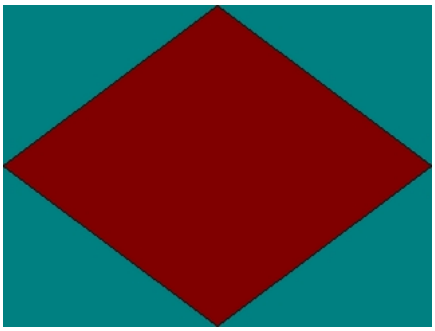
Pixel Manipulation and Surface Flipping



Last Updated 3/28/10

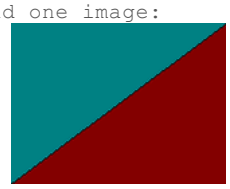
SDL has no built in functions to flip surfaces. So we'll just have to flip surfaces ourselves by playing with the pixels. In this tutorial we will read/write pixels to make our own surface flipping function.

```
bool load_files()
{
    //Load the image
    topLeft = load_image( "corner.png" );

    //If there was an problem loading the image
    if( topLeft == NULL )
    {
        return false;
    }

    //If everything loaded fine
    return true;
}
```

In our loading function we only load one image:



In order to show all four corners we will have to create them ourselves by flipping the image we loaded.

```
Uint32 get_pixel32( SDL_Surface *surface, int x, int y )
{
    //Convert the pixels to 32 bit
    Uint32 *pixels = (Uint32 *)surface->pixels;

    //Get the requested pixel
    return pixels[ ( y * surface->w ) + x ];
}

void put_pixel32( SDL_Surface *surface, int x, int y, Uint32 pixel )
{
    //Convert the pixels to 32 bit
    Uint32 *pixels = (Uint32 *)surface->pixels;
```

```
    //Set the pixel
    pixels[ ( y * surface->w ) + x ] = pixel;
}
```

Here's our functions to get and put pixels. In case you missed the bitmap font tutorial, here's a quick review on how pixel access works:

First thing we do is convert the pixel pointer from type void to 32bit integer so we can properly access them. After all, a surface's pixels are nothing more than an array of 32bit integers. Then we get or set the requested pixel.

You maybe be wondering why I don't just go "return pixels[x][y]".

The thing is the pixels aren't stored like this:

They're stored like this:

in a single dimensional array. It's because different operating systems store 2D arrays differently (At least I think that's why).

So to retrieve the red pixel from the array we multiply the y offset by the width and add the x offset.

These functions only work for 32-bit surfaces. You'll have to make one of your own if you're using a different format.

You can learn more about pixels in article 3.

```
SDL_Surface *flip_surface( SDL_Surface *surface, int flags )
{
    //Pointer to the soon to be flipped surface
    SDL_Surface *flipped = NULL;

    //If the image is color keyed
    if( surface->flags & SDL_SRCCOLORKEY )
    {
        flipped = SDL_CreateRGBSurface( SDL_SWSURFACE, surface->w, surface->h, surface->format->Bit
    }
    //Otherwise
    else
    {
        flipped = SDL_CreateRGBSurface( SDL_SWSURFACE, surface->w, surface->h, surface->format->Bit
    }
```

Now here's our surface flipping function.

At the top we create a blank surface using SDL_CreateRGBSurface(). It may look complicated, but ultimately we're just creating a surface of the same size and format as the surface we are given.

Before we can create the blank surface, we have to check if the surface is color keyed. If it is, we set the alpha mask to 0 on the new blank surface. The reason is if the alpha mask is anything but 0, the color key gets ignored. If the source surface is not color keyed, we just copy its alpha mask to the new blank surface.

Take a look at the SDL Documentation to see how SDL_CreateRGBSurface() works.

```
    //If the surface must be locked
    if( SDL_MUSTLOCK( surface ) )
    {
        //Lock the surface
        SDL_LockSurface( surface );
    }
```

Before we can start altering pixels we have to check if the surface has to be locked before accessing the pixels using SDL_MUSTLOCK(). If SDL_MUSTLOCK() says we have to lock the surface, we use SDL_LockSurface() to lock the surface.

