UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Adaptación de un ejemplo de uso nativo de *sprites* de 3DS a Switch utilizando SDL

Apellidos, nombre	**Agustí i Melchor, Manuel** (magusti@disca.upv.es)
Departamento	**Departamento de Informática de Sistemas y Computadores (DISCA)**
Centro	Universitat Politècnica de València

1 Resumen de las ideas clave

En el SDK no oficial (*homebrew*) de *devkitPro* [1] para el desarrollo en plataforma nativa de la **3DS**, encontramos los recursos para desarrollar en esta plataforma, de forma que podemos acceder al hardware que nos proporciona la plataforma y, por tanto, poder implementar sobre ella las acciones lo más optimizadas posible. De hecho, el API de Citro3D y Citro2D permiten realizar las operaciones gráficas utilizando la GPU de la consola y, con ello, la aceleración hardware que esta ofrece. Eso sí, trabajar a este bajo nivel de la plataforma de un computador, exige un conocimiento profundo de sus características y hace costoso el reimplementar una aplicación en otra plataforma de cara a plantear la portabilidad de una aplicación, como un videojuego. Una de las aplicaciones de ejemplo más interesantes que proporciona el SDK como base para desarrollos basado en el uso de **sprite** es el ejemplo *gpusprites* que podemos ver en la Figura 1.

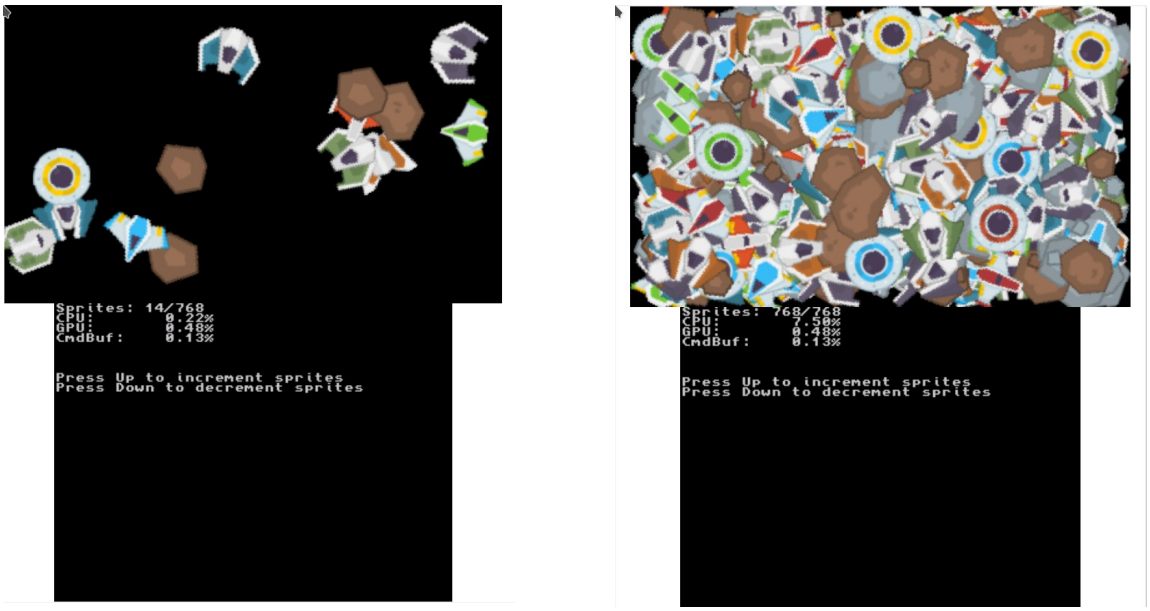


Figura 1: Resultado de la ejecución del ejemplo para 3DS gpusprites en Citra: dos instantes extremos de la ejecución.

