Bonknuts
Punchy Pedro

Posts: 1934

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc. Quote
« **Reply #50 on:** October 14, 2015, 06:58:06 PM »

I almost always use sprite sheets. I translate/transform the entry from the sprite sheet (referenced by an object) into a SAT format (single entry or meta). For instance, I might have an object that is made up of 3 sprite entries in the SAT. While I have that object array in ram, its not defined in a SAT structure. And whether there's a change to the object or not, that object is always translated into a SAT entry *every* frame as long as it's in that array. The object in the array has attributes like palette number, X/Y position, bounding box, frame number, etc. But those objects still need to be translated into a proper SAT entry or entries. So if the destination is vram instead of local ram, it wouldn't be anymore complex to write to vram instead. At least for my setup.

Report to moderator Logged

http://pcedev.wordpress.com/

elmer
Punchy Pedro

Posts: 2137

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc. Quote
« **Reply #51 on:** October 14, 2015, 08:20:39 PM »

OK, now I'm *really* confused!

It's been a long time since I've actually written a game on a sprite-based machine, so please forgive me if I'm missing something simple.

> **Quote**
>
> While I have that object array in ram, its not defined in a SAT structure. And whether there's a change to the object or not, that object is always translated into a SAT entry *every* frame as long as it's in that array.

That makes perfect sense ... AFAIK that was always the most common way of doing things.

But if you're doing that, then isn't the sort normally part of the translation phase? i.e. you normally just "render" the objects in the order that you want them to appear in the SAT, and write the SAT directly to RAM/VRAM in the correct order.

If you want a part of an object to be behind something else, then you'd just have the object place 2 different "render" calls into the list of meta-sprites to be translated.

Now on the PCE, unlike the SNES & Genesis, you don't need to write the SAT to local RAM because you don't have to wait until hsync/vsync to write to VRAM. (Non-programmers really don't understand just

how incredibly wonderful the PCE's design is.)

I'm having a hard time figuring out when you'd want to sort-and-copy single-or-multiple SAT entries unless you're doing something like uploading a whole bunch of a level's sprite data-and-animated-SAT-entries semi-permanently into VRAM and then compositing each frame's SAT from the previously-uploaded SAT-fragments.

Can you help me understand what usage case that I'm not thinking of here?

« *Last Edit: October 14, 2015, 10:13:50 PM by elmer* »

Report to moderator Logged

touko

Kongo Zilla

Posts: 949

AMIGA integrist

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #52 on:** October 15, 2015, 05:47:03 AM »

> **Quote**
>
> So if the destination is vram instead of local ram, it wouldn't be anymore complex to write to vram instead. At least for my setup.

Exact,writing a sprite attribute to a SAT in RAM or VRAM is the same approach,only destination differs, you write in RAM or through port,code is the same .

This is why, with the possibility to write in VRAM anytime,the use of a RAM buffer for sat is pure useless IMO .
Even better, if you change a sprite attributes you only need to set the good location of your sprite in VRAM, and just write to ports $0002/$0003 consecutively and let the auto-increment do the job .
I use that for my meta-sprite routine, it take 700/800 cycles max for a 4 sprites meta-sprite, and all is in VRAM.

« *Last Edit: October 15, 2015, 06:01:23 AM by touko* »

Report to moderator Logged

Bonknuts

Punchy Pedro

Posts: 1934

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #53 on:** October 15, 2015, 11:32:03 AM »

> **Quote from: elmer on October 14, 2015, 08:20:39 PM**
>
> I'm having a hard time figuring out when you'd want to sort-and-copy single-or-multiple SAT entries unless you're doing something like uploading a whole bunch of a level's sprite data-and-animated-SAT-entries semi-permanently into VRAM and then compositing each frame's SAT from the previously-uploaded SAT-fragments.
>
> Can you help me understand what usage case that I'm not thinking of here?

I can't think of any good examples off hand. Because normally, if all sprites are objects and all objects have to be built as sprites per frame, then you can easily sort objects simply by way of a reference list (single byte array). Do that before the object->sat process and there's no need to sort afterwards.

Maybe an example would be where some objects are only rendered into a SAT entry once and stay in that format, and their attributes are manipulated in a simple way, so updating those changes in vram isn't so bad (maybe only X or Y position, or cell #.. something like that). For example if an object is always a 32x64 sprite and all that changes is cell#/X/Y, then it can easily be kept in SAT format. If priority issues need to be evaluated by other objects that are constantly being rebuilt, the DMA list sort would probably be the faster/better option.

I've mixed and matched stuff like this before; debris, bullets, clipping/overlay.. stuff that really only needs X/Y or basic stuff updated in SAT format. Basically because object->sat process (lots of indirect and redirect of there are quite a but of frames and phases for an object) can eat up a decent amount of cpu cycles. I like sprite sheets (frame tables) because they are so easy to design animation "cells" for objects. It's neat, clean and organized, but the down side is processing time. Cheat where you can.

Touko: you can eliminate an extra frame delay by setting up the VDC to do two frames inside a VCE frame. It's tricky but it's doable. I've set it up so that the start of the display does SATB DMA instead of at v-int. That would give you your vblank time for DMAing and the start of the display SATB DMA for syncing the update for the frame to be shown. Normally it's not a problem, but this DMA list thing does make that an issue on a stock frame setup.

Report to moderator Logged

http://pcedev.wordpress.com/

touko
Kongo Zilla

Posts: 949

AMIGA integrist

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #54 on:** October 16, 2015, 01:34:26 AM »

> **Quote**
>
> you can eliminate an extra frame delay by setting up the VDC to do two frames inside a VCE frame. It's tricky but it's doable

Why do you want have an extra frame delay ??

If you desable the auto SATB DMA, and do it manualy after your DMA list is complete, you shouldn't have this delay .

