Programación Gráfica 2D (III)
Bliteando más y mejor.

Autor: Sergio Hidalgo
serhid@wired-weasel.com

En el primer tutorial hice una introducción a la librería, y comenté que se basa en dos pilares fundamentales: las superficies y el blit. El segundo tutorial estaba dedicado por entero a las superficies, y por supuesto, éste tratará sobre el blit.

Este es el último tutorial sobre las operaciones básicas de SDL. Por supuesto la librería tiene muchas más cosas que no he tocado (Overlays, paletas, gamma, etc...). Pero para lo que vamos a hacer, en general con lo de estos tres tutoriales es suficiente.

Así que el próximo tutorial tratará ya sobre técnicas 2D propiamente dichas, y conceptos como tile-mapping, optimizaciones, etc... mas que sobre la librería. De todos modos, recomiendo que probéis las SDL por vuestra cuenta, escribais un par de programas cargando superficies y bliteando, etc...

Introducción

En el primer tutorial dije que el blit es el equivalente a hacer un “copy-paste” de una superficie a otra, y que básicamente se trata de una copia de datos de una zona de memoria a otra. Bueno, eso no es completamente cierto. Un blitter (unidad que realiza los blits) hace más cosas que simplemente mover datos.

Viendo el último tutorial, por ejemplo, podemos hacernos una idea. ¿Qué ocurre si las dos superficies involucradas tienen distintos formatos de pixel, por ejemplo? El blitter se encargaría de “traducir” de un formato a otro.

En general, todo eso es transparente. Lo único que hacemos como programadores es establecer algún parámetro en algunos casos para indicarle al blitter qué resultado queremos. No nos importa como esté implementado el blitter. Pero si que hay que tener en cuenta que ciertas operaciones pueden repercutir en el rendimiento, y evitarlas en esos casos.

La regla simple es que a más complejidad, más lentitud. Cuantas mas cosas le obliguemos al blitter a hacer, más va a tardar en hacerlas. Por eso mismo comenté en el otro tutorial que es una buena idea que todas nuestras superficies compartan el mismo formato de pixel. Esto ahorra al blitter hacer traducciones.

También hay que tener en cuenta que en realidad existen 2 blitters. Uno en la tarjeta gráfica, implementado mediante hardware, y otro por software. El blitter HW es mucho más rápido, pero también mucho más limitado. En el caso de que el blitter HW no pueda hacer una operación (porque no la soporte, o porque las superficies no estén en memoria de video), el blitter SW se encarga de ella.

Así que conviene conocer qué operaciones son las que los blitters HW suelen no tener implementadas, pera tratar de evitarlas. Recuerdo antes de continuar cómo era la instrucción Blit:

```
int SDL_BlitSurface(SDL_Surface *src, SDL_Rect *srcrect, SDL_Surface *dst,
```

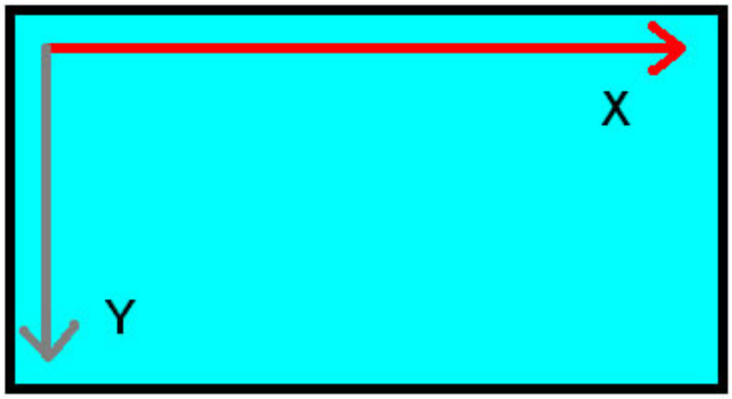
```
                    SDL_Rect *dstrect);
```

src = Superficie Origen
dst = Superficie Destino
srcrect = Rectángulo de la superficie origen que copiaremos
dstrect = Rectángulo de la superficie destino sobre el que copiaremos