Dada la potencia de cálculo de la plataforma **Switch**, sobre ella no existe un ejemplo similar al de *gpusprites* de 3DS. En lugar de desarrollar un API específico de dibujado en pantalla, los desarrolladores del SDK no oficial promueven el uso de bibliotecas de alto nivel que ya existen en el contexto del desarrollo para el computador de escritorio. Estas se encuentran agrupadas en un paquete denominado PORTLIBS y son un excelente punto de partida para plantearse la portabilidad de un desarrollo entre diferentes plataformas, ya que al estar disponibles en diferentes plataformas los cambios son mínimos para hacerlas funcionar en otra plataforma diferente de la de partida.

2 Objetivos

Una vez que el lector haya revisado este artículo con detenimiento y explore el código que se adjunta, dispondrá de una referencia para implementar el uso de *sprites* en sus aplicaciones con el uso de la biblioteca SDL. En particular, será capaz de:

- Describir qué es SDL.
- Describir cómo se ha reconstruido un ejemplo de *sprites* que sea equivalente al nativo sobre la plataforma 3DS.

Para no extender la longitud del artículo mostrando todo el código realizado, se ha creado un repositorio en GitHub [3] donde se alojarán los ejemplos de código que se referencian es este artículo. El resto de este documento se centrará en describir cómo se ha planteado la versión del ejemplo original en la 3DS con Citro2D a *Switch* con SDL.

3 Introducción

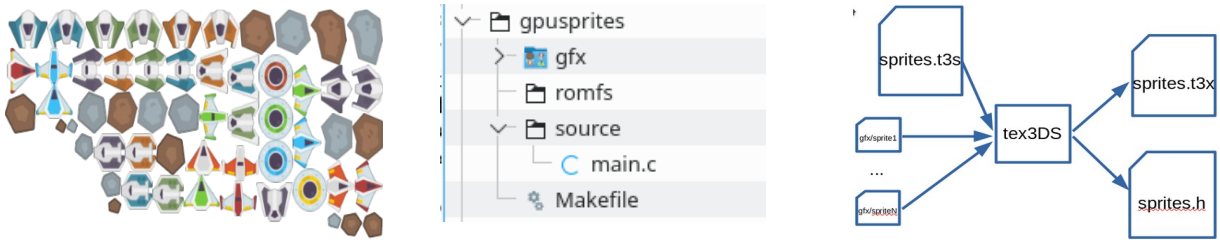
Hemos señalado ya que, entre las posibilidades de desarrollo que nos ofrece el SDK no oficial (*homebrew*) para el desarrollo de aplicaciones en plataforma Nintendo 3DS (N. 3DS o simplemente 3DS) y Nintendo Switch (Switch), se encuentra el uso de la biblioteca de funciones SDL [2], que nos brinda un API multiplataforma para la implementación de aplicaciones de carácter multimedia y con especial atención a facilitar la portabilidad del código realizado. Habitualmente se habla de las posibilidades de portabilidad de SDL para aplicaciones en el contexto del computador de escritorio. También es posible realizar aplicaciones en las plataformas móviles más habituales. En nuestro caso, nos interesa comprobar que es posible para videoconsolas, en particular para Nintendo *Switch*.

Como ejemplos de uso básico de SDL en plataforma *Switch* se pueden consultar los ejemplos de desarrollo *[1]* que acompañan al SDK *homebrew*. Ahí veremos cómo se puede pintar directamente en la memoria de vídeo (*framebuffer*) de la consola o utilizar imágenes (en formato de mapas de bits) que un gestor se encarga de actualizar y refrescar, además de permitir operaciones de mayor nivel de abstracción como movimiento, rotaciones, escalado, etc. A los objetos que gestionan este segundo enfoque de dibujo se les denomina *sprites*.

3.1 Ejemplo inicial: *gpusprites* para 3DS

Las aplicaciones basadas en el uso de sprites, los suelen cargar todos en una sola imagen y de esta van escogiendo el “trocito” que necesitan. A estas imágenes se las denomina spritesheets o texture atlas, son ficheros de tipo imagen en mapa de bits que agrupan, a conveniencia y criterio del desarrollador, las imágenes que se utilizan en un nivel, en un personaje o cualquier otro espacio de una aplicación.