Now that the surface is locked it's time to mess with the pixels.

```
    //Go through columns
```

```
    for( int x = 0, rx = flipped->w - 1; x < flipped->w; x++, rx-- )
    {
        //Go through rows
        for( int y = 0, ry = flipped->h - 1; y < flipped->h; y++, ry-- )
        {
```

Here is our nested loops used for going through the pixels. You may be wondering why we declare 2 integers in our for loops. The reason is that when you flip pixels, you have to read them in one direction and write them in reverse.

In the x loop, we declare "x" and "rx". "x" is initialized to 0, and "rx" (which stands for reverse x) is initialized to the width minus 1 which is the end of the surface.

Then in the middle the condition is normal. At the end "x" is incremented and "rx" is decremented. "x" will start at the beginning and go forward, "rx" starts at the end and goes backward.

So if the surface's width was 10, the for loop will cycle like so:
"x" : 0 1 2 3 4 5 6 7 8 9
"rx" : 9 8 7 6 5 4 3 2 1 0

I'm pretty sure you can now figure out what "y" and "ry" do.

```
            //Get pixel
            Uint32 pixel = get_pixel32( surface, x, y );

            //Copy pixel
            if( ( flags & FLIP_VERTICAL ) && ( flags & FLIP_HORIZONTAL ) )
            {
                put_pixel32( flipped, rx, ry, pixel );
            }
            else if( flags & FLIP_HORIZONTAL )
            {
                put_pixel32( flipped, rx, y, pixel );
            }
            else if( flags & FLIP_VERTICAL )
            {
                put_pixel32( flipped, x, ry, pixel );
            }
        }
    }
```

Here's the middle of the nested loops.

First we read in a pixel from the source surface. Then if the user passed in the FLIP_VERTICAL and the FLIP_HORIZONTAL flags, we write the pixels to the blank surface right to left, bottom to top.

If the user just passed the FLIP_VERTICAL flag, we write the pixels to the blank surface left to right, bottom to top.

If the user just passed the FLIP_HORIZONTAL flag, we write the pixels to the blank surface right to left, top to bottom.

If you're wondering what the flag values are, they're near the top of the source.

```
    //Unlock surface
    if( SDL_MUSTLOCK( surface ) )
    {
        SDL_UnlockSurface( surface );
    }

    //Copy color key
    if( surface->flags & SDL_SRCCOLORKEY )
    {
        SDL_SetColorKey( flipped, SDL_RLEACCEL | SDL_SRCCOLORKEY, surface->format->colorkey );
    }

    //Return flipped surface
    return flipped;
}
```

At the end of our surface flipping function, we check if the surface had to be locked. If it did, we unlock it using SDL_UnlockSurface().

Then if the surface we had to flip was color keyed, we copy the color key from the original surface to the new flipped surface we made.

Lastly we return a pointer to our newly created flipped surface.

```
    //Flip surfaces
    topRight = flip_surface( topLeft, FLIP_HORIZONTAL );
    bottomLeft = flip_surface( topLeft, FLIP_VERTICAL );
    bottomRight = flip_surface( topLeft, FLIP_HORIZONTAL | FLIP_VERTICAL );

    //Apply the images to the screen
    apply_surface( 0, 0, topLeft, screen );
    apply_surface( 320, 0, topRight, screen );
    apply_surface( 0, 240, bottomLeft, screen );
    apply_surface( 320, 240, bottomRight, screen );

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

and here's the surface flipping function in action. Then the surfaces are applied and the screen is updated to show the diamond pattern.

Download the media and source code for this tutorial here.

Previous Tutorial Next Tutorial

SDL Forums SDL Tutorials Articles OpenGL Tutorials OpenGL Forums
News FAQs Games Contact Donations