Arquitectura y Entornos de desarrollo para Videoconsolas

Práctica 3
Uso de la imagen en la videoconsola 3DS

M. Agustí

AEV, 2019 / 2020

1 Las imágenes han sido extraídas de la instalación de DevkitPro y de la ejecución de los ejemplos para la plataforma 3DS.

La práctica se podrá realizar en entorno GNU/Linux en el laboratorio. Vamos a abordar el uso de la imagen en la videoconsola de estudio, tanto en lo que se refiere a las posibilidades de mostrar una imagen estática en 2D o 3D, así como del soporte a la animación y la interacción.
Recuerde que ha de guardar los resultados de sus acciones a lo largo de las prácticas para confeccionar el portafolio de prácticas, que es la forma en que se evaluará las prácticas de la asignatura. No descuide hacer copias de lo que necesite del laboratorio en su espacio del servidor de la asignatura.

1 Introducción

Trabajaremos a partir de algunos de los ejemplos de código disponibles en la documentación del SDK de *devkitPro* y con la documentación disponible: *CTRU* [1], *Citro2D* [2], para *Citro3D* encontrará el archivo *citro3D_doc.zip* que se ha generado a partir del código fuente de esta librería. Tenga a mano los ejemplos que en la primera práctica se copiaron en el espacio del usuario[1]. Recuerde que esto era necesario para poder compilarlos, pero que puede llevarse el directorio base del proyecto a cualquier directorio de su espacio de usuario. Las rutas a los ejemplos asumen que se conoce el directorio base en que cada uno dejó esta copia de los ejemplos y se indican con la ruta relativa al mismo.

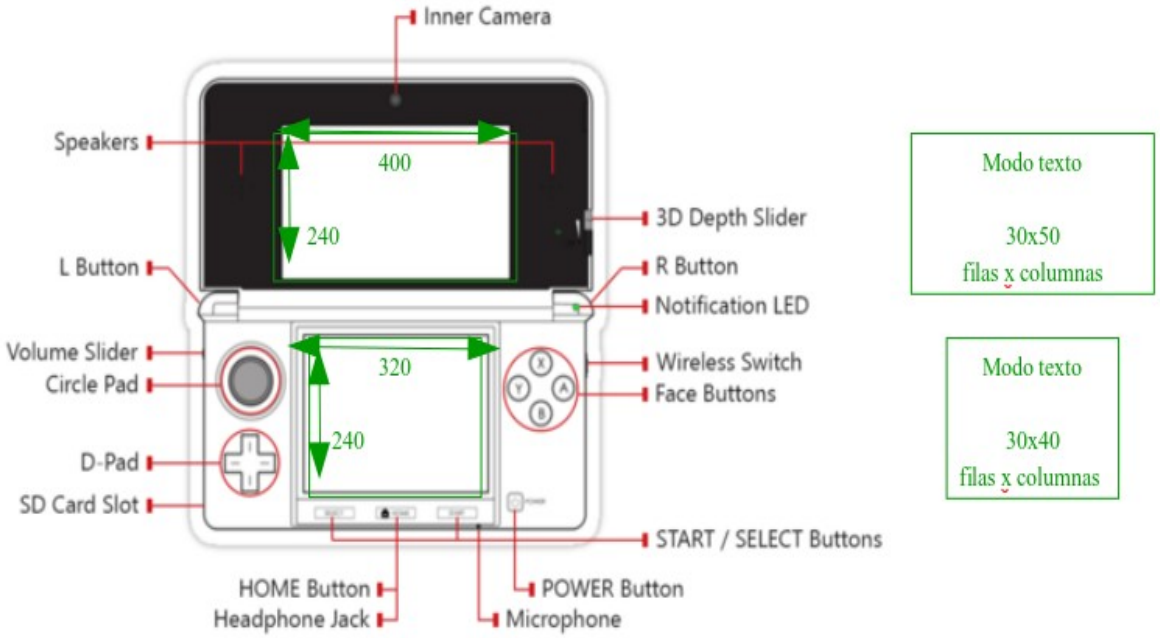


Figura 1: Elementos del interfaz de la consola 3DS y resolución de las pantallas en píxeles y caracteres. Imagen extraída de <https://en.wikipedia.org/wiki/Nintendo_3DS>.

En concreto, en esta práctica nos vamos a referir a los ejemplos del SDK relacionados con el tema gráfico: desde la inicialización del modo gráfico de las pantallas, al acceso directo a la memoria de vídeo (con *libctru*), pasando por las primitivas de dibujo 2D (con libcitro2D) y terminando con una breve introducción al uso de las primitivas para 3D, a través de la *libctro3d*, aprovechando la aceleración que ofrece la GPU de la plataforma. Vamos a ver de estos. los relativos a los apartados mencionados dentro del subdirectorio *graphics*. Asumiremos que ya ha instalado las herramientas necesarias en las sesiones previas: *devkitPro* para 3DS y el emulador *Citra*.

1 También pueden descargar los ejemplos directamente desde el repositorio en *GitHub* desde <https://github.com/devkitPro/3ds-examples>, en cuyo caso habrá de descargue el ZIP y descomprímalo en un directorio de su elección. **En el curso 2019/2020 ha sido necesario descargarlos manualmente.**

2 Mapas de bits: acceso directo a la memoria de vídeo

Los aspectos básicos de acceso al hardware de la consola 3DS están accesibles en nuestro SDK a través del interfaz que ofrece CTRU [1]. La utilizamos ya en la práctica relativa al acceso al hardware y, ahora, volvemos sobre ella para fijar nuestra atención en cómo se puede inicializar el modo gráfico de ambas pantallas, así como comprobar que se puede trabajar en modo de acceso directo a la memoria de vídeo,

Para experimentar con el acceso a la memoria de vídeo empezaremos por ver el ejemplo de 3DS dentro de *graphics/bitmap/24bit-color*, que se muestra en la Figura 2. **Si estás trabajando en la plataforma Windows, en el Anexo II verás que hay un problema en su compilación en este sistema operativo y cómo resolverlo.**

Figura 2: Captura de la ejecución del ejemplo de 3DS dentro de gráphics/bitmap/24bit-color.

