Compressed Graphics Tutorial

In this tutorial, we will edit a compressed graphic with the help of four utilities. We will first decompress the graphic from the ROM, modify its palette in the ROM, edit the graphic and finally compress the new graphic back to the ROM.

The Tools

The first utility you will need is Peer Sprite Viewer (PSV). This utility can compress and decompress data from the ROMs for FF3us and Chrono Trigger. Second, you will need YY-CHR. There is a .NET and C++ versions. I personally go with the C++ one; it is older, but it has more features that have not yet been ported to the .NET version. The image shown here is from the C++ version. The other utility required is SNESpal, which can modify color palettes in the ROM.

Finally, you will need an all purpose image editor. My favorite editor for many things is Gimp. It is great for icon making, spriting, and other graphic editing. Visit gimp.org for the latest version.

Decompressing the Graphic

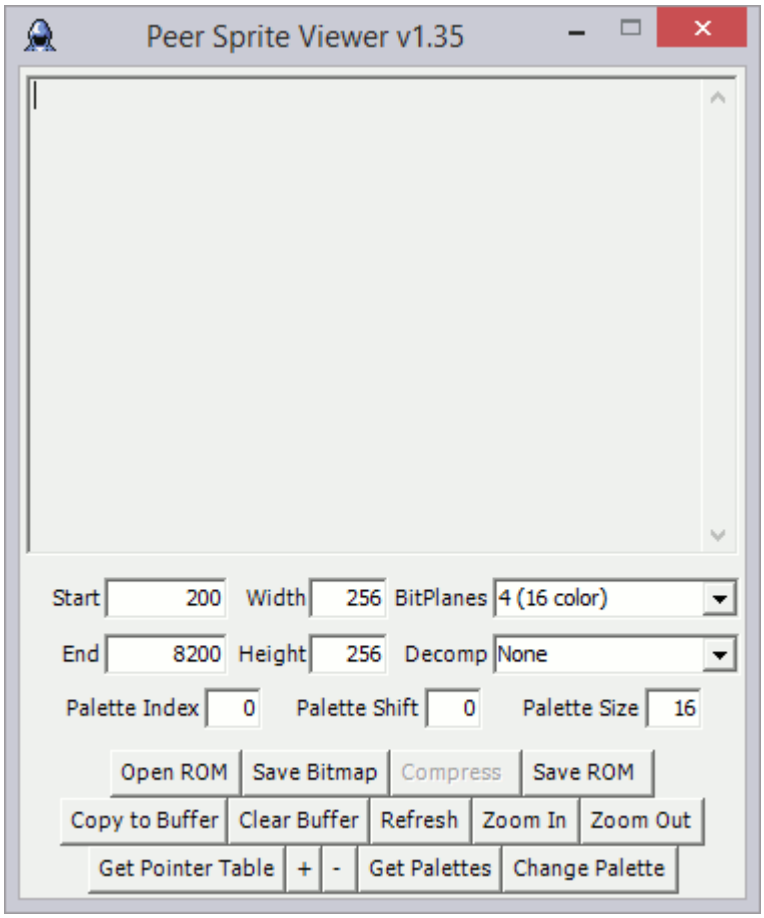
The first thing to do is to locate what you need to decompress. We have a Compressed Data List which includes graphics. Alternatively, you can look at the ROM Map, but everything should theoretically be in the Compressed Data List. In this tutorial, we will edit the battle shield used by party members when they block with their shields. It is part of a larger *graphic sheet*, meaning you cannot just decompress / compress the shield alone. Modifying the shield will have an impact on the size and arrangement of the data once re-compressed.

The battle graphics we are looking for are at $D2E000 (unheadered ROM). This is $12E000 when using absolute addressing. If you open a hex editor and go to this address, you'll see the following:

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0012E000   44 0B DE DD 1F 0C 04 0C 0C DD 0F 01 01 F0 DE 27
```

The first two bytes are always an inverted word representing the compressed data size. This means that our graphics are $0B44 bytes in size (including the first two bytes themselves), so the graphics are stored between $D2E000 and $D2EB43. The wiki lists the graphic as ending at $D2EBFF, which is not totally false since $D2EB44-$D2EBFF is unused space. While the starting offset is very likely to be correct, I suggest that you check the data size to ensure that you only decompress the data that makes up the graphic.

