

Lazy Foo' Productions

[News](#) [FAQs](#) [Games](#) [Tutorials](#) [Articles](#) [Contact](#) [Donations](#)

Want to Code like a Pro?

Code Faster and More Efficiently Download a Free Trial Today!

www.slickedit.com

Ads by Google

Surface Loading and Blitting



Last Updated 1/23/08

This tutorial covers how to do Hello World SDL style.

A surface is just a fancy word for an image. Blitting a surface simply means applying a surface onto another surface. So this tutorial will teach you how to load a bitmap then apply it to the screen.

```
//The headers
#include "SDL/SDL.h"
#include <string>
```

Here are our headers for this program.

SDL.h is included because obviously we're going to need SDL's functions.

The string header is used because ... eh I just like std::string over char*

```
//The attributes of the screen
const int SCREEN_WIDTH = 640;
const int SCREEN_HEIGHT = 480;
const int SCREEN_BPP = 32;
```

Here we have the various attributes of the screen.

I'm pretty sure you can figure out SCREEN_WIDTH and SCREEN_HEIGHT are. SCREEN_BPP is the bits per-pixel. In all of the tutorials, 32-bit color will be used.

```
//The surfaces that will be used
SDL_Surface *message = NULL;
SDL_Surface *background = NULL;
SDL_Surface *screen = NULL;
```

These are the three images that are going to be used.

"background" is obviously going to be the background image, "message" is the bitmap that says "Hello World" and "screen" is obviously the screen.

Remember: its a good idea to always set your pointers to NULL if they're not pointing to anything.

```
SDL_Surface *load_image( std::string filename )
{
    //Temporary storage for the image that's loaded
    SDL_Surface* loadedImage = NULL;

    //The optimized image that will be used
    SDL_Surface* optimizedImage = NULL;
```

Here we have our image loading function.

What this function does is load the image, then returns a pointer to the optimized version of the loaded image.

The argument "filename" is the file name of the image to be loaded. "loadedImage" is the surface we get when the image is loaded. "optimizedImage" is the surface that is going to be used.

```
//Load the image
loadedImage = SDL_LoadBMP( filename.c_str() );
```

First the image is loaded using SDL_LoadBMP().

But it shouldn't be used immediately because the bitmap is a 24-bit. The screen is 32-bit and it's not a good idea to blit a surface onto another surface that is a different format because SDL will have to change the format on the fly which causes slow down.

```
//If nothing went wrong in loading the image
if( loadedImage != NULL )
{
    //Create an optimized image
    optimizedImage = SDL_DisplayFormat( loadedImage );

    //Free the old image
    SDL_FreeSurface( loadedImage );
}
```

Next we check if the image was loaded properly. If there was an error, loadedImage will be NULL.

If the image loaded fine, SDL_DisplayFormat() is called which creates a new version of "loadedImage" in the same format as the screen. The reason we do this is because when you try to stick one surface onto another one of a different format, SDL converts the the surface so they're the same format.

Creating the converted surface every time you blit wastes processing power which costs you speed. Because we convert the surface when we load it, when you want to apply the surface to the screen, the surface is already the same format as the screen. Now SDL won't have to convert it on the fly.

So now we have 2 surfaces, the old loaded image and the new optimized image.



SDL_DisplayFormat() created a new optimized surface but didn't get rid of the old one.

So we call SDL_FreeSurface() to get rid of the old loaded image.



```
//Return the optimized image
return optimizedImage;
}
```

Then the newly made optimized version of the loaded image is returned.

```
void apply_surface( int x, int y, SDL_Surface* source, SDL_Surface* destination )
{
    //Make a temporary rectangle to hold the offsets
    SDL_Rect offset;

    //Give the offsets to the rectangle
    offset.x = x;
    offset.y = y;
```

Here we have our surface blitting function.

It takes in the coordinates of where you want to blit the surface, the surface you're going to blit and the surface you're going to blit it to.

First we take the offsets and put them inside an SDL_Rect. We do this because SDL_BlitSurface() only accepts the offsets inside of an SDL_Rect.

An SDL_Rect is a data type that represents a rectangle. It has four members representing the X and Y offsets, the width and the height of a rectangle. Here we're only concerned about x and y data members.

```
//Blit the surface
SDL_BlitSurface( source, NULL, destination, &offset );
}
```

Now we actually blit the surface using SDL_BlitSurface().

The first argument is the surface we're using.

Don't worry about the second argument, we'll just set it to NULL for now.

The third argument is the surface we're going to blit on to.

The fourth argument holds the offsets to where on the destination the source is going to be applied.

```
int main( int argc, char* args[] )
{
```

Now we start the main function.

When using SDL, you should always use:

```
int main( int argc, char* args[] )
```

or

```
int main( int argc, char** args )
```

Using int main(), void main(), or any other kind won't work.

```
//Initialize all SDL subsystems
if( SDL_Init( SDL_INIT_EVERYTHING ) == -1 )
{
    return 1;
}
```

Here we start up SDL using SDL_Init().

We give SDL_Init() SDL_INIT_EVERYTHING, which starts up every SDL subsystem. SDL subsystems are things like the video, audio, timers, etc that are the individual engine components used to make a game.

We're not going to use every subsystem but it's not going to hurt us if they're initialized anyway.

