == SDL_BUTTON_LEFT) ...
        }
    }
    ...
}
```



# Control básico del tiempo

- ✦ La función **int** `SDL_GetTicks()` nos devuelve el tiempo actual en milisegundos
- ✦ La función **void** `SDL_Delay(int delay)` suspende el programa SDL `delay` milisegundos
- ✦ Tenemos dos alternativas básicas para controlar que el juego solo actualice cada `FRAME_RATE` milisegundos:
  1. Contar el tiempo consumido en una iteración y si no se ha llegado a `FRAME_RATE` suspender el programa el tiempo correspondiente
  2. Contar el tiempo consumido y si no se ha llegado a `FRAME_RATE` repintar y tratar eventos, pero NO actualizar

# Control básico del tiempo

Alternativa 1 (suspendiendo el programa):

```
uint32_t startTime, frameTime;

while (!exit) { // Bucle del juego
    startTime = SDL_GetTicks();
    handleEvents();
    update(); // Actualiza el estado de todos los objetos del juego
    render(); // Renderiza todos los objetos del juego
    frameTime = SDL_GetTicks() - startTime; // Tiempo de la iteración
    if (frameTime < FRAME_RATE) // Suspende por el tiempo restante
        SDL_Delay(FRAME_RATE - frameTime);
}
```

Es más eficiente (suelta el procesador y ahorra energía) pero dependiendo del juego puede resultar inaceptable.

# Control básico del tiempo

Alternativa 2 (actualizar solo cada FRAME\_RATE):

```
uint32_t startTime, frameTime;
startTime = SDL_GetTicks();

while (!exit) { // Bucle del juego
    handleEvents();
    // Tiempo desde última actualización
    frameTime = SDL_GetTicks() - startTime;
    if (frameTime ≥ FRAME_RATE) {
        update(); // Actualiza el estado de todos los objetos del juego
        startTime = SDL_GetTicks();
    }
    render(); // Renderiza todos los objetos del juego
}
```

Mayor control y precisión aunque consume más recursos