Este ejemplo muestra la pantalla superior inicializada en modo "consola" para escribir texto y la inferior en modo gráfico, que utiliza acceso directo a la memoria de vídeo para cargar un fichero PNG que está en el subdirectorio *gfx* del proyecto. Lo hace en el momento de copiar la imagen con:

```
memcpy(fb, brew_bgr, brew_bgr_size);  //Copy our image in the bottom screen's frame buffer
```

El fichero *README.md* que acompaña al ejemplo describe que para cambiar la imagen que se muestra en el fichero hay que reordenar los canales RGB de la imagen y trasponer las filas por columnas en lo que parece que es un proceso previo y manual. Si revisa el *Makefile* comprobará que al fichero PNG original se le aplica una rotación de 90º y una reordenación de los canales de la imagen (véase la Figura 3), en el proceso de incorporación al ejecutable final. El actual contenido del fichero *Makefile* hace que esta tarea sea realizada de manera automática, así que solo es

necesario sobreeescribir la imagen con una propia y reconstruir el proyecto. Observe el contenido del *Makefile* y verá una regla que utiliza la orden *convert*[1] para conseguir este propósito. La acción que realiza se puede hacer manualmente para comprobar lo que hace:

```
$ convert gfx/brew.png -rotate 90 brew.bgr
$ convert  -size 240x320  -depth 8 RGB:brew.bgr brew_bgr.png
$ file  gfx/brew.png brew.bgr.png
gfx/brew.png: PNG image data, 320 x 240, 8-bit/color RGBA, non-interlaced
brew.bgr.png: PNG image data, 240 x 320, 8-bit/color RGB, non-interlace
```

Ejercicio 1: Compruebe que puede sobreescribir el fichero PNG mencionado, dentro del subdiretorio *gfx*, con cualquier fichero gráfico de su elección y que, al reconstruir el proyecto, esa nueva imagen aparece en la pantalla inferior.

a) b)

Figura 3: Visualización de lo cambios aplicados a la imagen del ejemplo graphics/bitmap/24bit-color por el Makefile: (a) Original y (b) la obtenida para incluirla en el proyecto.

2.1 Ejemplo de acceso directo a memoria

Volviendo sobre el contenido del fichero *README.md* mencionado, este ya hemos dicho que justifica estas operaciones sobre la imagen con: "As you can see [...] the image is clockwise rotated by 90 degrees and its B and R channels are swapped. The first operation is done because the 3DS screens are actually portrait screens rotated by 90 degrees (in a counter-clockwise direction), while the second one is done because the 3DS screens' framebuffers have a BGR888 color format, by default".

Vamos a profundizar en este aspecto con un ejemplo que está junto a este boletín y que se denomina *accesoMemoriaVideo*. Este ejemplo accederá a los píxeles de cada pantalla para asignarles un mismo valor a todos, en función de la pantalla, para facilitar su reconocimiento durante la ejecución. Compárelo con el anterior para observar las diferencias.

1 La orden convert del paquete *Image Magick* <https://imagemagick.org/index.php> sustituye en el SDK para 3DS a la utilidad GRIT que se ha venido utilizado en el caso de la NDS.

Podrá observar que, en este caso, el modo gráfico de las dos pantallas se inicializa con la función gfxInit, que tiene como definición

```
void gfxInit(   GSPGPU_FramebufferFormats  topFormat,
                GSPGPU_FramebufferFormats  bottomFormat, bool vrambuffers);
```

habitualmente hacemos uso de *gfxInitDefault* que equivale a

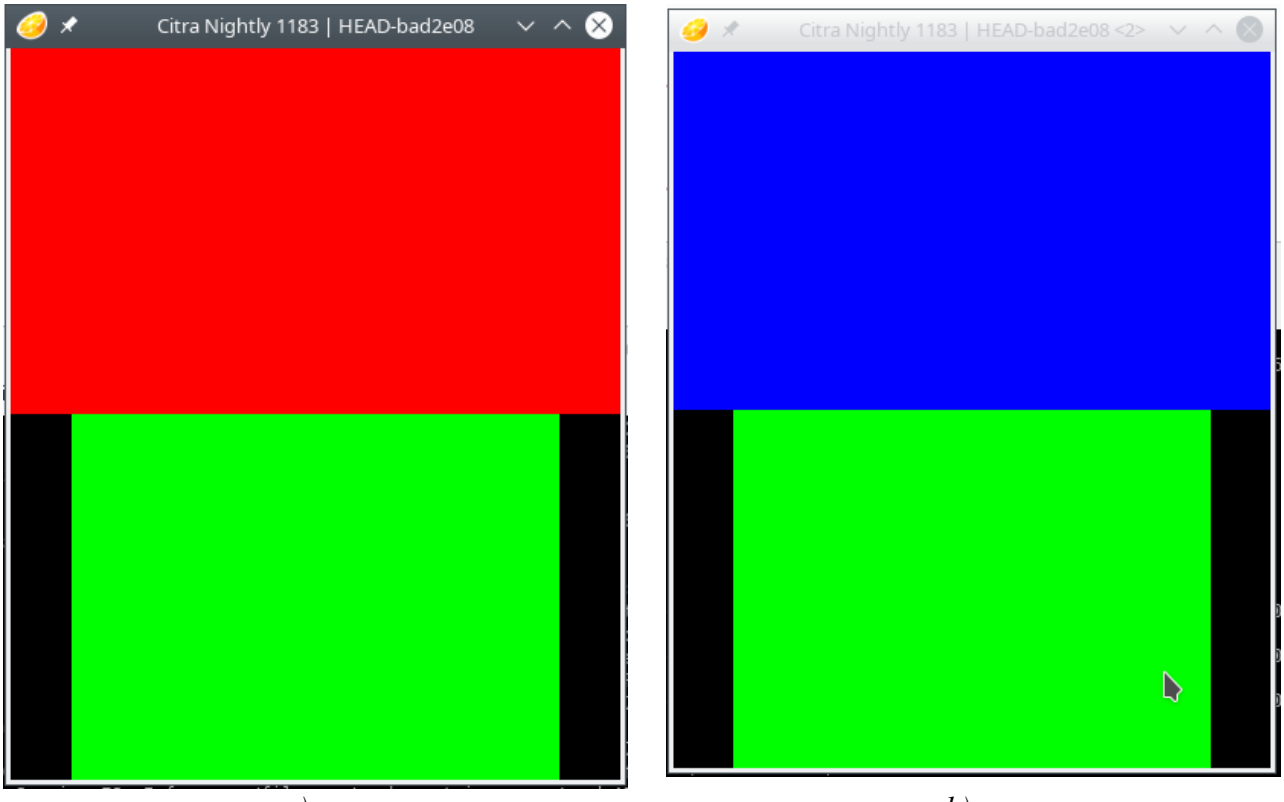
```
gfxInit(GSP_BGR8_OES,GSP_BGR8_OES,false);
```

Sin embargo, en el ejemplo actual se ha instanciado como

```
gfxInit(GSP_RGBA8_OES, GSP_RGBA8_OES, false);
```

teniendo en cuenta que, como dice en la documentación, existen diferentes valores aceptados por esta plataforma para *FramebufferFormats* y que representan cuántos bits se utilizan para representar el color de cada punto, de acuerdo con:

- GSP_RGBA8_OES → 4 bytes,
- GSP_BGR8_OES → 3 bytes,
- GSP_RGB565_OES → 2 bytes,
- GSP_RGB5_A1_OES → 2 bytes,
- y GSP_RGBA4_OES → 2 bytes.



a) *b)*

Figura 4: Disposición de la pantalla sup e inferior, en función de si se ha habilitado (a) o no (b) el efecto estereoscópico.