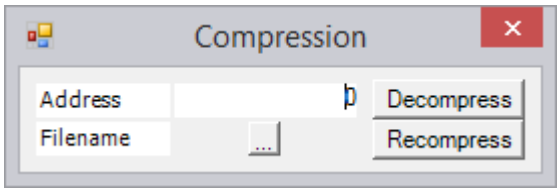
Open your ROM in Peer Sprite Viewer (PSV). You'll see the main interface:



Start is your starting offset, and **End** is your ending offset, **Width** and **Height** are the desired dimensions of your decompressed graphic, **BitPlanes** specifies the type of graphic after decompression, and **Decomp** is the compression type. I have always kept the **BitPlanes** value as *4 (16 colors)* for FF3us, and obviously **Decomp** should be set to *Final Fantasy VI*. This screenshot shows the correct values to enter for our example. Now click **Copy to Buffer**.

After clicking, you should see the graphics in another window. If the sheet size is incorrect, you can adjust it by changing the **Width**/**Height** fields and clicking on **Copy to Buffer** again. You may have to try several times to find the correct dimensions. The following two images show the same graphics. The first screenshot is 256×256 (the default value), and the second one is 128×128. The only difference is that it will be easier to edit if you find the correct size.

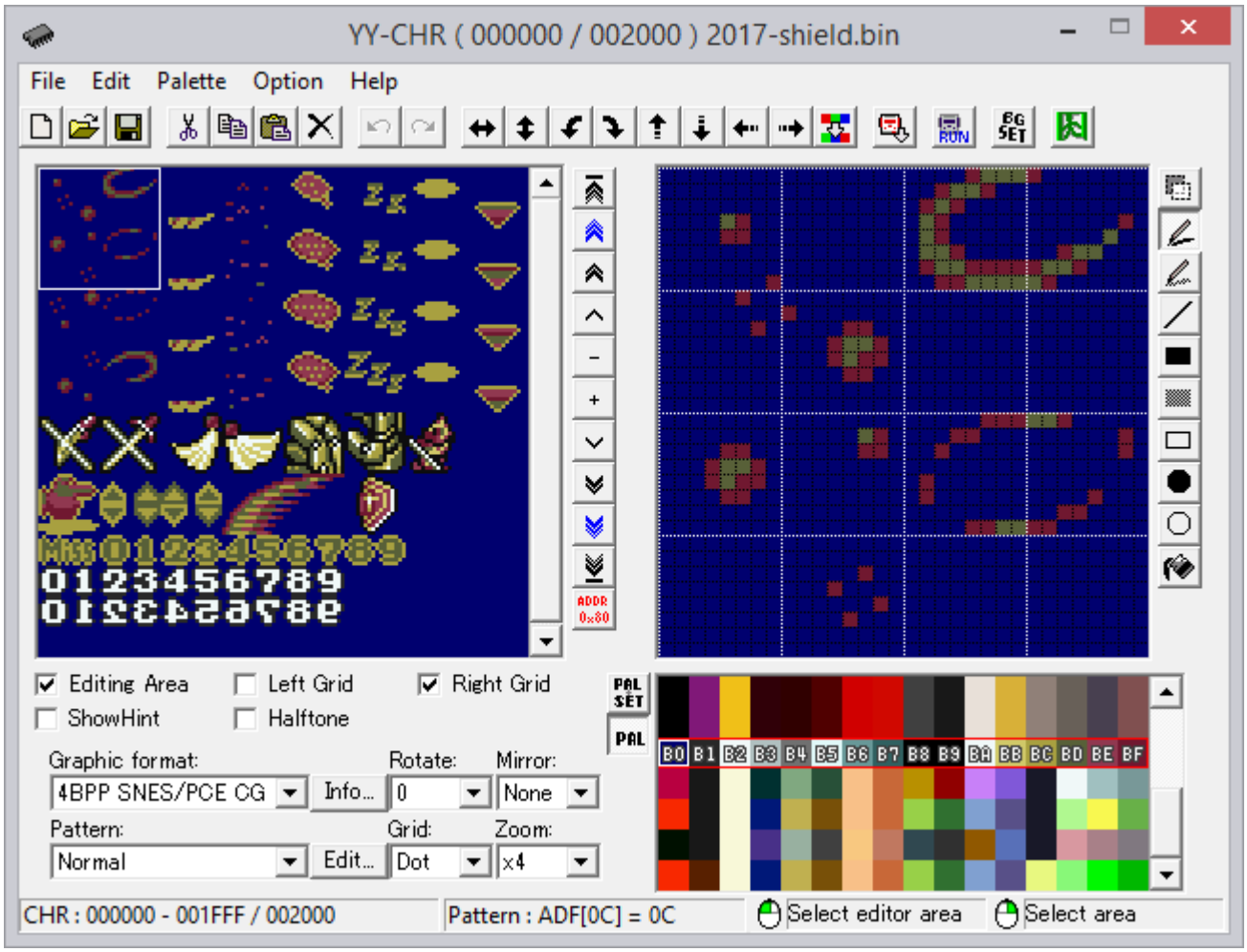
The next step is saving this graphic to a file. Click **Compress**, and you will see this window:



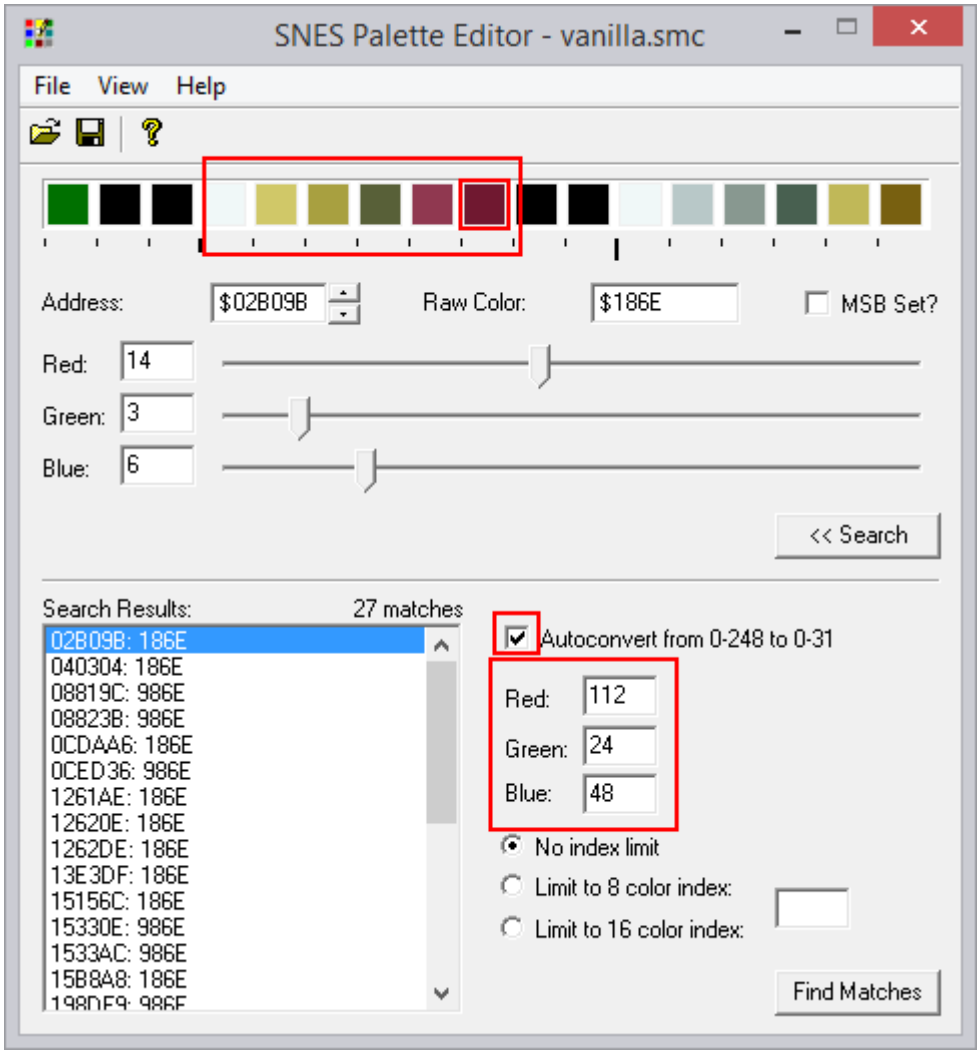
Enter the same starting offset ($12E000) and click **...** to enter a filename with the extension of your choice (a lot of people use .bin to hint at a binary file). Finally, click **Decompress**, as shown here. Your GFX is now saved in a file that we will edit with YY-CHR.