El ejemplo sobre 3DS utiliza las funciones de Citro2D para la gestión de sprites en la plataforma 3DS. La implementación de estas funciones está hecha con Citro3D, lo que permite acceder a la aceleración hardware que esta plataforma ofrece. El proyecto se basa en el uso de una colección de imágenes individuales para ser utilizadas como sprites dentro de la aplicación. Estas imágenes, originalmente en el directorio gfx del proyecto, están guardadas en formato PNG, con tamaños diferentes entre 16x15 y 50x38 píxeles, véase la Figura 2a, junto a un fichero de código C (en el subdirectorio source) y el fichero Makefile para compilar el proyecto, Figura 2b.



a) *b)* *c)*

Figura 2: Generación automática del spritesheet en 3DS: (a) colección de imgenes, (b) estructura del proyecto y (c) diagrama de bloques de la generación del spritesheet mediante la utilidad tex3ds.

Para facilitar la gestión de las imágenes, el *spritesheet*, dado que son cerca de sesenta imágenes, se utiliza un fichero de texto (T3S) a modo de catálogo que contiene la lista de imágenes:

```
--atlas -f rgba8888 -z auto
enemyBlack1.png
...
ufoRed.png
ufoYellow.png
```

La primera de las líneas del fichero T3S es la que instruye a la utilidad *tex3ds* para que utilice 32-bits por pixel (RGBA de 8 bits por componente) y compresión. A partir de este fichero T3S, la utilidad *tex3ds*, Figura 2c, genera un *spritesheet*, *que* cargará en la aplicación con una sola instrucción y dispondrá así de todos los gráficos en memoria:

```
spriteSheet = C2D_SpriteSheetLoad("romfs:/gfx/sprites.t3x");
```

Para referenciarlos desde la aplicación, tex3ds genera un fichero del mismo nombre y extensión H para tener una lista de identificadores sobre las componentes del *spritesheet*. Estos ficheros generados se encuentran en el directorio *build* que se crea automáticamente durante la compilación y en *romfs*.

Ejercicio. Sobreescriba los dos primeros PNG de la lista de sprites.t3s, en el ejemplo de gpusprites, para comprobar que, al reconstruir el proyecto, estas nuevas imágenes aparecerán en el lugar de las anteriores. Baje el número de sprites en pantalla para observarlo con facilidad.

Para realizar una versión compatible con el ejemplo de *gpusprites* de 3DS, vamos a ver una reimplementación del ejemplo de *gpusprites* para Switch con SDL. Vamos a reutilizar la estructura de directorios del proyecto 3DS, como se mostraba en la Figura 2b. Además, de esta versión para *Switch*, se ha obtenido una versión para el computador de escritorio con mínimas variaciones del código.

4 Desarrollo

En *Switch* no tenemos utilidades para construir los *spritesheets*, pero a partir de la lista de ficheros de imágenes, podemos generar una lista (llistaSprites.h, véase Listado 1) con la que realizar una acción equivalente de carga de los recursos en memoria[1].

[1] Es posible automatizar esta acción, pero por brevedad de la exposición, lo dejamos para la curiosidad del lector.

```
1.  #ifndef LLISTASPRITES
2.
3.  #define LLISTASPRITES
4.
5.  #define NSPRITES 56
6.
7.
8.  char *llistaSprites[NSPRITES] = {
9.  "gfx/enemyBlack1.png",
10. // resto de rutas a ficheros, obviado por brevedad...
11. "gfx/ufoYellow.png"};
12. #endif
```

Listado 1: llistaSprites.h

La versión para *Switch* los irá cargando individualmente y los añade a la aplicación, cargándolos uno a uno y reutilizará las estructuras de datos de la versión para 3DS para poder comparar su funcionalidad (véase el Listado 2): los nombres de las estructuras de datos se han mantenido. Un tipo *Sprite* define un objeto que tiene (líneas 14 a 17) un componente visible y dos velocidades, una en cada eje. El componente básico *C2D_Sprite* (líneas 10 a 13) se ha modificado el tipo "imagen" a *SDL_Texture* que es como lo define SDL y se complementa con *C2D_DrawParams*, que (líneas 1 a 9) es el que guarda la información de geometría gráfica del *sprite* y su posición en pantalla.