A la pantalla inferior (*BOTTOM*) se le asigna el color verde. Como la pantalla superior es

estereoscópica, en ella se entremezclan dos áreas (derecha e izquierda) en función del valor del control deslizante de la consola real. Este control no está disponible en el emulador. Pero sí lo están las dos "pantallas" que permiten componer la imagen que se muestra en la parte superior para cada ojo, por lo que ser verá que en el código se accede a *GFX_TOP + GFX_LEFT* o *GFX_TOP + GFX_RIGHT* y se inicializan a rojo y azul, respectivamente. La Figura 4 muestra el resultado de ejecutar esta aplicación, según se haya habilitado o no el efecto esteoscópico con:

```
gfxSet3D(true);
```

Ejercicio 2: Compruebe en el código entregado que los *buffer* se crean con tipo GSP_RGBA8_OES (esto indicará que se utilizan 32 bits para cada píxel, 8 por cada componente de RGB, más el canal alfa de transparencia) y que el *buffer* está rotado respecto a las coordenadas originales. Anote cómo se ve este último aspecto en el código que inicializa los *buffers* de memoria.

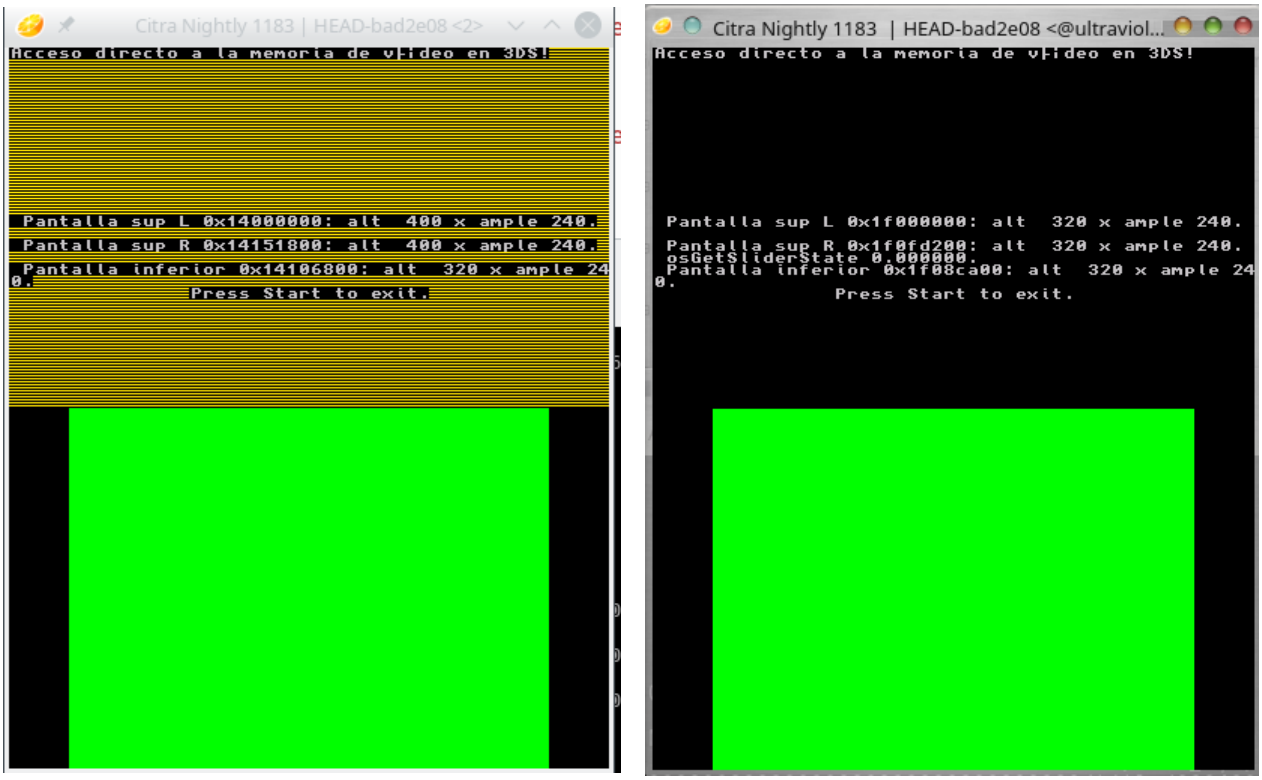


Figura 5: Si se reconfigura el modos gráficos de una pantalla en tiempo de ejecución a otro, hay que tener cuenta si estos son incompatibles: (a) en caso de que lo sea se obtiene una pantalla con errores, en caso contrario (b) la ejecución continua sin artefactos.

Estos modos gráficos no son compatibles con el modo "texto", por lo que si en algún momento posterior a la configuración del modo gráfico ejecutamos la habitual

```
consoleInit(GFX_TOP, NULL);
```

veremos el resultado que se muestra en la Figura 5, porque no son compatibles el modo RBGA8 (32 bits por píxel) y el modo BGR8 (24 bits por píxel). Así que hay que decidir si se funciona en un modo o en otro, o al cambiarlo, poner la pantalla a un color de fondo para borrar lo anterior.

La función *consoleInit* borra la pantalla, así que al escribir texto se elimina lo que se haya escrito

anteriormente.

3 Gráficas en 2D con Citro2D

Con la documentación de Citro2D delante [2] vamos a explorar las posibilidades de esta librería para el uso de operaciones de dibujo 2D aprovechando la aceleración del *hardware* de la consola. Veamos cómo se puede utilizar en primer lugar para poner texto con muchas más opciones que las que nos da el uso de la pantalla en modo consola. Después pasaremos a ver la parte gráfica con primitivas de dibujo 2D.

3.1 Gestión avanzada del texto

En la práctica 2 "Acceso a información en el hardware de la videoconsola 3DS" hemos visto en un ejemplo de 3DS (*graphics/printing/hello-world/*) que la resolución del terminal en modo texto es de: 30filas x 50columnas para la pantalla superior y 30 x40 para la inferior.