Finding and Editing the Palette

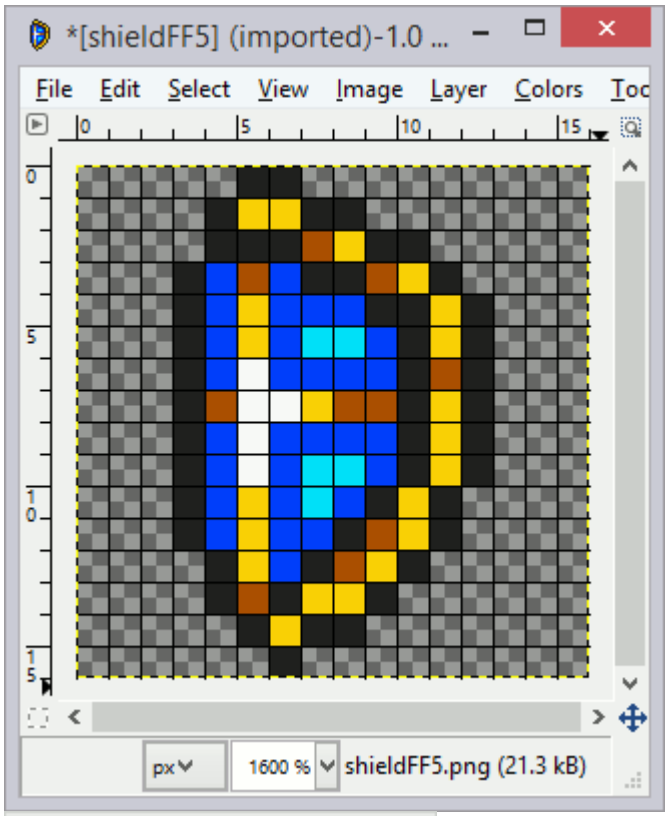
Now we need to use YY-CHR. If you want the compressed files and a savestate that will allow you to load the palette correctly, you can download them here. When you load the decompressed GFX, it should look like this. No need to worry about the incorrect palette; we will load it from a ZSNES savestate that was taken when the shield was on screen. Select *Palette → Load Emulator state file* and choose your savestate. Click **Pal** at the bottom right window and try each palette until you get the correct one. It should now look like this:

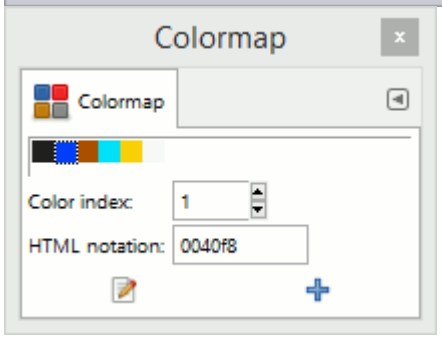


When you right click on a palette color, you will see its RGB value as shown here. In this example, the RGB values of color BB are D0,C8, and 68, which are in hexadecimal notation. You'll need to convert the hex values into decimal (0-255) to find the pallete with SNESpal. When entering the values in SNESpal, make sure you check the autoconvert checkbox. The example in the next picture highlights the colors that correspond with colors BA-BF in the previous picture. There is no real trick to find the color except looking at the search results and finding a few colors that are in same order as in YY-CHR. Try searching for rare or uncommon colors in SNESpal, and do not search for black or white as it will give tons of results.



Right away, you can edit your palette to something new. It can be unclear how many exclusive colors your GFX has (some colors might be shared with other GFX), but try reusing the white and black that is already available in the palette, and only edit what seems to be exclusive. In this way, you'll minimize potential conflicts. For indexed images, Gimp Colormap is useful to get the right RGB values to edit in SNESpal, but you need to divide by 8 to do a hexadecimal to 1-31 scale conversion.