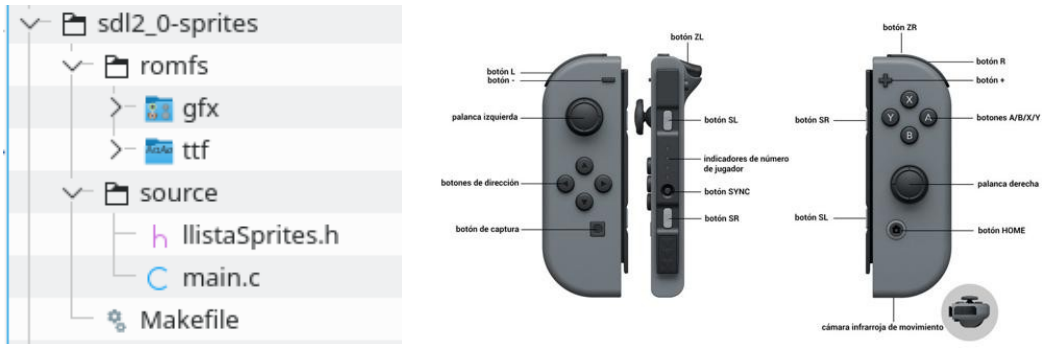
```
1.  typedef struct {
2.      struct {
3.          float   x, y, w, h;
4.      }       pos;
5.      struct {
6.          float   x, y;
7.      }       center;
8.      float   depth, angle;
9.  }  C2D_DrawParams;
10. typedef struct {
11.     SDL_Texture     *image;
12.     C2D_DrawParams    params;
13. } C2D_Sprite;
14. typedef struct {
15.     C2D_Sprite spr;
16.     float dx, dy;   // velocity
17. } Sprite;
```

Listado 2: Estructuras de datos reutilizadas del ejemplo original de spusprites para 3DS.

El resultado que se obtiene debe ser similar en aspecto y tiempos de uso y, como el original, habrá de permitir la interacción del usuario para que pueda cambiar el número de *sprites* en pantalla con el uso de los botones de flecha arriba y abajo.

4.1 *Sprites* con SDL: *sdlsprites*, una versión de *gpusprites* para *Switch* con SDL

Aquí hemos reimplementado el proyecto utilizando SDL, reutilizando la estructura de directorios del proyecto 3DS, como se muestra en la Figura 3a, las estructuras de datos (Listado 2) y adaptando la funcionalidad del ejemplo de partida a la botonera física de la nueva plataforma, Figura 3b.



a) *b)*

Figura 3: Estructura del ejemplo para Switch: (a) estructura del proyecto y (b) disponibilidad de funciones en los mandos de la Switch.

No vamos a ver línea a línea los cambios realizados. por brevedad en la exposición. En su lugar vamos a destacar los cambios debidos al uso de SDL y de la gestión de eventos de entrada en *Switch*.

En lo que respecta al uso de SDL se ha tenido que modificar las operaciones gráficas, por lo que vemos en el Listado 4 destacadas en negrita las funciones auxiliares modificadas y, también en negrita, las líneas que ha habido que modificar, podrá observar el prefijo "SDL_" para localizar las instrucciones de SDL, que comparado con el original se corresponden una a una con la utilizada en la versión nativa de 3DS. Como puede comprobar: *moveSprites* no necesita modificarse, *deinitSprites* solo en la línea 4 e *initSprites* en cuatro instrucciones para hacer la carga de la imagen y la conversión al tipo *Surface* que usa SDL internamente.

