8X8 PROPORTIONAL FONT SECRETS 2012-12-06

Over the past decade and a half, I've been repeatedly hacking Bahamut Lagoon over and over from scratch. I've hacked the entire game about three and a half times now. I've also spent eight years developing an SNES emulator.

During this time, I've learned countless tricks for ROM hacking, and here I'll share my most successful techniques for 8x8 proportional fonts.

BACKGROUND

Traditionally, nobody has attempted 8x8 proportional fonts. There are worries about poor performance, redrawing artifacts, trouble keeping track of allocated tiles, limited VRAM space, etc.

But I feel this is a shame. It's quite possible to work around these issues. In fact, the solutions turn out to be painfully obvious and simple ... once you find them. As simple as what I'm about to describe is, I've never seen nor heard of anyone employing these techniques.

So here you go, the free knowledge of years of research on my part.

SECRET #1: EXPAND THE ROM ... AND RAM!

You would think this would be the most obvious. But nothing is more illogical than some hackers' ardent opposition to expanding the ROM. If you have a 24mbit ROM, expand it to 32mbit. If it's 32mbit already, expand it to 48mbit. And if you are working on a 48mbit ROM, expand it to 64mbit.

Emulators like bsnes and Super Sleuth allow up to 96mbit ROMs easily. If you put out a stellar translation, Snes9X will add support for your game as well. They've done it before with past translations.

But realistically, there aren't any >32mbit games left to translate anyway, and everything supports 48mbit ROMs easily. So expand away! We have 3TB hard drives. Sacrificing the quality of your work to save one millionth of that space is insane.

And why stop there? You can have up to 128KB of SRAM in an SNES game, and most games only use 8KB or 32KB. Hackers will jump through hoops trying to find unused work RAM, only to end up trampling state data that causes bugs later on. Madness! Expand the SRAM, and use it for your scratchpad.

It's okay! Your game will still work on hardware, trust me! Flash carts these days are extremely adaptive and easily run 96mbit ROMs. Even older copiers from the '90s, such as the Game Doctor 3, can do this. And as much as I despise destroying official carts, game modders are increasingly sophisticated these days, as well. Your game will also work on emulators, so just do it!

SECRET #2: STATIC RENDERING

The biggest limitation is speed. Rendering an entire screen full of item names at 2bpp on a 2.68MHz CPU is indeed painful. Combined with limited Vblank time to transfer the tile data, you can find yourself needing up to 2000ms to render a full screen. And during this time, you will see the new items slowly replacing old items as you flip through screens, with significant lag.

The huge mistake is treating 8x8 text as if it is dialogue text, and rendering the tiles dynamically as each item is loaded. On the slow SNES CPU, even the most optimized 8x8 renderer can only produce one item name per 20-30 scanlines. There is no reason to do this.

Take the average item, it is comprised of roughly eight 8x8 tiles, for a total of 64x8. At 2bpp, each 8x8 tile consumes 16 bytes of space. If one were to render the entire item and store it in the ROM, each pre-rendered item name would consume 128 bytes of space, rather than 8 bytes in text form.

This is not that much space! The average game contains about 256 unique items, so you are only consuming about 32KB of data for every item. That is absolutely nothing. Especially now that you're expanding your ROM. You have at least 1MB of space. Dedicate 3% of that to faster text rendering.

And don't stop at items. Pre-render anything 8x8. Yes, you will have to dynamically render user-customizable names. But traditionally there are only one to four of these per screen. It's not a big deal. You might even get fancy and pre-buffer it into SRAM when the game is loaded or the character's name is changed, but it's not required.

When you are requested to write an item into the tile map, simply ensure that you are in Vblank, index into your block of all pre-rendered item names, and then DMA it over. This takes about one scanline per item.

With this, I can render 36 72-pixel wide tiles (324 tiles) in a mere two frames, or 32ms. So fast that the human eye won't even notice.

SECRET #3: REUSING VBLANK

So obviously, if you want your hack to run on hardware, you can only upload tiles to VRAM during Vblank, which lasts a mere ~38 scanlines. The naive approach is to wait for the start of Vblank each time you upload an item into VRAM. That means to render 36 items, you need 36 frames. Wasteful.

