Drawing Sprites with SDL

Note: This tutorial assumes that you already know how to display a window with SDL.

Loading and Drawing Sprites

To display a sprite in SDL, you first need to load a bitmap file. This task is accomplished using `SDL_LoadBMP()`, which takes the name of the bitmap file you want to load and returns a pointer to a surface that stores the bitmap's data.

You draw the bitmap using `SDL_BlitSurface()`, which takes four parameters. The first is a pointer to the source surface, which you got from `SDL_LoadBMP()`.

The second is an `SDL_Rect` storing the location in the source bitmap of the sprite you want drawn. The `SDL_Rect` is a very simple structure. It just stores an x,y value pair as well as width and height values.

One thing you need to know before going on is that in computer graphics, a positive y value almost always means down. Starting from the top of the screen and going down, you would go (0, 0)->(0, 1)->(0, 2), etc.

The third parameter is a pointer to the screen surface, which you got from `SDL_SetVideoMode()`.

The last parameter is an `SDL_Rect` storing where you want the image drawn to on the screen.

Although you can draw the entire bitmap at once, most of the time you'll only want to draw parts of the bitmap. This is because most bitmaps used in 2D games store more than one image.

The term "blit" stands for "block image transfer", in case you wanted to know.

A call to `SDL_BlitSurface()` will look like this (assume that bitmap is a previously loaded surface and that screen is the screen surface):

```
// Part of the bitmap that we want to draw
SDL_Rect source;
source.x = 24;
source.y = 63;
source.w = 65;
source.h = 44;
```

```
// Part of the screen we want to draw the sprite to
SDL_Rect destination;
destination.x = 100;
destination.y = 100;
destination.w = 65;
destination.h = 44;

SDL_BlitSurface(bitmap, &source, screen, &destination);
```

Before drawing a surface, we almost always need to fill in `SDL_Rect` structures to specify what we want drawn and where we want it drawn to. The only times we don't need to fill these structures in is when we want the entire surface drawn. In this case, we just pass in `NULL`.

Once we've drawn everything we need to, we have to call `SDL_Flip()` to tell SDL to display our scene. We pass `SDL_Flip()` our screen surface as a parameter.

Here is the code to load a window and draw a sprite. Most of the code is from the previous tutorial, so I've bolded the parts related to drawing the sprite. The bitmap I've used can be downloaded here. Make sure to put it in your main project directory before running the code.

```
#include "SDL.h"

const int WINDOW_WIDTH = 640;
const int WINDOW_HEIGHT = 480;
const char* WINDOW_TITLE = "SDL Start";

int main(int argc, char **argv)
{
   SDL_Init( SDL_INIT_VIDEO );

   SDL_Surface* screen = SDL_SetVideoMode( WINDOW_WIDTH,
WINDOW_HEIGHT, 0,
      SDL_HWSURFACE | SDL_DOUBLEBUF );
   SDL_WM_SetCaption( WINDOW_TITLE, 0 );

   SDL_Surface* bitmap = SDL_LoadBMP("bat.bmp");

   // Part of the bitmap that we want to draw
   SDL_Rect source;
   source.x = 24;
   source.y = 63;
   source.w = 65;
   source.h = 44;

   // Part of the screen we want to draw the sprite to
   SDL_Rect destination;
```

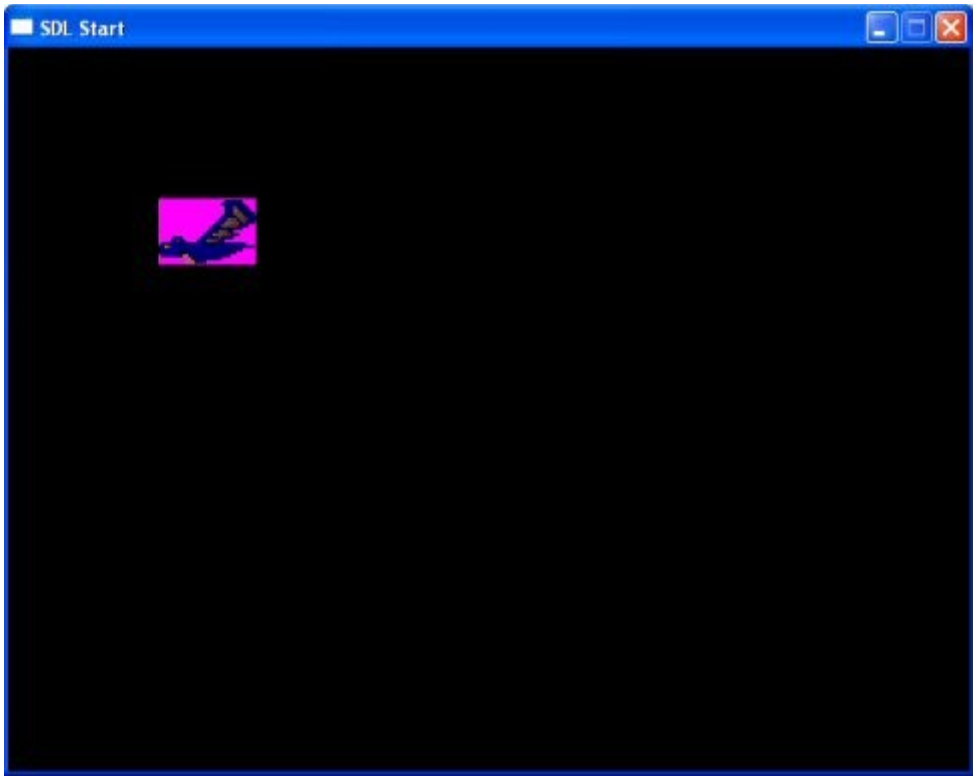
```
    destination.x = 100;
    destination.y = 100;
    destination.w = 65;
    destination.h = 44;

    SDL_Event event;
    bool gameRunning = true;

    while (gameRunning)
    {
        if (SDL_PollEvent(&event))
        {
            if (event.type == SDL_QUIT)
            {
                gameRunning = false;
            }
        }
        SDL_BlitSurface(bitmap, &source, screen, &destination);
        SDL_Flip(screen);
    }
    SDL_FreeSurface(bitmap);
    SDL_Quit();
    return 0;
}
```