Una cuestión muy importante es que se puede "escribir" texto utilizando los tipos de letra del sistema y con mayores posibilidades de control sobre el pintado del mismo. Para ello vamos a ver el ejemplo de 3DS (*graphics/printing/system-font/*). En él se hace uso de la librería Citro2D. El conjunto de funciones para manejo de texto en Citro2D permiten, como muestra la Figura 6, entre otras operaciones, dibujar texto con la función:

```
C2D_DrawText (const C2D_Text *text, u32 flags, float x, float y, float z, float scaleX, float scaleY,...)
```

Los cuatro sitios en que aparece esta función permiten dibujar el texto que aparece en la ejecución de este ejemplo, véase la Figura 6,

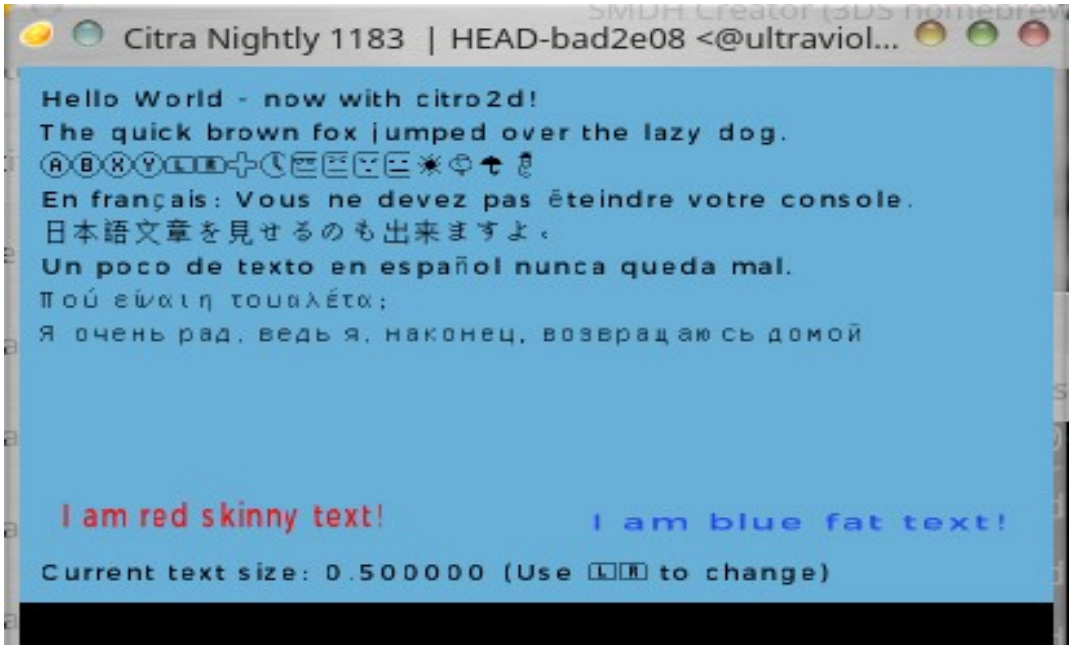


Figura 6: Salida de system-font.

Ejercicio 3. Sobre el ejemplo de 3DS *graphics/printing/system-font*, comprobará que si es posible

utilizar caracteres como la ‘ñ’ del español. Veamos si también es posible hacer uso de la ‘ç’ del valenciano, las tildes acentuadas o la diéresis de estos idiomas.

3.2 Primitivas de dibujo 2D

El conjunto de primitivas de dibujo contenidas en *Citro2D* permiten también el uso de primitivas gráficas 2D[1], como:

- bool C2D_DrawImage (C2D_Image img, const C2D_DrawParams *params,
  const C2D_ImageTint *tint, C2D_OPTIONAL(nullptr))
- bool C2D_DrawTriangle (float x0, float y0, u32 clr0, float x1, float y1, u32 clr1,
  float x2, float y2, u32 clr2, float depth)
- C2D_DrawEllipse (float x, float y, float z, float w, float h, u32 clr0, u32 clr1, u32 clr2, u32 clr3)
- C2D_DrawCircle (float x, float y, float z, float radius, u32 clr0, u32 clr1, u32 clr2, u32 clr3)

Veamos cómo se utilizan examinando el ejemplo de 3DS (*graphics/gpu/2d_shapes*) que constituye un ejemplo básico de las primitivas básicas sin recurrir a las texturas (las imágenes). El resultado de su ejecución se muestra en la Figura 7.

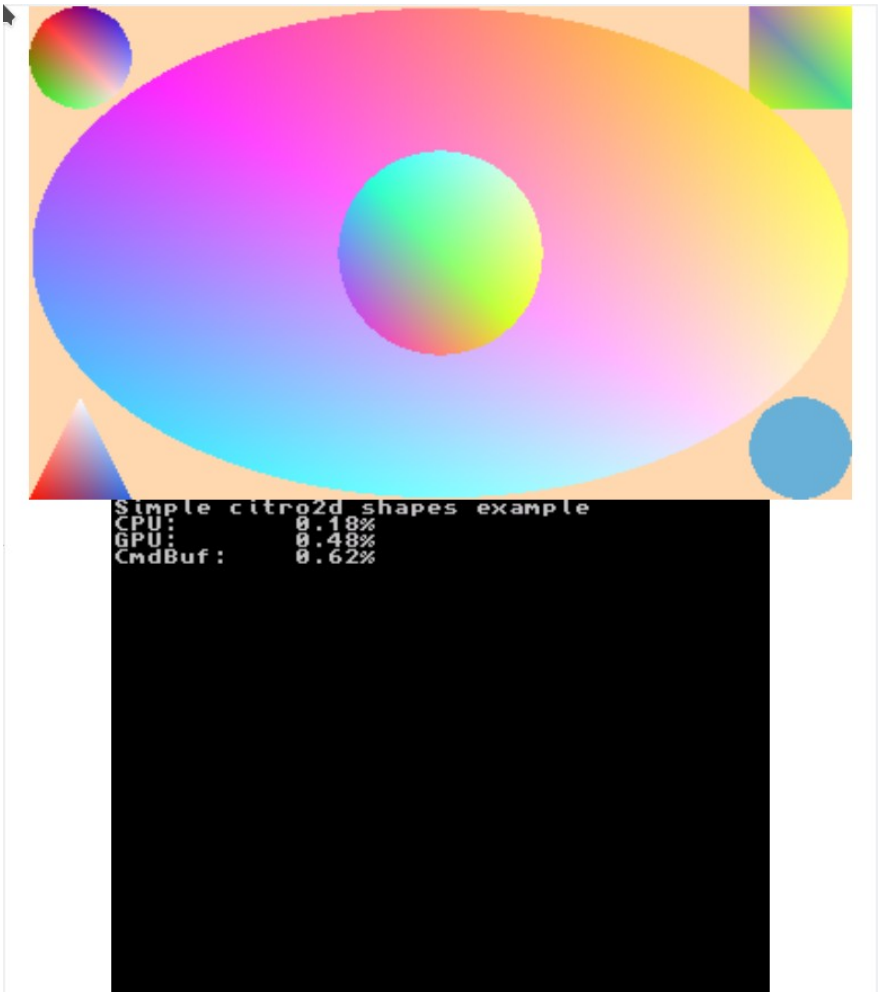


Figura 7: Resultado de la ejecución del ejemplo de 3DS graphics/gpu/2d_shapes.

