Programación Gráfica 2D (I)
Introducción

Autor: Sergio Hidalgo
serhid@wired-weasel.com

Bueno, aquí está el primero de una serie de “tutoriales” para explicar cómo funciona la programación gráfica en 2D, especialmente con SDL.

Antes de nada, pido disculpas por la poca calidad de las imágenes, al mas puro estilo "programmer art", pero creo que se entienden bien.

Estos no van a ser unos tutoriales completos. Prácticamente no voy a meter nada de código. La idea de esto es contar un poco los principios teóricos, y que luego podáis verlo en acción en cualquiera de los tutoriales de SDL que hay por Internet. Es más un complemento, así que espero que os busquéis un poco la vida por vuestra cuenta xD

Esto va dirigido a los que vayan a hacer la parte gráfica (no voy a tocar nada de SDL en cuanto a sonidos, eventos, y demás), pero si le interesa a más gente, pues vale, hay gustos para todo, no os voy a echar ni nada.

Diferencias entre 3D y 2D

Como he visto que algunos teneis experiencia en librerías 3D (ej: OpenGL), creo que lo mejor será empezar diciendo cuales son las diferencias fundamentales entre las dos filosofías (3D vs 2D).

En primer lugar, una API 3D se basa en polígonos. Estos polígonos están formados por vértices con coordenadas en 3D. Lo que hacemos (o lo que hace la API), es “proyectar” estos polígonos sobre un plano (la pantalla, o más concretamente, un trozo de la pantalla al que se llama “viewport”)

Cuando esos polígonos los dibujamos en pantalla, además tenemos la opción de rellenarlos con una imagen, a la que llamamos “textura”. Este proceso se llama “rasterización”.

Entonces, para hacer una aplicación 2D en una librería 3D, bastaría con usar una proyección ortogonal.

Eso es completamente correcto para las librerías 3D, que se basan en el proceso de “proyección + rasterización”. Sin embargo, al usar una librería 2D (SDL), esos conceptos sencillamente no existen.

Para entender cómo es la filosofía de la programación clásica 2D, hay que pensar cómo eran los ordenadores de hace 15 años. En esa época las tarjetas aceleradoras no existían (o costaban una pasta), y todo este proceso de “proyección + rasterización” tenía que hacerse mediante software. Y en aquella época, los procesadores no eran especialmente rápidos, especialmente en operaciones en punto flotante.

Así que es lógico que al programar una aplicación en 2D, se hiciese mediante un sistema en el que las proyecciones y rasterizaciones no existan, y todos los cálculos sean únicamente operaciones enteras, que los procesadores pueden realizar mucho más rápido.