En el Listado 4, aparece una función nueva que agrupa un alto número de funciones de SDL que es *render_text*. Esta es una función de conveniencia, esto es, nos sirve para agrupar un número de instrucciones que vamos a repetir en tres ocasiones en el código para pintar texto con SDL, que es totalmente diferente de cómo se realiza en la versión nativa para 3DS.

La función main está repartida entre los listado 5 y 6. Aquí aparece el grueso de instrucciones que ha sido necesario cambiar y que se agrupan en dos bloques:

- Instrucciones para el desarrollo del interfaz gráfico con SDL.
  - Entre las líneas 74 a la 85 del Listado 5, donde se puede ver la inicialización de SDL.
  - Entre las líneas 111 a 139 del Listado 6 que actualizan las posiciones de los *sprites* en pantalla.
  - Y las líneas 143 a 151, del Listado 6, que liberan los recursos de SDL que se han ido obteniendo.
- Instrucciones relativas a la gestión de eventos de la *Switch*.
  - Entre las líneas 96 y 98 del Listado 5 se inicializa la gestión de la botonera del dispositivo.
  - Entre las líneas 100 a 110 del Listado 6 se procesan los eventos para detectar la pulsación del botón '+' y terminar o una de la teclas de subir o bajar para aumentar o disminuir, respectivamente, el número de *sprites* en pantalla.

```
1.  static void deinitSprites() {
2.      for (size_t i = 0; i < MAX_SPRITES; i++)
3.          SDL_DestroyTexture( sprites[i].spr.image );
4.  }
5.  static void initSprites() {
6.      SDL_Surface      *unaImage;
7.      size_t numImages = NSPRITES;
8.      for (size_t i = 0; i < MAX_SPRITES; i++)  {
9.          Sprite* sprite = &sprites[i];
10.         unaImage = IMG_Load( llistaSprites[rand() % numImages] );
11.         if (unaImage) {
12.             sprite->spr.params.pos.w = unaImage->w;
13.             sprite->spr.params.pos.h = unaImage->h;
14.             sprite->spr.image = SDL_CreateTextureFromSurface(renderer,
15.                                                              unaImage);
16.         }
17.         sprite->spr.params.center.x = unaImage->w/2; //0.5f;
18.         sprite->spr.params.center.y = unaImage->h/2; //0.5f;
19.         sprite->spr.params.pos.x = rand() % SCREEN_WIDTH;
20.         sprite->spr.params.pos.y = rand() % SCREEN_HEIGHT;
21.         sprite->spr.params.angle = rand()/(float)RAND_MAX;
22.         sprite->dx = rand()*4.0f/RAND_MAX - 2.0f;
23.         sprite->dy = rand()*4.0f/RAND_MAX - 2.0f;
24.         SDL_FreeSurface( unaImage );
25.     }
26. }
27. static void moveSprites() {
28.     Sprite *sprite;
29.     for (size_t i = 0; i < numSprites; i++)     {
30.         sprite = &sprites[i];
31.         sprite->spr.params.pos.x =
32.                                 sprite->spr.params.pos.x + sprite->dx;
33.         sprite->spr.params.pos.y =
34.                                 sprite->spr.params.pos.y + sprite->dy;
35.         if (sprite->spr.params.angle < 350 )
36.              sprite->spr.params.angle += 10;
37.         else
38.              sprite->spr.params.angle = 0;
39.         if ((sprite->spr.params.pos.x < sprite->spr.params.pos.w / 2.0f
40.               && sprite->dx < 0.0f) || (sprite->spr.params.pos.x >
41. (SCREEN_WIDTH-(sprite->spr.params.pos.w / 2.0f)) && sprite->dx > 0.0f))
42.             sprite->dx = -sprite->dx;
43.         if ((sprite->spr.params.pos.y < sprite->spr.params.pos.h / 2.0f
44.               && sprite->dy < 0.0f) || (sprite->spr.params.pos.y >
45. (SCREEN_HEIGHT-(sprite->spr.params.pos.h / 2.0f)) && sprite->dy >0.0f))
46.             sprite->dy = -sprite->dy;
47.     }
48. }
    ...
```