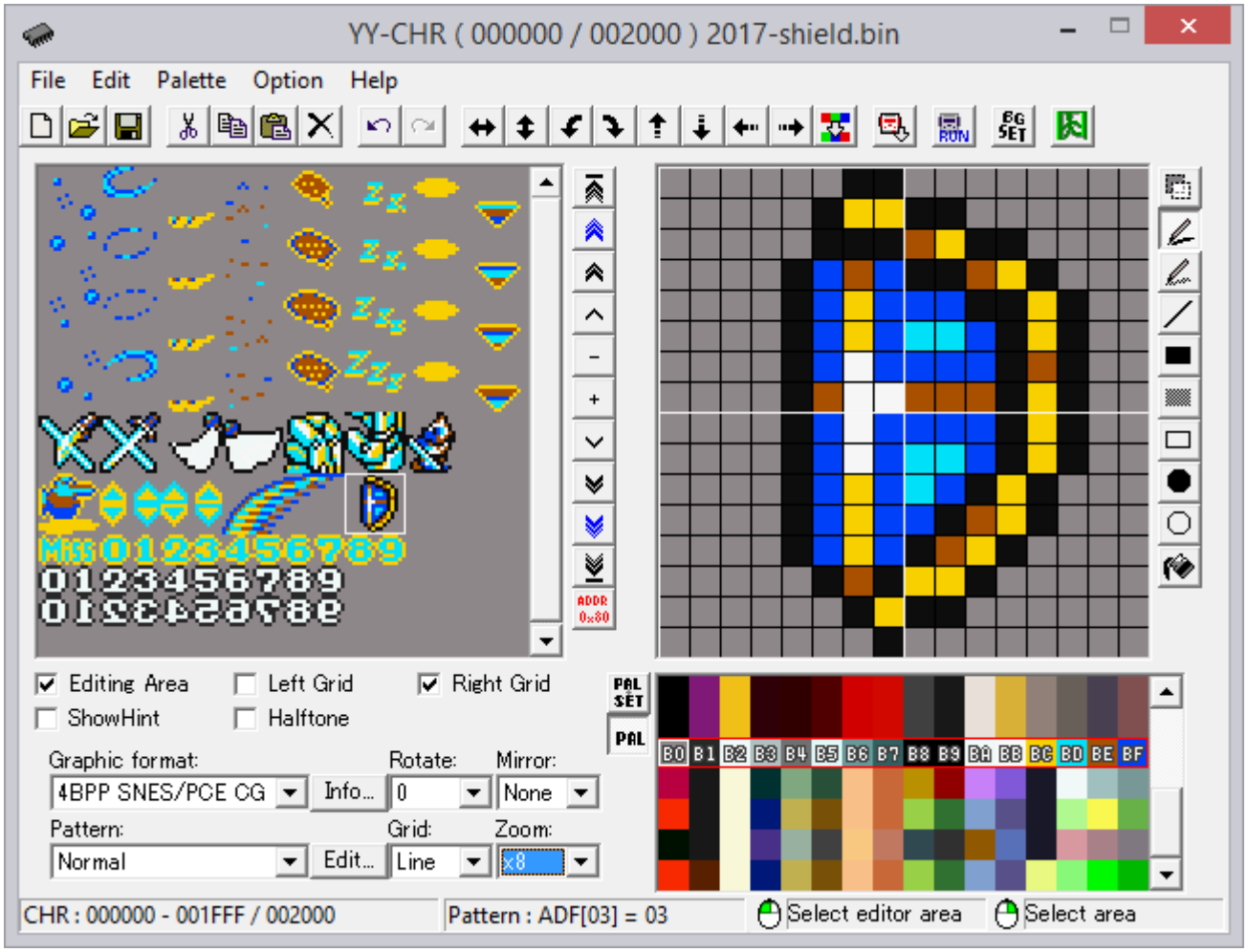




Editing the Graphic

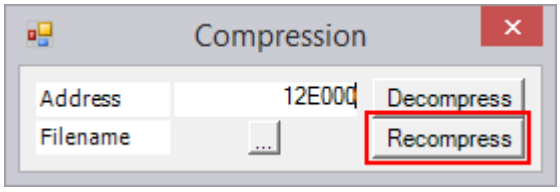
This is the fun part! (to me...)

First you can edit your YY-CHR palette to the palette you made in Gimp and SNESpal, as shown here. Next, just edit the GFX, and the final result should look like this:



Recompressing the Graphic

You can now reload your ROM in PSV by entering the same info as if you were decompressing. In this case, you should enter $D2EBFF as the ending offset to have some extra room if your GFX happens to take more space (I have not validated if this really has an impact or not). Once again, click **Compress** and enter your starting offset in the compression window. Finally, click **Recompress** and save your ROM.



The result in-game should be this one:

Conclusion

I hope I made this tutorial clear enough to give you a good indication of how to proceed with editing compressed graphics. If you ever get a compression error due to lack of space, there is no simple trick; you must either relocate the GFX (which requires code modifications) or make your graphic take less space once compressed. If you want more info on the compression algorithm, it is available in the FF6LE source code or in this pearl script.

A while back, I also covered the same example in a video tutorial in pretty much in the same way. It is a good complement to this written tutorial.

Video