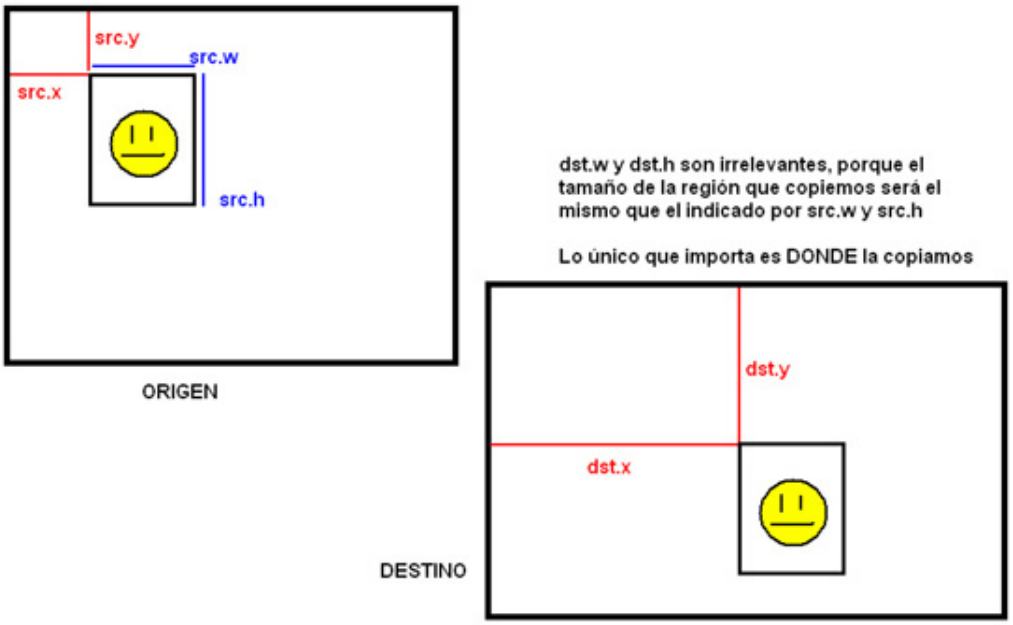
Hay un par de cosas que creo que no coment é bien en la introducción, asi que aprovecho aquí:

- Los rectángulo SDL tienen las coordenadas (X, Y) de la esquina superior izquierda, y luego el ancho y el alto del rectángulo. En conjunto: {x, y, w, h}

- Los ejes de coordenadas son tal que así:



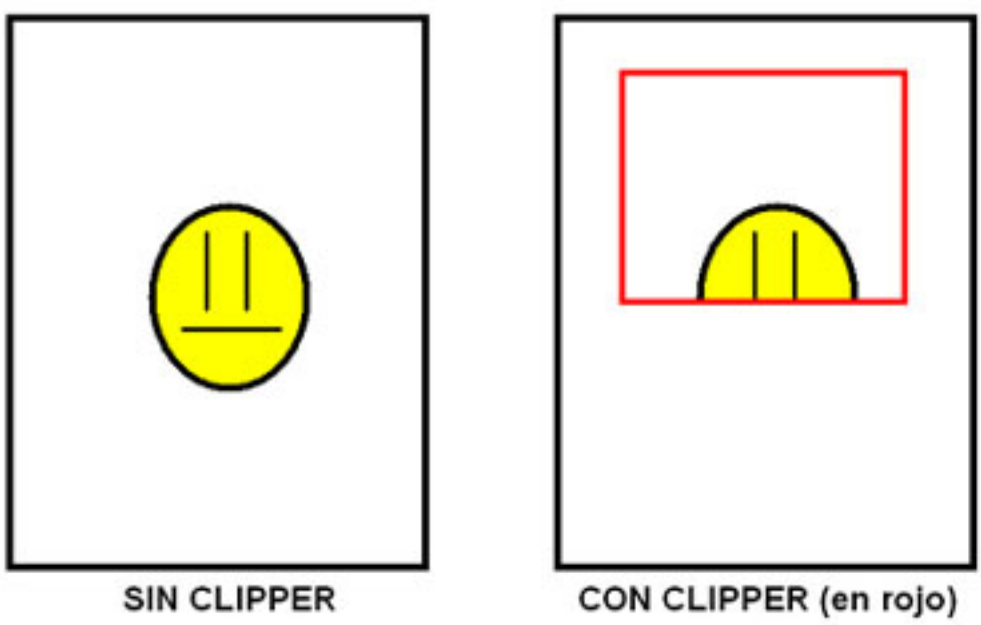
- Cuando hacemos un blit, el tamaño de la región que copiaremos lo indica el rectángulo de origen. El tamaño del rectángulo destino no influye en absoluto. Sólo importa las coordenadas del lugar en donde copiaremos la región. No es posible redimensionar una región al copiarla. O al menos, no con la librería básica.