Listado 3: llistaSpritesmain.c (1ª parte).h

```
...
static void deinitSprites() {
    for (size_t i = 0; i < MAX_SPRITES; i++)
        SDL_DestroyTexture( sprites[i].spr.image );
}
static void initSprites() {
    SDL_Surface      *unaImage;
    size_t numImages = NSPRITES;
    for (size_t i = 0; i < MAX_SPRITES; i++)  {
        Sprite* sprite = &sprites[i];
        unaImage = IMG_Load( llistaSprites[rand() % numImages] );
        if (unaImage) {
            sprite->spr.params.pos.w = unaImage->w;
            sprite->spr.params.pos.h = unaImage->h;
            sprite->spr.image = SDL_CreateTextureFromSurface(renderer,
                                                             unaImage);
        }
        sprite->spr.params.center.x = unaImage->w/2; //0.5f;
        sprite->spr.params.center.y = unaImage->h/2; //0.5f;
        sprite->spr.params.pos.x = rand() % SCREEN_WIDTH;
        sprite->spr.params.pos.y = rand() % SCREEN_HEIGHT;
        sprite->spr.params.angle = rand()/(float)RAND_MAX;
        sprite->dx = rand()*4.0f/RAND_MAX - 2.0f;
        sprite->dy = rand()*4.0f/RAND_MAX - 2.0f;
        SDL_FreeSurface( unaImage );
    }
}
static void moveSprites() {
    Sprite *sprite;
    for (size_t i = 0; i < numSprites; i++)     {
        sprite = &sprites[i];
        sprite->spr.params.pos.x =
                              sprite->spr.params.pos.x + sprite->dx;
        sprite->spr.params.pos.y =
                              sprite->spr.params.pos.y + sprite->dy;
        if (sprite->spr.params.angle < 350 )
             sprite->spr.params.angle += 10;
        else
             sprite->spr.params.angle = 0;
        if ((sprite->spr.params.pos.x < sprite->spr.params.pos.w / 2.0f
              && sprite->dx < 0.0f) || (sprite->spr.params.pos.x >
(SCREEN_WIDTH-(sprite->spr.params.pos.w / 2.0f)) && sprite->dx > 0.0f))
            sprite->dx = -sprite->dx;
        if ((sprite->spr.params.pos.y < sprite->spr.params.pos.h / 2.0f
              && sprite->dy < 0.0f) || (sprite->spr.params.pos.y >
(SCREEN_HEIGHT-(sprite->spr.params.pos.h / 2.0f)) && sprite->dy >0.0f))
            sprite->dy = -sprite->dy;
    }
}
...
```