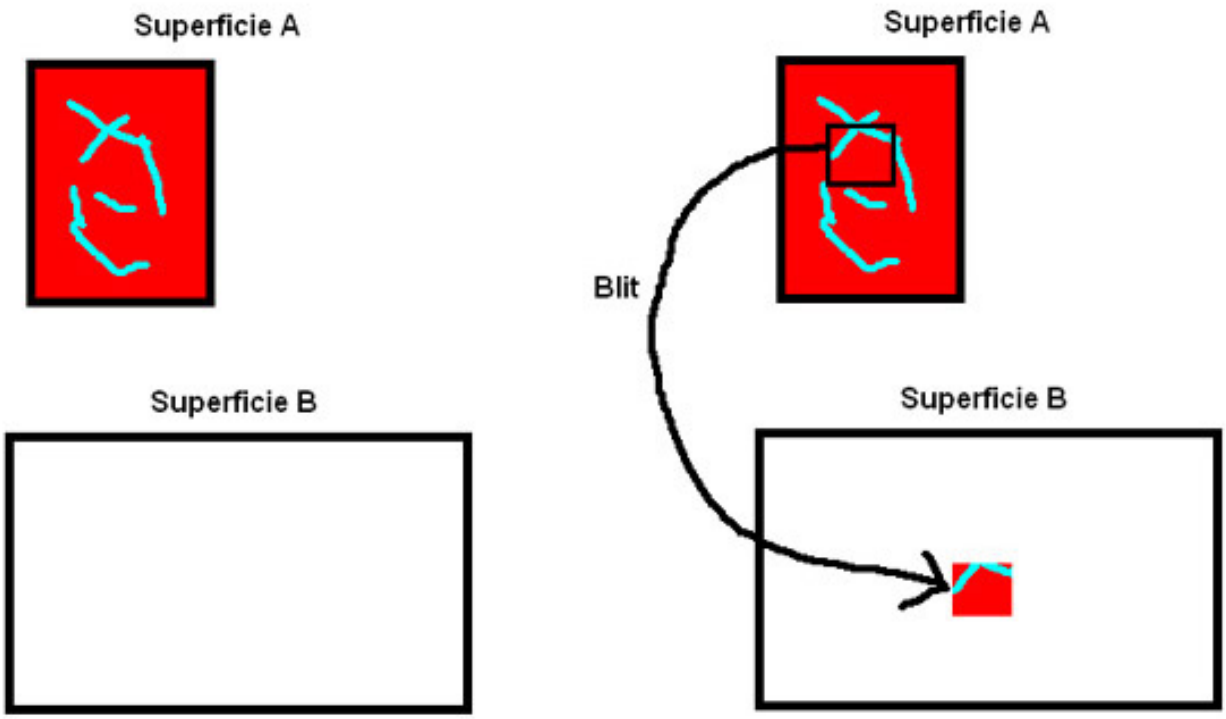
Conceptos básicos de programación 2D

Una librería gráfica 2D se basa en el concepto de **SUPERFICIES**, y de operaciones entre superficies. Una superficie no es más que un espacio en memoria donde guardar una imagen, y las operaciones que se realizan entre superficies, son copias de trozos de una superficie origen sobre una superficie destino.

Un ejemplo: el Paint.
Al cargar el paint, el lienzo representaría una superficie. Puede tener una imagen cargada desde un archivo, puede estar en blanco, etc...

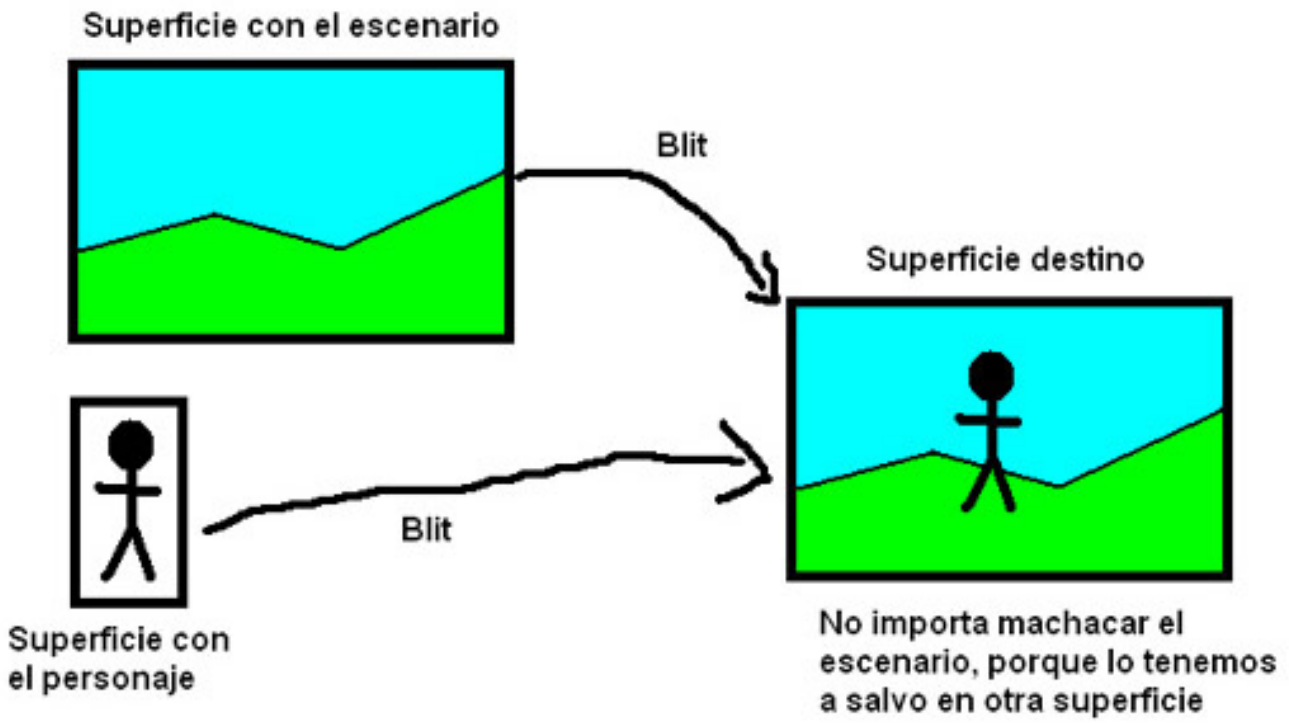
Ahora imaginad que ejecutamos 2 paints. Cada uno con su lienzo. Eso representa 2 superficies en memoria. Podemos tener una superficie vacía (blanca), y otra con una imagen cargada del archivo.