1 “Drawing functions“ disponible en la URL <https://citro2d.devkitpro.org/group__Drawing.html#ga19535c770c9907bea5d9c4072d0b8186>.

Ejercicio 4. Sobre el ejemplo de 3DS *graphics/gpu/2d_shapes* anote cómo han dibujado los tres círculos para que cada uno haya salido con un color o degradado de color de relleno.

Un apunte al respecto del dibujo de primitvas, lo dice el código de este ejemplo y, también lo dice la documentación, es que el orden de dibujado de las primitivas gráficas puede ser importante.[1]

4 Sprites y animaciones con Citro2D

Este apartado revisa el ejemplo para 3DS graphics/gpu/gpusprites/ cuya salida se muestra en la Figura 8, observe que puede cambiar el número de *sprites* en pantalla con el uso de los botones de flecha arriba y abajo.

Este ejemplo 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 para ser utilizadas como *sprites* dentro de la aplicación. Para facilitar su gestión, dado que son cerca de sesenta imágenes, se utiliza un fichero de texto (T3S) a modo de catálogo.

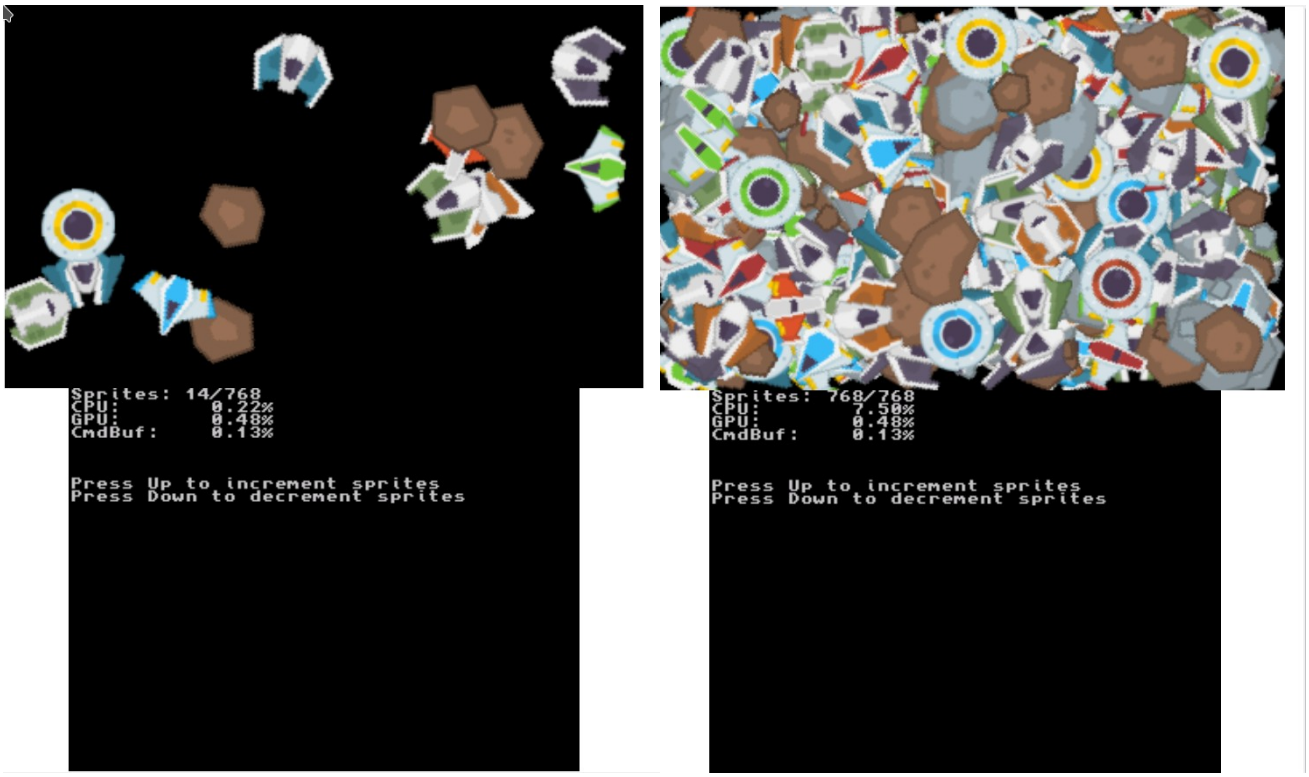


Figura 8: Resultado de la ejecución del ejemplo para 3DS gpusprites en Citra: dos instantes extremos de la ejecución.

Abra, con un editor de texto el fichero gpusprites/gfx/sprites.t3s y podrá comprobar que contiene la

1 Sacado de <https://citro2d.devkitpro.org/group__Drawing.html#gac8f73e2fabe4e60bbe78eb4264827549>: "Switching to and from "circle mode" internally requires an expensive state change. As such, the recommended usage of this feature is to draw all non-circular objects first, then draw all circular objects".

secuencia de líneas:

```
--atlas -f rgba8888 -z auto
enemyBlack1.png
enemyBlack2.png
…
ufoRed.png
ufoYellow.png
```

La primera de las líneas del fichero T3S es la que instruye a la utilidad *tex3ds* que se describe en el "Anexo I tex3ds - 3DS Texture Conversion" para que genere un *spritesheet* (atlas o diccionario de texturas)[1], utilizando 32-bits por pixel (RGBA de 8 bits por componente) y compresión. Estas imágenes están guardadas en formato PNG, con tamaños diferentes entre 16x15 y 50x38 píxeles.

En la documentación de *Citro2D*[2] se pueden consultar las funciones para la gestión de *spritesheets*. En nuestro caso se hará uso de

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

que buscará el fichero binario *sprites.t3x* en el sistema de archivos de solo lectura que puede ir incluido en el ejecutable y lo cargará en memoria. El programa lee en su inicio este mapa de *sprites*. La aplicación sabe indexarlos (identificarlos), véase el fichero graphics/gpu/gpusprites/build/sprites.h, porque *tex3ds* los ha declarado utilizando el nombre del fichero PNG original de cada uno y en el orden en que se han declarado en el archivo T3S como se ve en esta pequeña muestra de su contenido:

```
/* Generated by tex3ds */
#pragma once

#define sprites_enemyBlack1_idx 0
#define sprites_enemyBlack2_idx 1
#define sprites_enemyBlack3_idx 2
```

…

Ejercicio 5. 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 y haga una captura de pantalla.

5 Dibujo en 3D con Citro3D

Terminaremos esta práctica con una breve introducción al uso de las primitivas para dibujo en 3D aprovechando la aceleración que ofrece la GPU de la plataforma, a través de *Citro3D* [3]. Esta librería es la que ofrece el soporte a las operaciones de renderizado en tres dimensiones en cualquiera de las dos pantallas de la consola con la aceleración *hardware* que puede ofrecer la 3DS. El autor de esta librería es "fincs"(͡° ͜ʖ ͡°), el mismo autor que en el caso de *Citro2D*, de hecho *Citro2D* se ha desarrollado sobre la base de *Citro3D*.