Report to moderator Logged

Bonknuts
Punchy Pedro

Posts: 1934

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #55 on:** October 16, 2015, 11:16:12 AM »

When I tested doing manual SATB DMA, during vblank, it didn't happen until the next v-int. Guess I'll have to revisit/retest that.

Report to moderator Logged

http://pcedev.wordpress.com/

touko
Kongo Zilla

Posts: 949

AMIGA integrist

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #56 on:** October 16, 2015, 01:14:29 PM »

> **Quote from: Bonknuts on October 16, 2015, 11:16:12 AM**
>
> When I tested doing manual SATB DMA, during vblank, it didn't happen until the next v-int. Guess I'll have to revisit/retest that.

Aaaah, it's possible then ..
I always thought it was like other DMA, you have the entire VBLANK for that, if not finished, it resume next vblank ..

« *Last Edit: October 17, 2015, 12:27:08 AM by touko* »

Report to moderator Logged

elmer
Punchy Pedro

Posts: 2137

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #57 on:** October 16, 2015, 02:14:09 PM »

> **Quote from: touko on October 16, 2015, 01:14:29 PM**
>
> I always thought it was like other DMA, you have the entire VBLANK for that, if not finished, it continu next vblank ..

The VDC manual says ...

```
For VRAM-SATB block transfers, 256 words are transferred at the beginning of a vertical
blanking period.
It is triggered by access to the high byte of the VRAM-SATB block transfer source
address register (DVSSR).
If the register is set, a block transfer operation will start at the beginning of the
following vertical blanking period.
```

Charles MacDonald's pcetech.txt also makes it sound like it's an only-at-the-start-of-vblank trigger.

Hahaha ... am I going to get the chance to use my Inigo Montoya quote again? 

Report to moderator Logged

touko
Kongo Zilla

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #58 on:** October 17, 2015, 12:25:13 AM »

Posts: 949

AMIGA integrist

Yes i know that, but in my mind it was for auto DMA ..

Damn ..

Quote

> Hahaha ... am I going to get the chance to use my Inigo Montoya quote again?

Report to moderator Logged

Bonknuts

Punchy Pedro

Posts: 1934

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

« **Reply #59 on:** October 19, 2015, 01:20:33 PM »

Quote

Ok, so I've been crunching numbers all weekend in relation to dynamic tiles. Basically, I was watching some snes game longplays and tried to come up with ways to replicate some of the same effects.

I had this problem, where I wanted to do *large* area patterns in a faux BG layer - but I didn't want to update the tilemap at all. I only wanted to up the tile dynamic buffer. This presented a challenge, because all I had were 8 horizontally shifted frames. In other words, I would need a *complete* rotation in order for this to work (not just an 8 pixel rotation). No only is this problematic, but storing the these frames in ST1/ST2 opcodes immediately doubles that size. For a large half screen pattern - that's just not doable.

The first solution I had, was to allow a wider buffer (horizontally) than the intended target. Since I use ST1/ST2 opcodes to draw a bitmap line into VDC ram, with an RTS at the end, I'm limited to the width of that stored *bitmap* line (see below). I figured if I could start in the middle of the line, and then restart the line again. The craziness comes in here; I would use the VDC interrupt as a timer for the cpu. Basically, put the cpu on a timer leash and when the that timer runs out - reposition the PC back to the parent routine. This keeps the cpu from writing too much on the second call of the same line (to create the completed the rotation). Not only is this overly complex, it also exotically dangerous. Kinda.

So a better solution, is to break down the large pattern of bitmap lines into segments of smaller horizontal sections (and lines as well). As in, there would be multiple breaks in a single bitmap line (as RTSs). And call each line in sequence. The vram pointer doesn't need to be re-adjusted because it's still a sequential sequence in relation to vram. For example, say I have a pattern that is 15 tiles wide (120px). If I broke that down into 5 segments, it would be 24px line segments. The buffer would only need to be enough for (n+1)*segments wide for overflow handling. So 120+24= 144px wide buffer in vram. This allows you to start in one of three positions of a segment; 0, 1, 2. So for a full horizontal rotation of a 15 tile (120px) wide pattern, you need the segment offset, and offset inside the segment, and the frame rotation number.

This gives me a little bit of overhead; one JSR/RTS set per line segment, but the approach is much cleaner and less wasteful for vram. I still have a smaller buffer for over-run, but not nearly as big.

So to give some ideas for clarification here, I'll explain a few details.

1) To draw a bitmap line into vram, you need to set the autoincrement vram pointer to 32+. Since only a WORD can be written into vram at a time (two 8pixel planes), you'll have to do a second pass to write the full 4bit color image (assuming 4bit color is the goal). 32+ mean increment by 32 WORDS, so you'll need to organize the tiles (interleave). Here is where another optimization comes in; you can draw a bitmap line into a buffer of tiles @ (n*8)+offset. So if you start at line 0 of the buffer and keep writing passed it (with autoincrement of 32+), you'll end up writing line into the next line in the buffer at line 8, and then line 16, etc - until you reach the end. You'll need to reposition the offset in vram to point to line 1, and that'll draw 1, 9, 17, etc. So you save cycles by not having to constantly set VRAM pointer every scanline. You only need to do this 8 times. But that's just for the first 2bit planes. You need to do this again with the proper offset into the second 2bit planes of the tile. So data needs to be organized in a very specific way in rom, with embedded opcodes, but the result is very optimal in terms of speed.