Instead, utilize the PPU's vertical counter ($213d): if you are already in Vblank *and* have enough lines to transfer your item, then you don't have to wait at all. Otherwise, wait for the start of the next Vblank. In this way, you can get those 36 items uploaded in a mere two frames.

SECRET #4: YOU HAVE MORE THAN 256 TILES TO WORK WITH

Tile maps for backgrounds reference up to 1024 tiles. For most menu systems, much of VRAM is unused. See if you can write to some VRAM below the 256 tiles you already have. Or try moving the VRAM base address if you have to. Get creative, and you can quadruple the tiles you have to work with.

SECRET #5: DOUBLE BUFFER

Now that you have all of these tiles, put them to good use. A common side effect of a poorly made 8x8 text renderer is seeing garbled tiles on the screen during rendering. As the old tiles are replaced, tiles later on in the screen get over-written, and start showing the wrong text in the wrong areas.

This is easy to avoid. You don't have to waste time flushing the tile map to show blank tiles. Just write your tile data to tile map indices that are not currently being shown on the screen.

By doing this, the entire screen will refresh in one instance. Nice and clean, just like 8x8 monospace text.

SECRET #6: CONSISTENCY IS KING

Don't be tempted to get cute with the widths of your items. "Potion" may fit on three tiles, whereas "Leather Armor" may need seven. So what? Render eight tiles every time.

This keeps your DMA simple, it keeps your tile pool simple, it makes sure you always have enough tiles available no matter how long your item names are, and most importantly ... it creates consistent performance.

Now that we only need two frames, let's keep it always two frames. That way, each time the user moves through a page, it takes the same amount of time. Humans notice change much more than anything else. Some inventory pages taking longer to render than others will be very obvious and jarring. Perhaps counter-intuitively, it's better for everything to take a little bit longer than for uneven speed.

SECRET #7: DON'T KEEP STATE

So now that we have our expanded SRAM, we have a lot of unused variables. You might be tempted to start trying to keep track of which tiles you have rendered. Don't bother.

Instead, use a ring buffer. Pick a unique variable for each text rendering area. You don't even have to initialize it! Say you are drawing 36 items onto the screen. You don't know which item is being drawn at which time. You don't have to keep track of what item# is being drawn, you don't have to keep track of which double-buffered area you are on. Reserve a range for your 36 items, so that's 36(items)*8(tiles)*2(pages)=576 tiles, or 36*2=72 for an item counter.

When it's time to render a tile to the screen, read the counter. If the value is 72 *or greater*, set it to 0 and use that. Otherwise, use the value as you read it for your index. Render your tile data to tile #s index*8+{0-7}. Finally, post-increment the index for the next time the routine is hit.

Since only your text rendering routine will ever be incrementing this variable, it'll always do the right thing.

Reserve separate pools of tiles for each area of the screen. One for the items list, one for the menu options list, etc. Reuse areas only when it's impossible for two lists to be shown on the same screen at the same time.

And if you have a static list that is only rendered once, you don't even need to double buffer it anymore. Just make sure the tile data is uploaded before the tile map is.

SECRET #8: DID I MENTION THAT CONSISTENCY IS KING?

Do not mix and match your fonts. 8x8 proportional fonts are going to want a rather thin font, most likely. Don't use a bold 12x12 font for your dialogue, the contrast is too stark and it will stand out like a sore thumb. Yes, Chicago is a beautiful font in classic games. But don't use it unless you also use a bold 8x8 font. And don't forget that a bold font eats up an extra pixel for every letter. Thin fonts are perfectly legible, even with graphics filters applied. Use them to avoid having to abbreviate things.

IN CLOSING

As I said, the tricks are all simple and obvious. There's no great magic here. But combine all these techniques, and you can render entire pages full of 8x8 proportional text at speeds that are virtually indistinguishable from 8x8 monospace text.

The techniques here are not hypothetical. Even though I don't have anything in release state to show you, I have used all of these. They really do work, and they work fantastically well, I can assure you.

So please, don't abbreviate every item to eight letters or less. And don't try drawing two lines of text per item. And don't attempt to cut the number of items displayed in half — displaying single items that eat the entire width of the user's display. Put in the effort for an 8x8 proportional font, and do it right, and your users will thank you.

Happy hacking!