Lesson 02 – Bitmaps

Home

Bitmaps & Blitting | Loading a Bitmap | Saving a Bitmap | Blitting to the Window | Optimized Surfaces | Scaling

Bitmaps & Blitting

You've probably heard of bitmaps before; the possibly out-of-date ".bmp" images. Well, a bitmap is technically just an image stored in a format where pixel data is represented by a block of memory, or you could say, a map of bits. The "bmp" format denotes an image where the color values of each pixel (monochrome, RGB, RGBA, etc) are simply stored sequentially after a header. Other formats such as "png" and "jpg" are still a from of bitmap, but make use of compression technologies to reduce file size (ideally) without sacrificing image quality.

In software (CPU) 2D rendering, once a bitmap is loaded into memory, the CPU simply modifies and/or copies pixel values from a loaded bitmap to the region representing the window. This is called "blitting." Blitting can be

Example Program

Download

```
#include <iostream>
#include <SDL.h>

using namespace std;

bool init();
void kill();
bool load();

// Pointers to our window and surfaces
SDL_Window* window;
SDL_Surface* winSurface;
SDL_Surface* image1;
SDL_Surface* image2;

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

	if ( !init() ) return 1;

	if ( !load() ) return 1;

	// Blit image to entire window
	SDL_BlitSurface( image1, NULL, winSurface, NULL );

	// Blit image to scaled portion of window
```

more complex than simply copying pixels; it can involve resizing, stretching, flipping, or otherwise post-processing images as well.

SDL provides functionality for loading and blitting bitmaps. The blitting API can be useful, but it is relatively limiting. By default SDL can only load "bmp" format bitmaps—the extension SDL_Image must be used to load more image formats, such as "png," "jpg," "gif," or "tif." We'll learn about more extension libraries in lesson 07. Furthermore, in lesson 04, we'll learn about the much more fully-featured hardware (GPU) rendering API.

Loading a Bitmap

Last lesson, I mentioned how SDL abstracts pixel memory to surfaces. A surface can represent a loaded bitmap, a new texture you're creating, or even the window you're drawing to.

To load a bitmap into a surface, use the function SDL_LoadBMP(). Its usage is very straightforward—simply send it the name of the .bmp file you wish to load (relative to the location of the executable). If the load succeeds, you will receive a pointer to a new SDL_Surface containing your bitmap data. If the load fails, the function will return NULL.

```
        SDL_Rect dest;
        dest.x = 160;
        dest.y = 120;
        dest.w = 320;
        dest.h = 240;
        SDL_BlitScaled( image2, NULL, winSurface, &dest );

        // Update window
        SDL_UpdateWindowSurface( window );
        system("pause");

        kill();
        return 0;
}

bool load() {
        // Temporary surfaces to load images into
                // This should use only 1 temp surface, but for conciseness we use two
        SDL_Surface *temp1, *temp2;

        // Load images
        temp1 = SDL_LoadBMP("test1.bmp");
        temp2 = SDL_LoadBMP("test2.bmp");

        // Make sure loads succeeded
        if ( !temp1 || !temp2 ) {
                cout << "Error loading image: " << SDL_GetError() << endl;
                system("pause");
                return false;
        }

        // Format surfaces
        image1 = SDL_ConvertSurface( temp1, winSurface->format, 0 );
        image2 = SDL_ConvertSurface( temp2, winSurface->format, 0 );

        // Free temporary surfaces
        SDL_FreeSurface( temp1 );
        SDL_FreeSurface( temp2 );

        // Make sure format succeeded
        if ( !image1 || !image2 ) {
```

```
SDL_Surface* image =
SDL_LoadBMP(
"image.bmp" );

if ( !image ) {
	// load failed
}
```

Freeing a surface is just as simple as allocating one—when a surface is no longer in use, the function SDL_FreeSurface() can be used to destroy it and free the memory.

```
SDL_FreeSurface( image );
```

Saving a Bitmap

The opposite of loading a bitmap is, of course, saving one. The function SDL_SaveBMP() just does that. Pass it the surface containing the bitmap data and the name of the bitmap file to save. The function will return 0 on success or a negative value on failure.

```
int result = SDL_SaveBMP(
surface, "saved.bmp" );

if ( result < 0 ) {
	// save failed
}
```

Blitting to the Window

```
		cout << "Error converting
surface: " << SDL_GetError() <<
endl;
		system("pause");
		return false;
	}
	return true;
}

bool init() {
	// See last example for
comments
	if ( SDL_Init(
SDL_INIT_EVERYTHING ) < 0 ) {
		cout << "Error initializing
SDL: " << SDL_GetError() << endl;
		system("pause");
		return false;
	}

	window = SDL_CreateWindow(
"Example",
SDL_WINDOWPOS_UNDEFINED,
SDL_WINDOWPOS_UNDEFINED,
640, 480, SDL_WINDOW_SHOWN );
	if ( !window ) {
		cout << "Error creating
window: " << SDL_GetError()  <<
endl;
		system("pause");
		return false;
	}

	winSurface =
SDL_GetWindowSurface( window );
	if ( !winSurface ) {
		cout << "Error getting
surface: " << SDL_GetError() <<
endl;
		system("pause");
		return false;
	}
	return true;
}

void kill() {
	// Free images
	SDL_FreeSurface( image1 );
	SDL_FreeSurface( image2 );

	// Quit
```

```
        SDL_DestroyWindow( window );
        SDL_Quit();
}
```

The basic way to blit to a surface is with SDL_BlitSurface(). The function takes a source surface and region and blits the pixels to a destination surface and region. These regions are specified with the SDL_Rect structure.

The source rectangle contains the coordinates and size of the area from the source surface to be copied. NULL can be passed instead of this rectangle to use the entire surface. Only the x and y coordinates of the destination rectangle are actually used, and clipping will be done automatically.

```
SDL_Rect dest;
dest.x = 100;
dest.y = 50;

int result = SDL_BlitSurface( surface, NULL, windowSurface, &dest );

if ( result < 0 ) {
        // blit failed
}
```

Optimized Surfaces

While not absolutely necessary, a surface can be optimized with regard to another to improve the speed of blits. Essentially, this just means changing the pixel format of a surface to match another. Then, the program can simply copy the data when blitting, rather than also having to change the format.

This is done with the function SDL_ConvertSurface(). This function takes the surface you wish to convert and the format to convert to. This format is specified in the "format" data member of the target surface, which should almost always be the window surface. If the conversion succeeds, the function returns a new, converted surface. It does not change or free the original (unconverted) surface—you must do that yourself.

```
SDL_Surface* load = SDL_LoadBMP("test.bmp");

SDL_Surface* image = SDL_ConvertSurface( load, windowSurface->format, 0 );

if ( !image ) {
        // convert failed
} else {
        // convert succeeded
        SDL_FreeSurface( load );
```

```
        load = NULL;
}
```

Scaling

With SDL_BlitSurface(), you can stamp bitmaps anywhere you wish. However, I mentioned that only the x and y coordinates of the destination rectangle are taken into account. This means that the blitted image will be the same size every time. To stretch or scale your bitmap to fit an arbitrary area, you can use the function SDL_BlitScaled(). This function takes the same parameters as SDL_BlitSurface(), except that it takes into account the width and height of the destination rectangle.

```
SDL_Rect dest;
dest.x = 100;
dest.y = 50;
dest.w = 200;
dest.h = 100;

int result = SDL_BlitScaled( surface, NULL, windowSurface, &dest );

if ( result < 0 ) {
        // blit failed
}
```