2) If the amount of data is quite large to write to vram, you'll probably be racing the display because there won't be enough time in vblank for really large patterns. This is ok, though. There are 455 cpu cycles per VCE scanline. As long as you're under that, you'll be fine... kinda. If you followed along with the above method, you'll see that it requires multiple passes, and then a second pass for the second bitplane. Even if your data being written to vram, in the form of a scanline, is faster than the beam - the order of the written data presents the problem. One solution is to split the bitmap buffer in vram into two halves; and upper bitmap and lower bitmap. And start drawing this process earlier in vblank, or put a status bar at the top of the screen to effectively increase screen drawing time. Maybe two halves isn't enough. Maybe more are needed. Or maybe just do 16-24 lines full color, and then switch

methods. The idea here, is to get enough buffered room between the beam and where you are writing in that bitmap position. Remember, if your data is stored as scanlines, you have a very flexible method and control in how you write to that vram bitmap.

The idea here, is to do a large pattern of dynamic tiles at 60fps and no double buffering. Of course double buffer will work, assuming it'll fit on vblank vram-vram time frame, but with a large buffer already that means eating into your vram space.

The whole idea of doing dynamic tiles as scanlines, means you no longer have limitations of tiles. You can do sine wave effects or line scrolls because you have direct control over the X position of that line being transferred into vram. You can Y-scale and do vertical effects as well. You can also draw parts of an image in reverse order (vertical flipping/mirroring, etc). And maybe the best of all, you can do transparency effects on that pseudo layer. Remember, you have to write the image as two 2bit images. Since the hardware puts them together, you have hardware assisted compositing. Sure, the colors are low (4 colors for one plane, and 3 colors of transparency to overlay) - but you also have the aid of palettes to assign to any 8x8 area. If you're ambitious, you could do the tilemap reposition trick to get 8x4 or 8x2 palette association.

This is just one approach. I have other dynamic tile approaches with different abilities. Some of them that allow object drawing into vram, overlapping edges on tiles without using sprites, etc. But there's not enough time to do that in 60fps, so those effects would be 30fps. Hopefully I can demonstrate these effects in a demo of sorts (playable with a simple engine).

Just to note: I'm using the bitmap line approach because it offers way more control over the pattern/pseudo layer than the tile write approach. I could have easily arranged a column of tiles in vram to show as a row format in the tilemap (which is a super easy setup/approach), but because of how the tiles are composed of two 2bit tiles, doing independent vertical scrolling on this fake BG layer becomes *much* more complex. A bitmap line approach easily allows for independent vertical and horizontal scrolling of the fake BG layer, but also allows hsync style effects (horizontal), vertical effects, as well transparency or split layer effects.

Report to moderator Logged

http://pcedev.wordpress.com/

elmer

Punchy Pedro

Posts: 2137

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc. Quote

« **Reply #60 on:** October 20, 2015, 10:09:54 AM »

It's taken me a couple of cups of coffee to get my old head around what you're describing.

Very Cool!

It's fun to figure out how to achieve some nice effects.

I'll be interested to see the demo if you decide to put one together.

Isn't this going to have a pretty high CPU & memory cost for any large pattern?

Do you envision using this in a game level, or is it more of a title-screen/demo-mode effect?

Could you see this technique helping you with a PCE version of Sonic?

Report to moderator Logged

touko

Kongo Zilla

Posts: 949

AMIGA integrist

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc. Quote

« **Reply #61 on:** October 20, 2015, 10:27:18 AM »

I'am also interested of how many CPU+RAM this effect could cost .

Report to moderator Logged

Bonknuts

Punchy Pedro

Posts: 1934

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #62 on:** October 21, 2015, 11:18:23 PM »

All the approaches I've been looking at, are all for in game stuffs. Some might be on the extreme side - but still allowable for some type of game engine. I had the above line style one clocking in about ~55% cpu resource with a 120x192 dynamic tile window.

By demos, I mean playable. I have shmup engine that I'll use for a few examples (forced scrolling always lends itself to nice optimizations). I need to make a platform engine to demo on.

Report to moderator Logged

http://pcedev.wordpress.com/

touko

Kongo Zilla

Posts: 949

AMIGA integrist

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #63 on:** October 22, 2015, 01:32:24 AM »

55% is huge,and not bad at same time !!
There are only 45% remaining for all the rest, it's short .
It's doable for a plateformer i thing, or for non intensive datas transfert games .
If you can keep most of datas in VRAM, 45% should be enought .

« *Last Edit: October 22, 2015, 01:34:47 AM by touko* »

Report to moderator Logged

Bonknuts

Punchy Pedro

Posts: 1934

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #64 on:** October 31, 2015, 02:04:26 PM »

So this is just theoretical, but there's a possible way to save some cycles when doing modifications to stuff stored in vram.

So say you need to modify just the LSB or MSB of data in vram. Normally, you set the read and write positions to be the same, and do a redundant read/write in order for the process to be updated. I.e. dealing with modifying only a byte, but you have to write a word for the pointer to be updated.

So here's something that might help out:
Normally, you would do something like LDA $0002, sta $0002 and then write new data to $0003.

That read/write $0002 is 12 cycles (+1 penalty for each access to the port). So, is there an instruction that could read and write back to the same address without modifying the data? Yes. TRB or TSB are read-modify-write instructions. As long as Acc is zero, nothing will be modified. The instruction is 7 cycles for Absolute addressing. So, given that there should be a +1 and another +1 for the read and write back, total cycle count should be 9 cycles.

With Acc as zero, use either X or Y to update the data for $0003. Both X and Y are flexible enough for reading in data from an array; LDX array,y or LDY array,x. And both can write to the vdc port as well.

« *Last Edit: October 31, 2015, 02:06:12 PM by Bonknuts* »

Report to moderator Logged

http://pcedev.wordpress.com/

touko

Kongo Zilla

Posts: 949

AMIGA integrist

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #65 on:** October 31, 2015, 03:56:51 PM »

Eh like you said before, tom you have a winner here ;-)
Very interesting trick.