If SDL can't initialize, it returns -1. In this case we handle the error by returning 1, which will end the program.

```
//Set up the screen
screen = SDL_SetVideoMode( SCREEN_WIDTH, SCREEN_HEIGHT, SCREEN_BPP, SDL_SWSURFACE
```

Now it's time to make our window and get a pointer to the window's surface so we can blit images to the screen.

You already know what the first 3 arguments do. The fourth argument creates the screen surface in system memory.

```
//If there was an error in setting up the screen
if( screen == NULL )
{
    return 1;
}
```

If there was a problem in making the screen pop up, screen will be set to NULL.

```
//Set the window caption
SDL_WM_SetCaption( "Hello World", NULL );
```

Here the caption is set to "Hello World".

The caption is this part of the window:



```
//Load the images
message = load_image( "hello_world.bmp" );
background = load_image( "background.bmp" );
```

Now the images are loaded using the image loading function we made.

```
//Apply the background to the screen
apply_surface( 0, 0, background, screen );
```

Now it's time to apply the background with the function we made.

Before we blitted the background, the screen looked like this:



But now that we blitted the background image, the screen looks like this in memory:

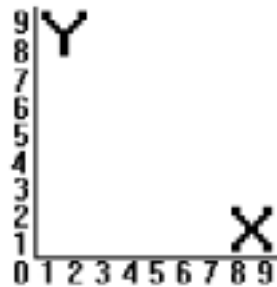


When you blit, you copy the pixels from one surface onto another. So now the screen looks like the background image we loaded.

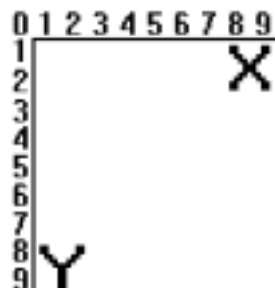
```
//Apply the message to the screen
apply_surface( 180, 140, message, screen );
```

Now we're going to apply the message surface onto the screen at x offset 180 and y offset 140.

The thing is SDL coordinate system doesn't work like this:



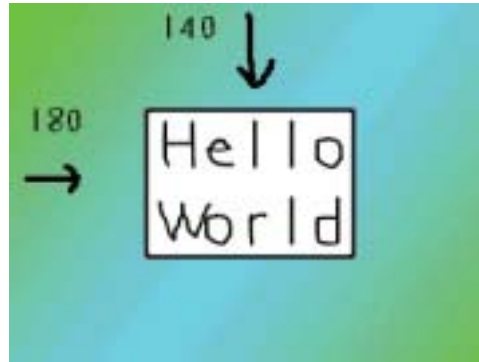
SDL's coordinate system works like this:



So the origin (0,0) is at the top left corner instead of the bottom left.

So when you blit the message surface, it's going to blit it 180 pixels right, and 140 pixels down

from the origin in the top left corner:



SDL's coordinate system is awkward at first but you'll get used to it.

```
//Update the screen
if( SDL_Flip( screen ) == -1 )
{
    return 1;
}
```

Even though we have applied our surfaces, the screen we see is still blank.

Now we have to update the screen using `SDL_Flip()` so that the screen surface we have in memory matches the one shown on the screen.

If there's an error it will return -1.

```
//Wait 2 seconds
SDL_Delay( 2000 );
```

We call `SDL_Delay()` so that the window doesn't just flash on the screen for a split second. `SDL_Delay()` accepts time in milliseconds, or 1/1000 of a second.

So the window will stay up for 2000/1000 of a second or 2 seconds.

```
//Free the surfaces
SDL_FreeSurface( message );
SDL_FreeSurface( background );

//Quit SDL
SDL_Quit();

//Return
return 0;
}
```

Now we do the end of the program clean up.

`SDL_FreeSurface()` is used to get rid of the surfaces we loaded since we're not using them anymore. If we don't free the memory we used, we will cause a memory leak.

Then `SDL_Quit()` is called to quit SDL. Then we return 0, ending the program. You may be asking yourself "why aren't we freeing the screen surface?". Don't worry. `SDL_Quit()` will take care of that for us.

If you run the program and the images don't show up or the window flashes for a second and you find in `stderr.txt`:

Fatal signal: Segmentation Fault (SDL Parachute Deployed)

It's because the program tried to access memory it wasn't supposed to. Odds are it's because it tried to access NULL when `apply_surface()` was called. This means you need to make sure the bitmap files are in the same directory as the program.


If the window pops up and the image doesn't show up, again make sure the bitmaps are in the

same folder as the program or in the project directory.

Also if you're using Visual Studio and you get the error "**The application failed to start because the application configuration is incorrect. Reinstalling the application may fix this problem.**", it's because you don't have the service pack update installed. [Do not forget to have the latest version of your compiler/IDE with the service pack update for your compiler/IDE and the platform SDK installed or SDL will not work with Visual Studio.](#)

Download the media and source code for this tutorial [here](#).

International Arbitration Int Lawyer team can help before ICC ICSID, BIT, UNCITRAL arbitrations	Shipping Insurance Online All Commodities Most competitive. Call & See, Direct and Online!
---	--



[News](#) [FAQs](#) [Games](#) [Tutorials](#) [Articles](#) [Contact](#) [Donations](#)

Copyright Lazy Foo' Productions 2004-2008