There are two things to make note of in the code above. One is that you should free any surfaces that you create yourself. The other is that you need to draw your entire scene before calling `SDL_Flip()`. The call to `SDL_Flip()` should be the last call in your rendering code.

Running the code now should give you something like this:


SDL Start

Setting a Transparent Color

Since most of our sprites won't be perfect squares, we need some way to make parts of them transparent. A common way to do this is to decide on a color that should never be drawn. This color is normally something you'd never use in a game, like magenta (255, 0, 255).

We define this color to be transparent with a call to `SDL_SetColorKey()`. This is done like so:

```
SDL_SetColorKey(surface, SDL_SRCCOLORKEY, SDL_MapRGB(surface->format, 255, 0, 255) );
```

The first parameter is a pointer to the surface that contains the color we're setting to be transparent. This allows us to set different transparent colors on different surfaces, if we really want to do that.

The second parameter is a flag. Passing it `SDL_SRCCOLORKEY` specifies that we want what we pass as the third parameter to be the transparent color.

The third parameter is the color itself. The color needs to be passed to `SDL_SetColorKey()` in the surface's format. We call `SDL_MapRGB()` to get this. `SDL_MapRGB()` takes a pixel format as well as the RGB values of the color we want. You can get the pixel format from the surface. It's stored in the format variable.

If you don't understand some of this stuff, don't worry. If you end up doing more advanced things with SDL, it might matter, but for current purposes you just need to set a transparent color. Doing so is quite easy (it only takes one line!).

In the code above, add the following line immediately after the bitmap surface is created:

```
SDL_SetColorKey( bitmap, SDL_SRCCOLORKEY, SDL_MapRGB(bitmap->format, 255, 0, 255) );
```

Running the code now should give you this:

Clearing the Screen

At the moment, there's no problem with the way we're drawing things. We'll run into problems when we start moving things around though. The problem is that we aren't clearing the screen before we start drawing. If something moves, it will still get drawn at its previous location.

To clear the screen, we use `SDL_FillRect()`, which takes three parameters. The first parameter is a pointer to an `SDL_Surface`. This surface will be cleared to a given color. To clear the screen, we pass in the screen buffer.

The second parameter is an `SDL_Rect` storing the area of the surface that we want cleared. Passing in `NULL` will clear the entire surface.

The third parameter is the color we want to clear the surface to. This works just like it did with `SDL_SetColorKey()`.

The following line of code will clear the screen to black. It should be placed before any other drawing code.

```
SDL_FillRect(screen, NULL, SDL_MapRGB(screen->format, 0, 0, 0));
```

Exercises

Ex 1) The only thing I find cumbersome about drawing sprites with SDL is that you have to fill two SDL_Rect structures every time. This is of course easily fixed by writing a function that does it for you. Try writing such a function now, and don't feel bad if you have to look back to remember an SDL function name or its parameters. Knowing how to use SDL is much more useful than memorizing it.

Here's an example of a function that draws a sprite:

```
void drawSprite(SDL_Surface* imageSurface,
                SDL_Surface* screenSurface,
                int srcX, int srcY,
                int dstX, int dstY,
                int width, int height)
{
   SDL_Rect srcRect;
   srcRect.x = srcX;
   srcRect.y = srcY;
   srcRect.w = width;
   srcRect.h = height;

   SDL_Rect dstRect;
   dstRect.x = dstX;
   dstRect.y = dstY;
   dstRect.w = width;
   dstRect.h = height;

   SDL_BlitSurface(imageSurface, &srcRect, screenSurface, &dstRect);
}
```