Y la operación básica entre superficies, sería hacer un "Copy-Paste". Seleccionar una región en la superficie A, y copiarla sobre algún lugar en la superficie B. Esta operación es a la que se llama "Blit", y es la más importante en cualquier librería 2D.



Al igual que en Paint, al hacer un Blit, lo que había en la superficie destino en la región que sobreescribimos se pierde para siempre. Al copiar sobre esa posición machacamos los datos anteriores. Esto es importante si tenemos, por ejemplo, un personaje moviéndose sobre un escenario. Al dibujar el personaje destruimos esa región del escenario, así que si el personaje se mueve, tendremos que volver a dibujar de nuevo el escenario por cada fotograma.

También de esto sacamos que el orden en que hagamos los blits importa, y mucho. Si algo tiene que quedar por encima, tiene que ser lo último que bliteemos.



Nota al margen: El verbo "blitear" (yo bliteo, tu bliteas, nosotros bliteamos...) no aparece en el diccionario, como muchos "palabros" que iré soltando más adelante (tilear, plottear, etc...). Pero vamos, creo que se entiende, ¿no?. Si algo queda raro y no se entiende, decidmelo.

Todo esto está muy bien para copiar imágenes entre superficies. ¿Pero cómo se muestra todo eso en pantalla?. Fácil, tendremos una superficie especial, que represente a la pantalla propiamente dicha. Todo lo que pongamos en esa superficie será lo que aparecerá por pantalla.

Esa superficie se crea al inicializar SDL, y se le suele llamar "Buffer Primario", o simplemente "Pantalla" ("Screen" en inglés). Y obviamente, el tamaño de esta superficie será exactamente idéntido a la resolución que estemos usando.

Flipping

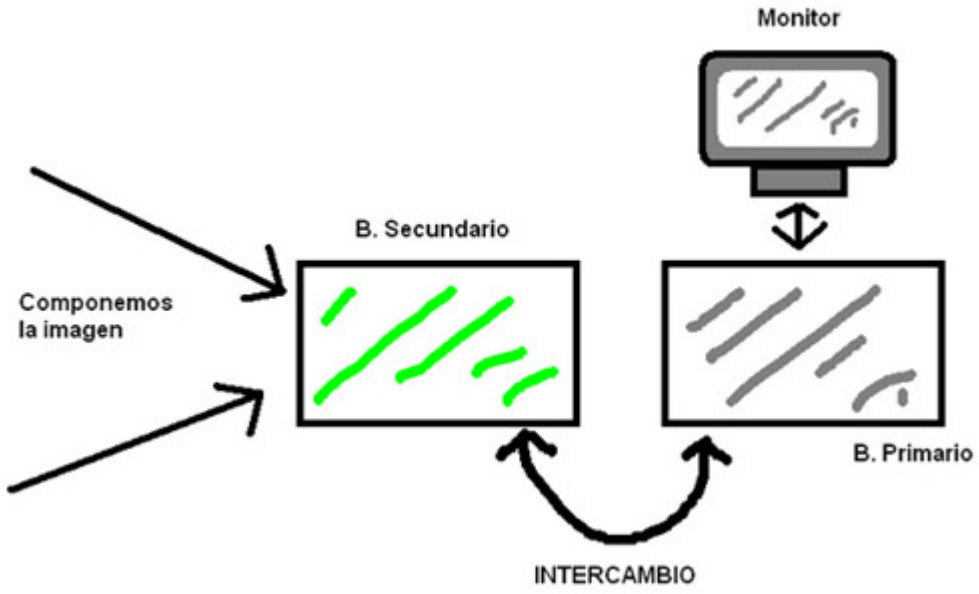
Vale, ahora tenemos una imagen que hemos compuesto sobre el Buffer Primario. Ya tenemos todo listo para que se muestre en el monitor. ¿Como hacemos para "actualizar" el monitor?

Depende, hay dos maneras de hacerlo, la fácil, y la difícil.

La fácil es llamar a **SDL_UpdateRects(...)**. Eso hace que la tarjeta gráfica envíe lo que haya en ese instante en el buffer primario al monitor. El problema de esto es que el monitor tiene un refresco de actualización, y si llamamos a esta función en el momento en que el monitor está dibujando algo, lo que veremos será un parpadeo bastante molesto.