Report to moderator Logged

Bonknuts

Punchy Pedro

Posts: 1934

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #66 on:** October 31, 2015, 06:24:14 PM »

I was specifically thinking tilemap and tile stuffs (dynamic tiles), etc - but there's something else that's interesting you can do with this.

Say you have a sprite cell that you want to dynamically clip holes into it to show parts of the background onto if it. I.e. creating a fake priority layer to certain pixel in the BG @ position x,y on the screen. Normally, you would do this in local memory and AND a mask against all planes. AND'ing all planes creates holes in the sprite cell.

Ok, so you have a "clean" sprite cell in vram. The object (no pun intended) is to write to a new cell in vram - the updated "look" of the cell. This bypasses having to transfer from main memory to vram after the ANDing process.

Something like
lda table.lo,x
trb $0002
lda table.hi,x
trb $0003
inx

VS
lda $0002
and table.lo,x
sta $0002
lda $0003
and table.hi,x
sta $0003
inx

Both approaches keep the source and the destination cells in vram, but the first method is 30 cycles VS 36 cycles of the second one. Of course you could optimize it a bit more with some unrolling, but the difference will always come from the TRB vs lda/sta.

Something I also thought about, but don't have an idea of what to use it for... is a TRB on $0002 followed by a TSB on $0002. I wonder if it would read back what's in the buffer (the modified byte). Probably not. But I should test just in case.

« *Last Edit: November 01, 2015, 12:52:14 PM by Bonknuts* »

Report to moderator Logged

http://pcedev.wordpress.com/

touko

Kongo Zilla

Posts: 949

AMIGA integrist

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #67 on:** November 01, 2015, 06:49:19 AM »

Not bad at all,but actually is less than 30 cycles(in fact 28 in the first method)and you must count the setting of vram read pointer,it's negligible when you have to do it multiple times .

Code: [Select]

```
lda table.lo,x        ; 5 cycles
trb $0002            ; 7 cycles + 1
lda table.hi,x        ; 5 cycles
trb $0003            ; 7 cycles + 1
inx                      ; 2 cycles

total =                28 cycles
```

« *Last Edit: November 01, 2015, 06:54:34 AM by touko* »

Report to moderator Logged

Bonknuts

Punchy Pedro

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #68 on:** November 01, 2015, 10:19:28 AM »

There could be in fact no penalty cycles for TRB. It's unknown because the extra penalty cycle for LDA, STA,Txx, and ST0/ST1/ST2 is done inside the CPU (anything accessing $1fe000-1fe03ff) and not a wait state thing from the VDC. It's possible TRB/TSB doesn't have it, or has +1 overall, or has +1 for the read and +1 for the write. I would have to test it.

Posts: 1934

It's weird how for example ST1 is listed 4 cycles, but since it writes to $1fe000-1fe03ff range, it has the internal +1 penalty. As far as I know, there's no difference between the 6280 and 6280a for this penalty area.

« *Last Edit: November 01, 2015, 10:21:39 AM by Bonknuts* »

Report to moderator Logged

http://pcedev.wordpress.com/

touko

Kongo Zilla

Posts: 949

AMIGA integrist

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #69 on:** November 01, 2015, 11:15:22 AM »

> **Quote**
>
> It's weird how for example ST1 is listed 4 cycles

Yes even in official doc is listed as 4 cycles .
i don't understand japanese, but is doesn't seem that there is any penalties when accessing VRAM, unleast nothing in official doc seems to suggest that .

Are you sure that this kind of penalities are really here ??

« *Last Edit: November 01, 2015, 11:17:11 AM by touko* »

Report to moderator Logged

elmer

Punchy Pedro

Posts: 2137

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #70 on:** November 01, 2015, 12:04:42 PM »

> **Quote from: touko on October 31, 2015, 03:56:51 PM**
>
> Eh like you said before, tom you have a winner here ;-)
> Very interesting trick.

Yep, Xanadu 2 is using the TSB/TRB trick for writing the font data ... together with self-modifying code to switch between TSB and TRB opcodes in order to set the color.

I've not looked at Xanadu 1's font code yet.

Report to moderator Logged

Bonknuts

Punchy Pedro

Posts: 1934

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #71 on:** November 01, 2015, 12:53:13 PM »

> **Quote from: elmer on November 01, 2015, 12:04:42 PM**
>
> Yep, Xanadu 2 is using the TSB/TRB trick for writing the font data ... together with self-modifying code to switch between TSB and TRB opcodes in order to set the color.

That's awesome!

Report to moderator Logged

http://pcedev.wordpress.com/

touko

Kongo Zilla

Posts: 949

AMIGA integrist

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #72 on:** November 01, 2015, 01:01:00 PM »

> **Quote**
>
> Yep, Xanadu 2 is using the TSB/TRB trick for writing the font data ... together with self-modifying code to switch between TSB and TRB opcodes in order to set the color.

Thanks, very interesting to know .

Report to moderator Logged

elmer

Punchy Pedro

Posts: 2137

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #73 on:** November 01, 2015, 01:58:46 PM »

> **Quote from: Bonknuts on November 01, 2015, 12:53:13 PM**
>
> > **Quote from: elmer on November 01, 2015, 12:04:42 PM**
> >
> > Yep, Xanadu 2 is using the TSB/TRB trick for writing the font data ... together with self-modifying code to switch between TSB and TRB opcodes in order to set the color.
>
> That's awesome!

In the case of Xanadu 2 it's more of a way to be "clever" than "fast".

They're drawing a 12x12 glyph as 3 separate passes of 4x12 pre-shifted strips ... pretty slow and ugly with a *lot* of changing of VRAM pointers.

Even the Xanadu 2 8x12 glyphs (that don't exist in Xanadu 1) are still drawn in 3 passes.

If you look at Xanadu 1's text on-screen ... the way that the glyph outlines overlap really suggests that they're doing the same.