Listado 4: main.c (1ª parte).

```
...
void render_text(const char* text, TTF_Font *font,
            SDL_Color color, SDL_Rect *rect, SDL_Renderer *renderer ){
    SDL_Surface *mensatges_surf;
    SDL_Texture *mensatges_tex = NULL;
    mensatges_surf = TTF_RenderText_Solid(font, text, color);
    rect->w = mensatges_surf->w;
    rect->h = mensatges_surf->h;
    SDL_SetRenderDrawColor(renderer, 0, 0, 0, 255);
    SDL_RenderFillRect(renderer, rect);
    mensatges_tex = SDL_CreateTextureFromSurface(renderer,
                                                 mensatges_surf);
    SDL_RenderCopy(renderer, mensatges_tex, NULL, rect);
    SDL_DestroyTexture( mensatges_tex );
}
int main(int argc, char* argv[]) {
    int exit_requested = 0;
    Uint32 start_time, stop_time, nCuadres, nTics;
    clock_t start, end;
    double cpu_time_used;
    int wait = 25;

    srand(time(NULL));
    romfsInit();
    chdir("romfs:/");

    SDL_Init(SDL_INIT_VIDEO|SDL_INIT_TIMER);
    SDL_ShowCursor( 0 );
    window = SDL_CreateWindow("SDL 2.0 sprites en Switch",
                      SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED,
                      SCREEN_WIDTH, SCREEN_HEIGHT, SDL_WINDOW_SHOWN);
    renderer = SDL_CreateRenderer(window, -1,
                                  SDL_RENDERER_ACCELERATED);
    SDL_SetRenderDrawColor(renderer, 0, 0, 0, 255); // RGBA
    SDL_RenderClear( renderer );
    IMG_Init(IMG_INIT_PNG);
    TTF_Init();
    font = TTF_OpenFont("ttf/LeroyLetteringLightBeta01.ttf", 36);
    mensatges_rect.x = 0;
    mensatges_rect.y = SCREEN_HEIGHT - tamanyLletra;
    mensatges_rect.w = 0;     mensatges_rect.h = 0;
    mensatges_rect2.x = 0;
    mensatges_rect2.y = SCREEN_HEIGHT - (tamanyLletra *2);
    mensatges_rect2.w = 0;    mensatges_rect2.h = 0;
    render_text("SDL 2.0 sprites!", font, colors[1],
                &mensatges_rect, renderer);
    initSprites();

    padConfigureInput(1, HidNpadStyleSet_NpadStandard);
    PadState pad;
    padInitializeDefault(&pad);

...
```

Listado 5: main.c (2ª parte).

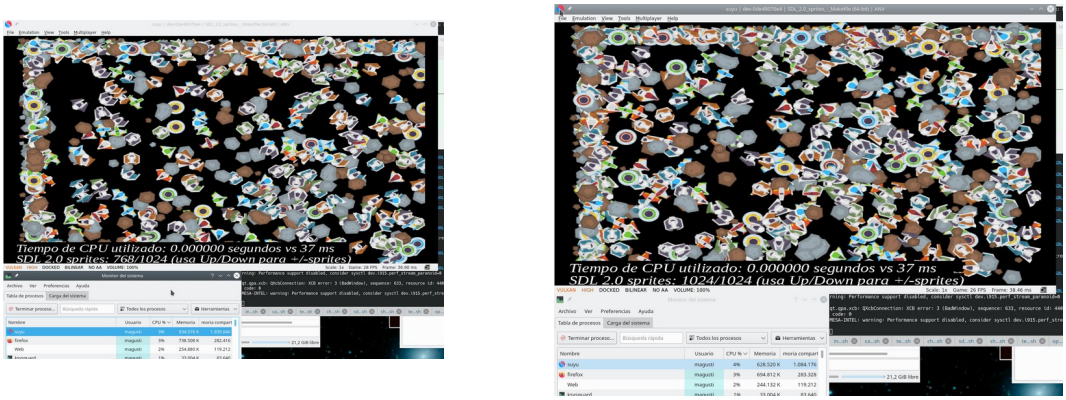
```
…
while (!exit_requested && appletMainLoop() ) {
        padUpdate(&pad);
        u64 kDown = padGetButtonsDown(&pad);
        u64 kHeld = padGetButtons(&pad);
        if (kDown & HidNpadButton_Plus)
            break; // break in order to return to hbmenu
            if ((kHeld & HidNpadButton_AnyUp)
                 && numSprites < MAX_SPRITES)
                numSprites++;
            if ((kHeld & HidNpadButton_AnyDown) && numSprites > 1)
                numSprites--;
            SDL_SetRenderDrawColor(renderer, 0, 0, 0, 255); // RGBA
            SDL_RenderClear( renderer );
            moveSprites();
            for (size_t i = 0; i < numSprites; i ++) {// Render

                SDL_Rect pos = { 0, 0, 0, 0 };
                Sprite* sprite = &sprites[i];
                pos.x = sprite->spr.params.pos.x;
                pos.y = sprite->spr.params.pos.y;
                pos.w = sprite->spr.params.pos.w;
                pos.h = sprite->spr.params.pos.h;
                if (sprite->spr.image) {
                    SDL_RenderCopyEx(renderer, sprite->spr.image, NULL,
                 &pos, sprite->spr.params.angle, NULL, SDL_FLIP_NONE);
                }
            } // for (size_t i = 0; i < numSprites; i ++) {
       char textMensatge[1024];
            sprintf(textMensatge, "SDL 2.0 sprites: %lu/%d (usa Up/Down
para +/-sprites)", numSprites, MAX_SPRITES );
            SDL_Surface *mensatges_surf = NULL;
            mensatges_surf = render_text(textMensatge, font, colors[1],
                                         &mensatges_rect, renderer);
            if ( mensatges_tex ) SDL_DestroyTexture( mensatges_tex );
            mensatges_tex = SDL_CreateTextureFromSurface(renderer,
                                                    mensatges_surf);
            SDL_RenderCopy(renderer, mensatges_tex, NULL,
                           &mensatges_rect);
            SDL_RenderPresent(renderer);
            SDL_Delay(wait);
    }//      while (!exit_requested && appletMainLoop() )

    deinitSprites();
    TTF_CloseFont(font);
    if ( mensatges_surf ) SDL_FreeSurface( mensatges_surf );
    if ( mensatges_tex ) SDL_DestroyTexture( mensatges_tex );
    if (pantalla) SDL_DestroyTexture( pantalla );
    IMG_Quit();
    TTF_Quit();
    SDL_DestroyRenderer(renderer);
    SDL_DestroyWindow( window );
    SDL_Quit();
    return 0;
}
```