De entre los ejemplos de uso de esta biblioteca de funciones para 3DS veremos aquí *texture_cube* y *cubemap* (véase la Figura 9), por su relación en cuanto al uso de las texturas sobre las caras de un cubo y porque son conceptos que se usan en otras plataformas y así, aquí veremos cómo se soportan con *tex3ds*.

---

1 Aunque se menciona otra herramienta de desarrollo que no es la que utilizamos en esta práctica, la descripción de un atlas o *sprite sheet* es interesante y breve.

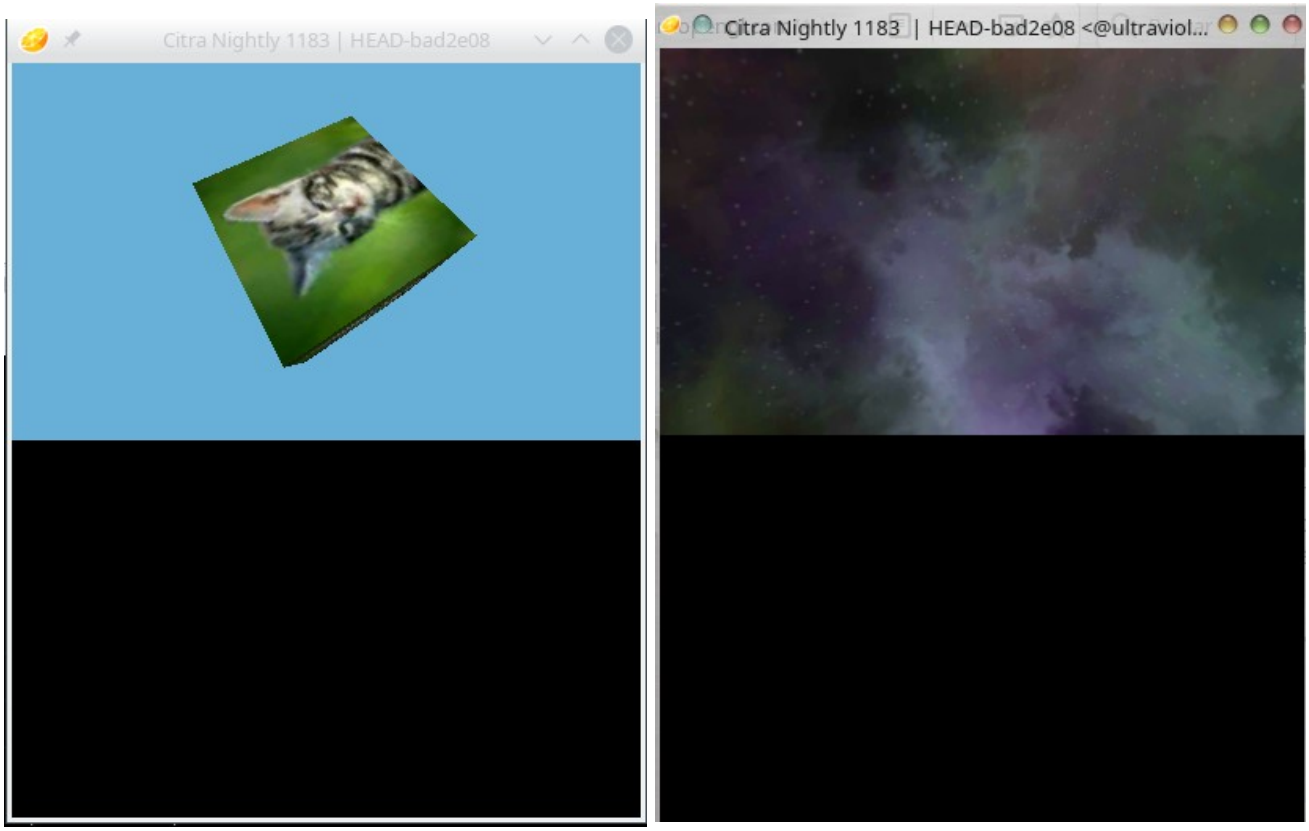
2 "Sprite sheet functions", en la URL <https://citro2d.devkitpro.org/group__SpriteSheet.html#gac7c6115169c5f0f4bb7a7d0beea4b75>

En el primer caso, *texture_cube*, veremos cómo se aplican las texturas a las caras de un objeto: en este caso, siempre es la misma textura (imagen) y sobre un objeto sencillo, un cubo. En el segundo ejemplo, *cubemap*, veremos cómo se puede "empapelar" con una textura las caras interiores de un cubo, consiguiendo así que el usuario se vea envuelto por esta imagen de fondo

Ambos ejemplos tienen una estructura básica común: el código de la función *main* que se muestra en la Figura 10 y que pertenece, en concreto, al ejemplo *texture_cube*. Es el encargado de inicializar el modo gráfico. En ambos casos se escoge la pantalla superior. Tras ello se llama a la función *sceneInit*, donde se cargan las texturas. El bucle del programa principal es el encargado, como es habitual, de recoger las entradas del usuario para decidir la acción a tomar. El caso de *texture_cube* es muy simple puesto que no se puede hacer nada más que terminar si se pulsa la tecla *Start*; pero en el caso de *cubemap* es posible mover la cámara con los botones de las flechas.

De nuevo, el código de los dos ejemplos se vuelve a hacer semejante por cuanto que decidida la acción del usuario, toca aplicarla y actualizar el renderizadeo de la escena. Especialmente significativo es el contenido de la función *sceneRender*, que si mostrará diferencias entre ambos casos analizados.

Cuando termina el bucle principal, ambos ejemplos utilizan la misma secuencia de liberación de recursos: con las funciones *sceneExit* , *C3D_Fini* y *gfxExit.*



a) b)

Figura 9: Ejemplos de uso de Citro3D: (a) texture_cube y (b) cubemap.