Because I've not looked at it, yet, I'm not sure if they are dynamically-generating the thick-and-outlined glyphs from the ROM font data, or if they are stored in the game data.

Either way ... since they're using 4x12 strips, then I can probably hack the font system to do bi-width glyphs in the same way that I did for Team Innocent (that's plan A).

Report to moderator Logged

Bonknuts

Punchy Pedro

Posts: 1934

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #74 on:** November 09, 2015, 12:23:29 PM »

Audio!

Report to moderator Logged

http://pcedev.wordpress.com/

Bonknuts

Punchy Pedro

Posts: 1934

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #75 on:** November 09, 2015, 12:41:07 PM »

Hmm. So I have a direct channel 4 xm software player @ 29% cpu resource or 35% if you use the last two channels to stream fixed PCM stuffs. 4 XM channels with volume and looping support, 2 fixed frequency channels with volume control, all 6 are **stereo** @ **5bit** PCM with a rate of 7khz... 35% cpu. No regular PCE channels if 2 auxiliary PCM channels used. 6 channels total.

So here goes this: 8 PCM channels: 4 XM style with volume and looping support, 2 fixed frequency with volume control, and 2 fixed frequency at full volume. All played back at 7khz @ **6bit** PCM **mono**... 31% cpu. 4 PCE **stereo** channels still left over for full use. 12 channels total.

The drawbacks of the second engine: mono PCM and the XM driver is ranges from 0%-100% of the 7khz. The first engine can play back up to 8 times the 7khz driver by skipping samples. The second one would require a couple of re-octave'd of the same instruments. Though that isn't too terrible if you consider the average instrument sample is somewhere between 1 to 2 seconds long. At 7khz, if that means having three octaves for one sample at 2 seconds long - that's 14k*3= 42k. All the XM channels on both players support looping, so an instrument can be longer than the sample itself.

So, I have the first engine complete. I'm just picking a quick MOD song that doesn't do fancy stuffs to playback on it (because I really don't want to write a full music engine).

And to back up my claims, I need to finish the tables for the second engine. And some sort of quick

demo to show it off as well. Maybe a song comprised entirely of 8 channels of glorious pan flutes.

I feel like audio, or rather PCM, is the theme of November...

« *Last Edit: November 09, 2015, 01:42:53 PM by Bonknuts* »

Report to moderator Logged

http://pcedev.wordpress.com/

touko

Kongo Zilla

Posts: 949

AMIGA integrist

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #76 on:** November 10, 2015, 01:45:57 AM »

Wahou, that's the proof that the PCE audio is really good and flexible .
Of course your second engine is the more impressive .

Good works .

Report to moderator Logged

Bonknuts

Punchy Pedro

Posts: 1934

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #77 on:** November 10, 2015, 10:53:43 AM »

Touko, you were saying that you keep your samples bit-packed? Have you looked into RLE? As in, $8n would be repeat last sample n times. Not RLE the bitpacked bytes, but the 5bit samples in 8bit format.

It *really* depends on the sample itself, but when values get compressed in range down to 5bit, you tend to runs of samples. I just noticed quite a few samples I was converting for the XM players, easily fit into this. One sample that I was ripping from a mod file, was actually double sampled. If I hadn't looked, I would have wasted double the space in rom for it. Now I'm going to add in analysis to my conversion tool to look for this. As well as wave forms that might benefit from *halving* the frequency and see what the rate of error is - if it's barely noticeable, the it's worth the savings. You could even do mixed mode samples (parts where it's at half frequency and you just repeat every sample twice for that section).

Report to moderator Logged

http://pcedev.wordpress.com/

touko

Kongo Zilla

Posts: 949

AMIGA integrist

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #78 on:** November 11, 2015, 03:36:50 AM »

i use this simple technique .

1 sample in the first 5 bit of the 1st byte
1 sample in the first 5 bit of the 2nd byte
and the 3th packed in the 3 last bits of the first byte, and the 2 last ones in the 2nd byte, in fact only the 3th sample needs to be shifted .
Of course you lose 1 bit, but your sample is reduced by 33% ..

Quote

> to summarise
>
> 11111333 2222233X
>
> 1 bits of the 1st sample
> 2 bits of the 2nd sample
> 3 bits of the 3th sample

it's very fast .

« *Last Edit: November 11, 2015, 03:47:39 AM by touko* »

Report to moderator Logged

Bonknuts

Punchy Pedro

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #79 on:** November 25, 2015, 10:18:02 PM »

Posts: 1934

I wrote up a small research blog thingy about how one could go about doing a Wolfenstein 3D style game on the PC-Engine, with decent results.

Report to moderator Logged

http://pcedev.wordpress.com/

Dicer

Punchy Pedro

Posts: 1829

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #80 on:** November 25, 2015, 11:10:40 PM »

> **Quote from: Bonknuts on November 25, 2015, 10:18:02 PM**
>
> I wrote up a small research blog thingy about how one could go about doing a Wolfenstein 3D style game on the PC-Engine, with decent results.

Love to see something, even if just a proof of concept....

and better than FACEBALL

Report to moderator Logged

Bonknuts

Punchy Pedro

Posts: 1934

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #81 on:** November 26, 2015, 11:06:16 AM »

Faceball is generally unoptimized. I know that it has to contend with planar graphics, but it doesn't even use the sprite planar format. I mean, the solid color walls are symmetrical about the Y axis. They could have simply rendered half the screen and used the SAT to show it flipped/mirrored for the lower half. Instant speed up in rendering. The largest viewable window area is only 128 real pixels (64 double internal rendered pixels). They could used a second layer of sprites on top of that for rendering the objects into that window.

I have no idea about how optimized their internal rendering engine is, but mirror trick alone would have speed it up regardless (on the sheer issue of pixel fill rate).