Si veis el primer tutorial, comprobareis que yo rellenaba el ancho y alto del rectángulo destino. Eso no es necesario, y lo hago por costumbre con de usar otra librería distinta. En general, no está de más hacerlo por si más adelante decidimos cambiar (que sería muy raro), pero vamos, que da igual xD

Clippers

Empezamos por fin con los añadidos al blitter. Un clipper es un rectángulo que tiene cada superficie. Es una de las propiedades que mencioné en el otro tutorial, pero que no detallé. Lo que hace el clipper es definir el área de la superficie destino sobre el que podemos blittear. Todo lo que caiga fuera de ese área, no se dibuja.



El clipper de una superficie solo afecta a los blits que tengan como DESTINO a esa superficie. No afecta para nada al origen.

En la imagen de la izquierda dice “sin clipper”, pero eso no es completamente correcto. En realidad, siempre hay un clipper, sólo que por defecto su área abarca toda la superficie, dejando que bliteemos en todas partes.

La razón de que existan los clippers es sencillamente el evitar que al hacer un blit podamos escribir en zonas de memoria que no corresponden a la superficie. El rectángulo nunca puede extenderse más allá del limite de la superficie, así que ese problema desaparece automáticamente.

Pero además de eso, podemos darle más utilidades para optimizar el dibujado. Si sólo necesitamos actualizar una zona de la superficie destino, pero la región que estamos blitteando es enorme, usando un clipper nos evitamos escribir pixels que no necesitamos (o que no debemos, porque no queremos machacarlos).

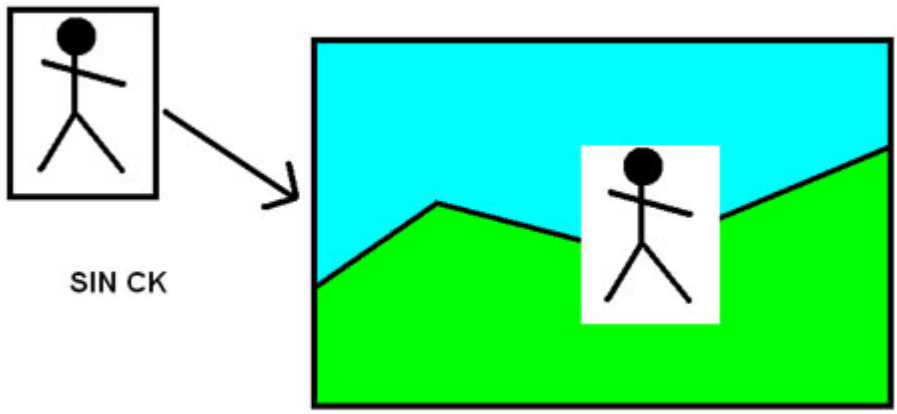
Como he comentado, cada superficie tiene su propio rectángulo de clip. Es el “clip_rect” que podiamos ver en la estructura SDL_Surface. Para fijar un clipper:

SDL_SetClipRect(SDL_Surface *surface, SDL_Rect *rect);

Como suponeis, “surface” es la superficie que queremos modificar, y “rect” el nuevo rectángulo de clip.

Color Keys

Imaginad que tenemos una superficie con un personaje que queremos blittear en otra superficie con un escenario. Haciendo lo que hemos estado haciendo hasta ahora, esto es lo que ocurre:



¿Veis el problema? Al blitear, bliteamos TODOS los pixeles que hay en la región. En realidad, no queremos que los pixeles de color blanco se bliteen en la superficie destino. Queremos que esos pixeles sean “transparentes”. ¿Cómo se lo decimos al blitter?

Lo hacemos definiendo ese color como “Color Key”. Un CK es precisamente eso, un color que el blitter considera como transparente al hacer la operación.

De nuevo, el CK es una propiedad de cada superficie, asi que podemos hacer que cada una tenga un color transparente distinto, que sólo se toma en cuenta cuando esa superficie es el ORIGEN de un blit (flag SDL_SRCCOLORKEY).