Listado 6: main.c (3ª parte).

Ejercicio: Vamos a comprobar que todavía se puede subir más el número de *sprites*. Localice la constante *MAX_SPRITES* y aumente su valor, progresivamente, a 1024, 2048 ó 4096. La Figura 4 muestra que se obtiene un 5% de uso de CPU con 768 *sprites* cargados. ¿Nota una variación importante en el rendimiento?



a) *b)*

Figura 4: versión sdl2_0-sprites para Switch con (a) 768 y (b) 1024 sprites en uso.

Ejercicio. Aprovechando lo que se muestra aquí (y el código de los ejemplos [3]) se puede ampliar este ejemplo, le propongo: añadir un imagen de fondo e incluir la gestión de eventos para dirigir uno de los objetos (una nave) y que vaya recolectando meteoritos.

5 Conclusión y cierre

A lo largo de este objeto de aprendizaje hemos visto cómo hemos llevado a cabo una versión de una aplicación nativa para 3DS para la plataforma Switch. Para ello hemos explorado el uso de la biblioteca SDL que nos ha permitido reproducir el interfaz gráfico de la aplicación original y con los mismos archivos utilizados en aquella. Hemos conservado las estructuras de datos y la gestión de eventos nativa a cada plataforma para facilitar su comparación.

En el repositorio de Github creado para este artículo [3] se puede encontrar el fichero *sdl2_switchsprites.zip* que contiene el código del proyecto para la plataforma de la videoconsola *Switch*. El resultado del ejemplo original para 3DS es compatible con el obtenido, compare la Figura 1 y Figura 4 para observarlo.

Otro tema que se ha quedado en el tintero es hablar de portabilidad: que necesidades hay de realizar modificaciones y cómo recompilar el código para ejecutarlo en plataformas como la 3DS o la del computador de escritorio (PC). Pero eso será tema de otro artículo.

6 Bibliografía y referencias

[1] devkitPro. <https://devkitpro.org/>.

[2] SDL. Sitio web. <https://www.libsdl.org/>.

[3] Repositorio de los ejemplos de este artículo <https://github.com/magusti/SDL/sdlsprites/>.