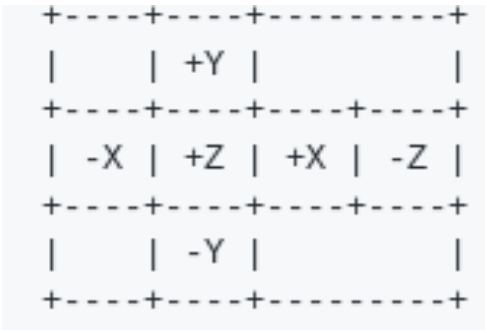
El ejemplo de 3DS *graphics/gpu/texture_cube*, muestra cómo poner una textura, siempre la misma, a las caras de un cubo. Para ello se le indica a la utilidad *tex3ds* que ha de asignar (véase Figura 11 y el Anexo I), la única imagen que hay, a cada cara del cubo o una parte de una imagen a cada cara. El contenido de textured_cube/gfx/kitten.t3s es:

```
-f auto-etc1 -z auto
kitten.png
```

```c
int main()
{
    // Initialize graphics
    gfxInitDefault();
    C3D_Init(C3D_DEFAULT_CMDBUF_SIZE);

    // Initialize the render target
    C3D_RenderTarget* target = C3D_RenderTargetCreate(240, 400, GPU_RB_RGBA8, GPU_RB_DEPTH24_STENCIL8);
    C3D_RenderTargetSetOutput(target, GFX_TOP, GFX_LEFT, DISPLAY_TRANSFER_FLAGS);

    // Initialize the scene
    sceneInit();

    // Main loop
    while (aptMainLoop())
    {
        hidScanInput();

        // Respond to user input
        u32 kDown = hidKeysDown();
        if (kDown & KEY_START)
            break; // break in order to return to hbmenu

        // Render the scene
        C3D_FrameBegin(C3D_FRAME_SYNCDRAW);
            C3D_RenderTargetClear(target, C3D_CLEAR_ALL, CLEAR_COLOR, 0);
            C3D_FrameDrawOn(target);
            sceneRender();
        C3D_FrameEnd(0);
    }

    // Deinitialize the scene
    sceneExit();

    // Deinitialize graphics
    C3D_Fini();
    gfxExit();
    return 0;
}
```

Figura 10: Estructura básica del programa principal de los ejemplos de renderizado en 3D con DevkitPro y Citro3D para la plataforma 3DS.

a)

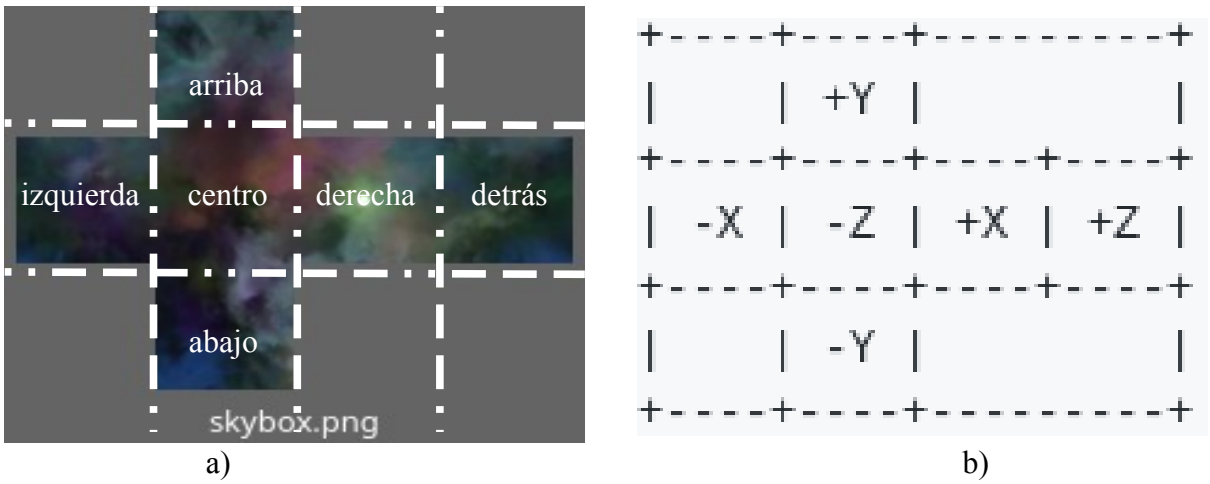


b)

Figura 11: Aplicando texturas a la caras del cubo con tex3ds: (a) todas iguales en el caso de texture_cube y (b) diseño general de una imagen con texturas para cada cara.

Ejercicio 6. Describa el contenido del proyecto del ejemplo para 3DS *graphics/gpu/texture_cube*, indicando qué archivos existen de partida y cuáles se generan con la construcción del ejecutable.

El ejemplo de 3DS *graphics/gpu/cubemap*, muestra cómo utilizar la idea de *Skybox* para ilustrar un paisaje de fondo dentro del cual se puede mover la cámara. Son las texturas interiores del cubo en el que se encuentra el usuario y dan la sensación de imagen de fondo. Conforme se cambia el punto de vista de la cámara se actualiza el contenido de la misma con el correspondiente "trocito" del "fondo". Para ello se define un patrón o estructura que debe tener la imagen que se utilizará como Skybox y que se muestra en la Figura 12.



a) b)

Figura 12: Ejemplos de uso del concepto de Skybox en DevkitPro para 3DS: (a) disposición de las texturas que se asignarán a las caras del cubo y (b) relación con las coordenadas de posición de la cámara para tex3ds.

Para obtener en el ejemplo esta funcionalidad, la utilidad *tex3ds* interpretará la imagen del directorio *gfx* de acuerdo con con las instrucciones del fichero T3S *graphics/gpu/cubemap/gfx/skybox.t3s:*:

```
--skybox -f etc1 -z auto
skybox.png
```

Ejercicio 7. Describa el contenido del proyecto del ejemplo para 3DS *graphics/gpu/cubemap*, indicando qué archivos existen de partida y cuáles se generan con la construcción del ejecutable.

6 Trabajo autónomo

Ahora que hemos visto unos cuantos ejemplos, es el momento de hacer uno. Se propone hacer una aplicación para 3DS que muestre:

- En la pantalla inferior, la foto del alumno (o alumnos que trabajan juntos en la práctica) como *sprite* cuyo movimiento ha de ser controlado por el usuario (utilizando los botones del *D-Pad* o la pantalla táctil), junto a otros dos *sprites* que se mueven aleatoriamente rebotando en la pantalla.
- En la parte superior de la consola, los mensajes de la aplicación (uso de recursos y cómo puede interactuar el usuario se mostrarán con texto utilizando diferentes colores de letra, posiciones y tamaños.

7 Bibliografía

[1] “Smealum”. “libctru - CTR User Library”. Disponible en la URL <http://smealum.github.io/ctrulib/>.

[2] “Fincs”. Documentación *Citro2D*. Disponible en la URL <https://citro2d.devkitpro.org/>.

[3] “Fincs”. “*Citro3D - Homebrew PICA200 GPU wrapper library for Nintendo 3DS*“. Disponible en la URL <https://github.com/fincs/citro3d >.