A closer inspection: for two window mode each window is 128x64 real pixels. Since this is kept in local ram, it takes 49k cpu cycles to clear each window for a new renderer using the Txx instruction. That's almost a whole frame, 82%, just to clear the buffer. But from what I've looked at, the game uses a very slow ORA method to clear it (read, modify, write-back).

Here's part of the buffer clear code:

Code: [Select]

```
.loop
txa
ora [$5D],y
sta [$5D],y
iny
iny
cpy #$10
bcc .loop
```

The value in Acc is always $ff. That's 26 cycles a byte for 8 bytes (it's skipping the second plane in the composite tile). I've seen a full playthrough of the game and no where have I've seen an OR pattern other than #$ff. There's also an AND routine that's built the same way. The routines that draw pixels use a similar routine, but the compare is a memory/variable and not a constant because they actually draw variable length runs of vertical of pixels. So right off the bat you're looking at a difference of 425,984 cycles vs 98,304 cycles.

The game's rendering engine is unoptimized in both high level design and lower level design.

Report to moderator Logged

http://pcedev.wordpress.com/

Black Tiger

Punchy Pedro

Posts: 11142

DEBDE DEBDA

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc. Quote

« **Reply #82 on:** November 26, 2015, 12:07:58 PM »

Faceball may not be efficient, but the 4-player splitscreen is noteworthy for console games.

ABlackFalcon made a big deal of the fact that Mario Kart 64 invented the now standard 4-player splitscreen for 3D games. If it would have been revolutionary as a N64 game if it was actually true, then it should be all the more mindblowing for a 16-bit game.

Report to moderator Logged

http://pcengine.proboards.com

Active and drama free PC Engine forum

spenoza

Punchy Pedro

Posts: 2775

Kyokugenryu Master

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc. Quote

« **Reply #83 on:** November 28, 2015, 10:46:35 AM »

What I really really want to see is a demo of this 2-engine sound hybrid beast. Love to witness the PCE putting out that many channels of sound.

Report to moderator Logged

My meager PC Engine Collection so far.
PC Engine Software Bible Forum
Racketboy Forum

Bonknuts

Punchy Pedro

Posts: 1934

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc. Quote

« **Reply #84 on:** December 01, 2015, 09:37:28 AM »

spenoza:

The sound engine demos? They'll be out soon. The first engine(well, driver) is completely done (the 4XM channels, or MOD thingy). Six octave ranges (which is way overkill), all 12 notes per octave, 32 steps in between notes. Frequency sliding works perfectly, as does finetune. Looping works. EOF markers work. My wave conversion tool to make the special waveform format is done (does 5bit and special 2's compliment 6bit and 7bit output formats). All the interfacing for the driver is done, and it's buffered too so that everything is synced (and also so that it updates during a "safe" window of time): the interrupt driver needs full control of all the PCE audio regs, so any other app is subordinate and needs to be buffered for a windowed update.

I also did some tests. The nyquist theorem says you should sample at two times the frequency. Frequency scaling is technically re-sampling. At two times the frequency output, the artifacts are surprisingly low. At three times the output, they are still surprisingly decent or bearable . Anything above that, depending on the sample, and the artifacts become predominant. If you listen to this sample, https://www.youtube.com/watch?v=jRl-A8uTkxE, the horns sound gritty. That's the artifacts that I'm talking about. Of course, samples in that video haven't been preprocessed. But to be honest, those "horns" are less gritty than the ones in Bloody Wolf. I had this idea of combining sample synthesis with PCE normal channels for a paired sound.

Another example of the artifacts from resampling too far above the driver output is https://youtu.be/m6_HvykkFKI?t=3m14s at 3:14. The main instrument sounds somewhat .. screech-y and distorted.

Not every sample instrument is going to sound great on a 7khz 5bit output driver, but some sound exceptionally decent. Some will need to be resampled by an external app with proper filtering to remove the artifacts, and if the octave range usage is wide for that instrument then it might need to be resampled into 2 octave ranges (two samples). Maximizing the sample amplitude with a preprocessing app is also important, even if that means some clipping. Resolution noise (static/hiss/etc) is very much less perceivable for the human ear to detect when things are loud. The closer the sample gets to the edges of the amplitude limits, the more values or steps it has access to to represent that waveform. This is especially true for quieter parts of a sample - the more steps you can throw at it, the better it will sound. Hardware volume can be re-adjusted to compensate for the louder sample, without losing resolution depth. At 5bit resolution depth, you take whatever you can get out of a sample.

This isn't my engine, but this is a MOD player written years ago for the PCE (never made public). It

gives an idea of what instruments sound good and what sounds gritty - for 7khz 5bit output. Though keep in mind *none* of these samples have been *preprocessed* to help smooth out frequency artifacts or bitdepth issues.

https://www.youtube.com/watch?v=bLhFbwS3tNc

https://www.youtube.com/watch?v=UvwQ0q6Xr54

https://www.youtube.com/watch?v=m6_HvykkFKI

https://www.youtube.com/watch?v=jRl-A8uTkxE

Note: These are played on an overclocked PCE (emulator) because the original frequency scaling code was so slow that it would often end up skipping samples and had some extreme jitter. I overclocked it to give an idea of a more solid sound of frequency scaling that can be achieved; the cpu is overclocked but the output is still 7khz 5bit samples. It's one of the reasons why I'm not sharing the source to this particular player (that and I never heard back on permission to do so).

But anyway, this should give a rough, or decent, idea of the *first engine's* capability; 7khz, 5bit waveforms, hardware volume, 4 channels of frequency scaling.