Lo que queremos es que esa actualización se produzca justo en el momento en que el monitor ha terminado de dibujar un fotograma, y esta preparándose para el siguiente. A esto se le llama "sincronizarlo con el refresco vertical", o en inglés, "VSYNC".

Para hacer eso, lo que se usa es una técnica llamada “Double Buffer”. Consiste en que tendremos un buffer Secundario (o “BackBuffer”), además del primario. Nuestra imagen la compondremos en el secundario, y llegado el momento, intercambiaremos ambos buffers para que el secundario pase a ser el primario y viceversa, actualizando la imagen en el monitor. Este intercambio estará sincronizado con el refresco del monitor.



Por suerte, SDL se encarga de todo eso por nuestra cuenta, pero nunca está de más saber qué es lo que realmente ocurre.

A nivel práctico, seguiremos usando nuestra superficie “pantalla” como antes, sólo que a la hora de inicializar la librería, habrá que indicarle que use Double Buffer. Y a la hora de actualizar la pantalla después de componer nuestra imagen, llamaremos a **SDL_Flip (...)** en lugar de a SDL_UpdateRects(...)

Resumiendo:

Programación 3D != Programación 2D

Programación 3D == Punto Flotante, matrices, proyecciones + rasterizaciones
Programación 2D == Operaciones enteras, ints, copias de trozos de memoria de un lado a otro

Superficie: Espacio en memoria donde tenemos una imagen.
Blit: Copy-paste de una región de una superficie sobre otra.
Buffer Primario: Superficie que contiene lo que se dibuja en pantalla.
Double Buffer: Algo raro que evita los parpadeos
Flip: Orden a SDL para que actualice lo que se muestra en el monitor

Y ahora, el código:

No voy a escribir un programa completo desde el principio hasta el final, para eso ya hay montones de tutoriales en internet, lo único que voy a decir es cómo se traduce lo que he comentado antes a código en SDL:

NOTA: Tampoco incluyo nada de comprobación de errores, por ahorrarme teclas, nada más. Pero no seais cafres y ponedlo vosotros xD

Inicializar SDL:

```
SDL_Init (SDL_INIT_VIDEO);
```

Configurar el modo gráfico y "crear" el buffer primario:

```
SDL_Surface *pantalla = NULL;

pantalla = SDL_SetVideoMode (800, 600, 16, SDL_DOUBLEBUF | SDL_FULLSCREEN);
```

Aquí, los dos primeros parámetros son la resolución (800x600 pixels), el siguiente es los bits por pixel (16 bpp), y detrás vienen los flags. En este caso inicializamos el modo gráfico para que use Double Buffer, y que ocupe toda la pantalla. En la documentación hay una lista completa con todos los flags que se pueden usar.

Crear una superficie a partir de un archivo .BMP:

```
SDL_Surface *superficieA = NULL;
superficieA = SDL_LoadBMP ("archivo.bmp");
```

Esto simplemente crea la superficie y la carga con la imagen del archivo. El tamaño de la superficie será el mismo que el tamaño de la imagen del archivo. Existen modos más complejos de crear las superficies, pero ya lo veremos.

Flip:

```
SDL_Flip (pantalla);
```

No hay mucho más que comentar

Blit:
Me dejo lo mejor para el final, la instrucción blit en SDL es tal que así:

```
SDL_BlitSurface (supOrigen, *rectOrigen, supDestino, *rectDestino);
```

rectOrigen y rectDestino son SDL_Rect. Este tipo es una estructura que guarda la información de un rectángulo. Por ejemplo, para copiar un rectángulo de 50x50 pixels de nuestra superficie origen a la posicion (x,y) = (100, 230) de nuestra superficie destino hay que hacer esto:

```
SDL_Rect rectOrigen;
rectOrigen .x = 0;
rectOrigen.y = 0;
rectOrigen .w = rectOrigen.h = 50;

SDL_Rect rectDestino;
rectDestino .x = 100;
```

```
rectDestino.y = 230;
rectDestino .w = rectDestino.h = 50;

SDL_BlitSurface (superficieA, &rectOrigen, pantalla, &rectDestino);
```

Si lo que queremos es copiar no un trozo, sino la superficie origen entera , solo tenemos que dejar el rectOrigen a NULL.

En el caso de que tengamos dos superficies de igual tamaño, y queramos copiar una completa sobre la otra, podemos dejar los dos rectángulos a NULL.

Y esto es todo por hoy, un saludo!