Anexo I. tex3ds - 3DS Texture Conversion

La utilidad tex3ds[1] la veremos utilizada para convertir las texturas en estructuras de datos que pueden ser manipuladas desde el SDK. La ayuda de esta aplicación se ha copiado aquí para tenerla a mano y poder entender algunos de sus usos.

```
Usage: ./tex3ds [OPTIONS...] <input>
  Options:
    -f, --format <format>        See "Format Options"
    -H, --header <file>          Output C header to file
    -h, --help                   Show this help message
    -i, --include <file>         Include options from file
    -m, --mipmap <filter>        Generate mipmaps. See "Mipmap Filter Options"
    -o, --output <output>        Output file
    -p, --preview <preview>      Output preview file
    -q, --quality <etc1-quality> ETC1 quality. Valid options: low, medium
(default), high
    -r, --raw                    Output image data only
    -t, --trim                   Trim input image(s)
    -v, --version                Show version and copyright information
    -z, --compress <compression> Compress output. See "Compression Options"
    --atlas                      Generate texture atlas
    --cubemap                    Generate a cubemap. See "Cubemap"
    --skybox                     Generate a skybox. See "Skybox"
    <input>                      Input file
```

Format Options

```
    -f rgba, -f rgba8, -f rgba8888
      32-bit RGBA (8-bit components) (default)

    -f rgb, -f rgb8, -f rgb888
      24-bit RGB (8-bit components)

    -f rgba5551
      16-bit RGBA (5-bit RGB, 1-bit Alpha)

    -f rgb565
      16-bit RGB (5-bit Red/Blue, 6-bit Green)

    -f rgba4, -f rgba444
      16-bit RGBA (4-bit components)

    -f la, -f la8, -f la88
      16-bit Luminance/Alpha (8-bit components)

    -f hilo, -f hilo8, -f hilo88
      16-bit HILO (8-bit components)
      Note: HI comes from Red channel, LO comes from Green channel

    -f l, -f l8
```

1 Está instalada ya con el SDK de DevkitPro y se puede obtener desde <https://github.com/devkitPro/tex3ds>.

```
  8-bit Luminance

-f a, -f a8
  8-bit Alpha

-f la4, -f la44
  8-bit Luminance/Alpha (4-bit components)

-f l4
  4-bit Luminance

-f a4
  4-bit Alpha

-f etc1
  ETC1

-f etc1a4
  ETC1 with 4-bit Alpha

-f auto-l8
  L8 when input has no alpha, otherwise LA8

-f auto-l4
  L4 when input has no alpha, otherwise LA4

-f auto-etc1
  ETC1 when input has no alpha, otherwise ETC1A4
```

Mipmap Filter Options

```
-m bartlett
-m bessel
-m blackman
-m bohman
-m box
-m catrom
-m cosine
-m cubic
-m gaussian
-m hamming
-m hanning
-m hermite
-m jinc
-m kaiser
-m lagrange
-m lanczos
-m lanczos-radius
-m lanczos-sharp
-m lanczos2
-m lanczos2-sharp
-m mitchell
-m parzen
-m point
-m quadratic
-m robidoux
-m robidoux-sharp
-m sinc
-m spline
-m triangle
```

```
-m welsh
```

Compression Options

```
-z auto              Automatically select best compression (default)
-z none              No compression
-z huff, -z huffman  Huffman encoding (possible to produce garbage)
-z lzss, -z lz10     LZSS compression
-z lz11              LZ11 compression
-z rle               Run-length encoding

NOTE: All compression types use a compression header: a single byte which
      denotes the compression type, followed by three bytes (little-endian)
      which specify the size of the uncompressed data. If the compression
      type byte has the MSB (0x80) set, the size is specified by four bytes
      (little-endian) plus three bytes of reserved (zero) padding.

Types:
  0x00: Fake (uncompressed)
  0x10: LZSS
  0x11: LZ11
  0x28: Huffman encoding
  0x30: Run-length encoding
```

Cubemap

```
A cubemap is generated from the input image in the following convention:
+----+----+---------+
|    | +Y |         |
+----+----+----+----+
| -X | +Z | +X | -Z |
+----+----+----+----+
|    | -Y |         |
+----+----+---------+
```

Skybox

```
A skybox is generated from the input image in the following convention:
+----+----+---------+
|    | +Y |         |
+----+----+----+----+
| -X | -Z | +X | +Z |
+----+----+----+----+
|    | -Y |         |
+----+----+---------+
```

Anexo II. Problemas de compilación con el ejemplo de 24bit-color en Windows

Si estás compilando el ejemplo de 3DS dentro de *graphics/bitmap/24bit-color*, posiblemente, te enconcontrarás con un error: el error es debido a que intenta usar la aplicación del sistema `convert` (`/c/WINDOWS/system32/convert.exe`) y no el conversor de formatos de imágenes `convert` del paquete *ImageMagick*. Así que se obtiene un error porque la aplicación llamada no entiende los parámetros que se le pasan.

Con los ejemplos de 3DS se está cambiado de GRIT a `convert` (*ImageMagick*) para resolver las conversiones de ficheros de formato gráfico.

La posible solución se basa en:

1. Instalar ImageMagick, desde la entrada "Downloads" del sitio web " ImageMagick " <https://www.imagemagick.org/script/index.php> y comprobar la ruta desde el terminal (`Start -> devkitPro -> MSYS`). En mi caso es `/c/Archivos\ de\ programa/ImageMagick-6.9.9-Q16/convert.exe`
2. Editar el fichero *Makefile* del mencionado ejemplo. Para ello
    1. Añadir la línea
       `CONVERT = /c/Archivos\ de\ programa/ImageMagick-6.9.9-Q16/convert.exe`
       al principio del *Makefile*. Y sustituir el bloque

```
--------------------
%.bgr: %.png
#---------------------------------------------------------------
	@convert $< -rotate 90
```

por

```
 --------------------
%.bgr: %.png
 #---------------------------------------------------------------
	$< -rotate 90
```

Y ya será posible recompilar desde el terminal: `Start -> devkitPro -> MSYS`

con la orden

```
$ make clean && make
```

PD. El fichero README.md que acompaña al ejemplo dice algo parecido:

24bit Bitmap Example

This example shows on bottom screen an upscaled version of the nds examples Drunken Coders

logo that can be found in devkitPro.

If you want to try with your own image follow these steps:

1. *Download & install: http://www.imagemagick.org/ (If you get an option to add the application to the path make sure to check it!).*
2. *convert fileIn.png -channel B -separate fileIn.png -channel G -separate fileIn.png -channel R -separate -channel RGB -combine -rotate 90 fileOut.rgb*
3. *Rename fileOut.rgb in fileOut.bin*
4. *Copy fileOut.bin in the data folder of your project*
5. *Replace any reference of drunkenlogo_bin in main.cpp with fileOut_bin (or however you named it)*
6. *Re-Build the project*

As you can see from the previos steps the image is clockwise rotated by 90 degrees and its B and R channels are swapped. The first operation is done because the 3DS screens are actually portrait screens rotated by 90 degrees (in a counter-clockwise direction), while the second one is done because the 3DS screens' framebuffers have a BGR888 color format, by default.