Kind of off/on topic, but it's *kinda* rare for mods and similar files (XM, IT) to emulate timbre bending of instruments through multiple samples. I experimented with this BITD with Fast Tracker 2, and it works pretty decent. If you keep the sample short, with a loop point, you have more room in rom/ram/whatever for multiple versions of that sample with different preprocessed changes over time - a set of samples, with each sample loop representing a specific change in time. In the driver, you can easily switch to which sample of the set is being played back - even in the middle of it (though I would do that on a frame basis or 1/60 sec). Basically wavetable synthesis (which is not *sample based* synthesis used in MODs, XMs, or the SNES and Amiga). Of course it wouldn't be on the level of a PPG synth, but the flexibility of building and controlling sounds instead of just playing back a sample at different frequencies - is pretty damn cool IMO. This is where a 15khz soft mixed engine would dominate: 2 channels capable of both wavetable and sample based synth (6 or 7bit PCM), and the rest of the soft mix channels fixed frequency. And you still have 4 regular PCE channels to give it a mix of that distinct PCE sound.

Anyway, the first engine is done. I just need to whip up a small music engine to show it off. Then finish the second engine, which is much more exciting.

Report to moderator Logged

http://pcedev.wordpress.com/

Bonknuts

Punchy Pedro

Posts: 1934

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc. Quote

« **Reply #85 on:** December 01, 2015, 08:53:07 PM »

FYI, you can use y/sin(a) instead of sqrt[x^2+y^2]. Not only is the table much smaller, but the table access is much faster (the addition of two squares is expensive for just building the index into the sqrt table). In trig, and in calculus, you always tend to resort to finding the root of r^2 just because it's easy and the calculator is fast. I remember almost all my stuff from trig and at least remember the basic stuff immediately off hand. But it's those people, with the right set of eyes, that see the connection and the easier way around. Those are the brilliant computer science math geeks.

Anyway, back on topic...

Report to moderator Logged

http://pcedev.wordpress.com/

elmer

Punchy Pedro

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc. Quote

« **Reply #86 on:** December 02, 2015, 10:41:31 AM »

Quote from: Bonknuts on December 01, 2015, 08:53:07 PM

> Anyway, back on topic...

Can I stay on the math side for a moment?

Posts: 2137

Back-in-the-days of limited hardware (like the PCE) we just used an approximation when we needed the distance (i.e. sqrt[x^2+y^2]).

It was fast, and "accurate-enough" for most tasks.

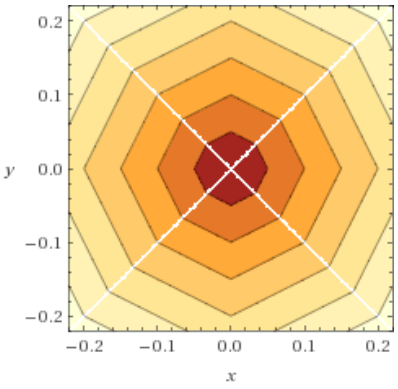
The classic one was ...

```
dx = abs(x1 - x0)
dy = abs(y1 - y0)

if (dx > dy)
  dist = dx + 1/2 * dy
else
  dist = dy + 1/2 * dx
```

On the PCE, that's a couple of compares and branches, a bit-shift, and an add. Nice and fast.

Here a nice plot of the function (stolen from another site just to add a pretty pic) ...



On the PC-FX, where you've got a fast integer multiply, you can improve the function with ...

dist = 1007/1024 * dx + 441/1024 * dy

I just dug up these modern references to the same old trick ...

http://www.flipcode.com/archives/Fast_Approximate_Distance_Functions.shtml

http://gamedev.stackexchange.com/questions/69241/how-to-optimize-the-distance-function

Report to moderator Logged

Bonknuts

Punchy Pedro

Posts: 1934

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #87 on:** December 03, 2015, 07:10:21 AM »

Hey, that's pretty decent! I've never seen that one before.

Report to moderator Logged

http://pcedev.wordpress.com/

touko

Kongo Zilla

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #88 on:** December 31, 2015, 11:32:54 AM »

> **Quote**
>
> Yep, Xanadu 2 is using the TSB/TRB trick for writing the font data ... together with self-modifying code to switch between TSB and TRB opcodes in order to set the color.

Posts: 949

AMIGA integrist

I Thought about it, like i have a routine which use sprites for some hud informations like scrore/hi-score etc..,with 8 pixel's tiles i use 2 indexed arrays (i write 2 number in one 16 pixel's sprite) .
i thing storing tiles data in VRAM and use TSB with acc=0 should be a faster VRAM to VRAM copy than 2 arrays in RAM ..

Report to moderator Logged

Bonknuts

Punchy Pedro

Posts: 1934

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #89 on:** January 03, 2016, 10:19:01 AM »

Just beware of TRB/TSB on the $0003 takes longer than normal. Even a LDA $0003/STA $0003 takes longer than normal. TRB/TSB $0003 takes ~ 15.6 cycles on the real console. TRB/TSB $0002 takes ~ 8.9 cycles on the real console.

I did some VDMA tests and can confirm that it's roughly 81 WORDs per line in 5.37mhz mode. So definitely ~324bytes per line in 10.74mhz mode. I was able to transfer 17.6k with a clipped 209 line display @ 10.74mhz. That's more than perfect for my other bitmap mode display. A few other things too.

Report to moderator Logged

http://pcedev.wordpress.com/

touko

Kongo Zilla

Posts: 949

AMIGA integrist

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #90 on:** January 03, 2016, 12:29:51 PM »

> **Quote**
>
> Just beware of TRB/TSB on the $0003 takes longer than normal. Even a LDA $0003/STA $0003 takes longer than normal. TRB/TSB $0003 takes ~ 15.6 cycles on the real console. TRB/TSB $0002 takes ~ 8.9 cycles on the real console.

8.9 cycles is faster than LDA/STA, but the overhead for writing to $0003 is quite annoying .