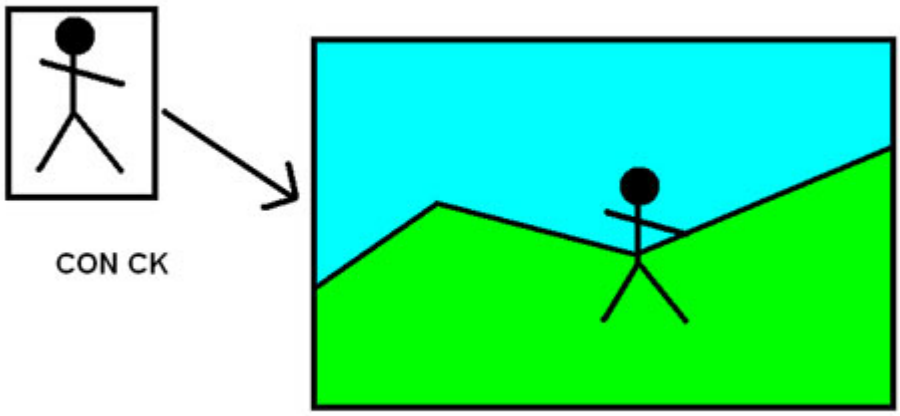
NOTA: Existe también un CK de destino, pero es poco común y no todos los blitters por HW lo implementan, asi que no lo voy a comentar.

Para fijar un CK para una superficie, usamos esta instrucción:

SDL_SetColorKey(SDL_Surface *surface, Uint32 flag, Uint32 key);

“surface” es la superficie, “flag” será en nuestro caso SDL_SRCCOLORKEY, y “key” será el color propiamente dicho. Como siempre, el color irá en el formato de pixel de la superficie, así que tendremos que usar la instrucción MapRGB.

Y el resultado si hacemos eso con el blanco sería:

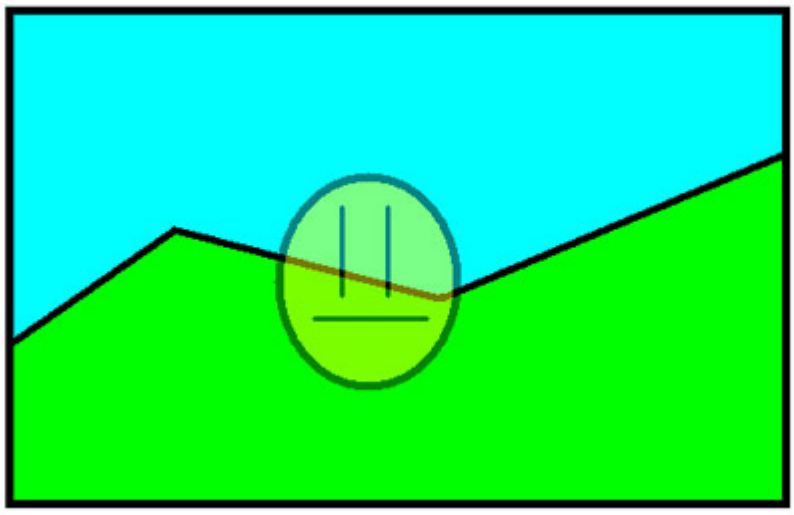


Por lo general, se suele usar un color rosa, el RGB = (255, 0, 255) como color transparente. La razón es que ese color en concreto es bastante feo, y es difícil encontrarlo como parte de la imagen.

Canal Alpha

¿Recordáis cuando hablaba de los formatos de pixeles que comenté algo sobre que el formato RGBA tenía un “canal alpha”, que iba junto a los demás canales del color (RGB)?

El canal alpha define el grado de opacidad de un pixel. A menor valor, más transparente es. La idea es que al blittear una superficie usando un canal alpha, el resultado será que la imagen bliteada quedará semi-transparente, y podemos ver lo que había detrás.



Entonces, si podemos usar el alpha para definir zonas transparentes con mucha más libertad, ¿por qué nos limitamos a usar los ColorKeys?

La razón es que, por regla general, los blitters HW no implementan blits con alpha. Esto es porque la ecuación para mezclar los colores requiere leer también desde la superficie destino, lo que es mas lento. Así que si usamos el canal alpha, nos forzamos a depender de los blits por SW. Y claro, es bastante más lento.

Por lo general, si REALMENTE necesitas usar el canal alpha para ciertos efectos, y necesitas que vaya considerablemente rápido (no lo usas para uno o dos blits, sino para un montón), es preferible pasar a utilizar una librería 3D (OpenGL o DirectX) que tienen un tratamiento mucho mejor de este tema.

En caso de usar SDL, lo mejor es que todas las superficies involucradas (tanto la origen como la de destino) se encuentren en SW, para agilizar los trámites y no tener que enviar datos a través del bus.

De todos modos, si os interesa, mirad en la documentación la función SDL_SetAlpha (...) para ver como funciona exactamente.

Y bueno, después de escabullirme de este modo tan ruin de explicar en profundidad el canal alpha, doy por terminado el tutorial. El siguiente ya tratará sobre técnicas gráficas de verdad.

Un saludo.