> **Quote**
>
> I did some VDMA tests and can confirm that it's roughly 81 WORDs per line in 5.37mhz mode. So definitely ~324bytes per line in 10.74mhz mode. I was able to transfer 17.6k with a clipped 209 line display @ 10.74mhz. That's more than perfect for my other bitmap mode display. A few other things too.

Good news, your tests confirm what aladar said about VDMA .

Report to moderator Logged

Bonknuts

Punchy Pedro

Posts: 1934

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #91 on:** January 03, 2016, 03:26:49 PM »

Yeah, the R-M-W on $0003 is disappointing. I only tested this during active display, though. Could be different for vblank.

I guess you could do trb $0002; ldx ABS,y ; stx $0003. Or trb $0002; st2 #nn.

Report to moderator Logged

http://pcedev.wordpress.com/

Bonknuts

Punchy Pedro

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #92 on:** January 13, 2016, 05:39:50 PM »

elmer: You were right about the devil in the details (sprite faking BG layer). I was working on making a presentation/demo, but realized that I'm going to need to make a custom map utility for this.

Report to moderator Logged

Posts: 1934

http://pcedev.wordpress.com/

elmer

Punchy Pedro

Posts: 2137

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #93 on:** January 14, 2016, 09:54:15 AM »

Quote from: Bonknuts on January 13, 2016, 05:39:50 PM

> elmer: You were right about the devil in the details (sprite faking BG layer). I was working on making a presentation/demo, but realized that I'm going to need to make a custom map utility for this.

Such are the "joys" of pushing technology forward into new areas!

Report to moderator Logged

elmer

Punchy Pedro

Posts: 2137

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #94 on:** January 17, 2016, 04:16:14 PM »

After the recent discussion on the PCE's CD booting, and my continuing discussion with TheOldMan, I thought that I'd document something that I've not seen mentioned before.

Anyone that has looked at making a PCE CD has already discovered the "IPL Information Data Block" that must be put in the 2nd sector on the CD, and who's data values tell Hudson's IPL (the code that must be put in the 1st sector on the CD) exactly where to load your game program and what address to call to run it.

Now while you can just let PCEAS handle creating a CD image for you ... you can also just do it yourself if you want complete control over the CD layout.

As a reminder, here is the "IPL Information Data Block" in PCEAS format, together with the PCEAS/HuC default values ...

```
org     $3000

; IPL INFORMATION DATA BLOCK

db      $00                     ; $00 IPLBLK_H (ISO Sector 2)
db      $00                     ; $01 IPLBLK_M
db      $02                     ; $02 IPLBLK_L
db      $10                     ; $03 IPLBLN   (Size $8000)
db      $00                     ; $04 IPLSTA_L (Load $4000)
db      $40                     ; $05 IPLSTA_H
db      $70                     ; $06 IPLJMP_L (Exec $4070)
db      $40                     ; $07 IPLJMP_H

db      $00                     ; $08 IPLMPR2  (bank $80)
db      $01                     ; $09 IPLMPR3  (bank $81)
db      $02                     ; $0A IPLMPR4  (bank $82)
db      $03                     ; $0B IPLMPR5  (bank $83)
db      $00                     ; $0C IPLMPR6  (bank $80)

db      $60                     ; $0D OPENMODE
                                ;     bit 0 - Load VRAM  : 0 = Off
                                ;     bit 1 - Load ADPCM : 0 = Off
                                ;     bit 5 - BG Display : 1 = Off
                                ;     bit 6 - Play ADPCM : 1 = Off
                                ;     bit 7 - Loop ADPCM : 0 = Off

db      $00                     ; $0E GRPBLK_H
db      $00                     ; $0F GRPBLK_M
db      $00                     ; $10 GRPBLK_L
db      $00                     ; $11 GRPBLN
```

```
        db      $00                     ; $12 GRPADR_L
        db      $00                     ; $13 GRPADR_H

        db      $00                     ; $14 ADPBLK_H
        db      $00                     ; $15 ADPBLK_M
        db      $00                     ; $16 ADPBLK_L
        db      $00                     ; $17 ADPBLN
        db      $00                     ; $18 ADPRATE

        db      $00                     ; $19 reserved
        db      $00                     ; $1A reserved
        db      $00                     ; $1B reserved
        db      $00                     ; $1C reserved
        db      $00                     ; $1D reserved
        db      $00                     ; $1E reserved
        db      $00                     ; $1F reserved

        db      $50,$43,$20,$45         ; |PC E|
        db      $6e,$67,$69,$6e         ; |ngin|
        db      $65,$20,$43,$44         ; |e CD|
        db      $2d,$52,$4f,$4d         ; |-ROM|
        db      $20,$53,$59,$53         ; | SYS|
        db      $54,$45,$4d,$00         ; |TEM.|
        db      $43,$6f,$70,$79         ; |Copy|
        db      $72,$69,$67,$68         ; |righ|
        db      $74,$20,$48,$55         ; |t HU|
        db      $44,$53,$4f,$4e         ; |DSON|
        db      $20,$53,$4f,$46         ; | SOF|
        db      $54,$20,$2f,$20         ; |T / |
        db      $4e,$45,$43,$20         ; |NEC |
        db      $48,$6f,$6d,$65         ; |Home|
        db      $20,$45,$6c,$65         ; | Ele|
        db      $63,$74,$72,$6f         ; |ctro|
        db      $6e,$69,$63,$73         ; |nics|
        db      $2c,$4c,$74,$64         ; |,Ltd|
        db      $2e,$00                 ; |..|

        ; Game Name (16 bytes)
        db      "                "
        ; Game Code (6 bytes)
        db      "      "
```

****

The "IPL Information Data Block" is 128 bytes long, which leaves 1920 bytes of unused space in the 2nd sector on the CD.

This space is normally wasted, but TheOldMan has found that he can put some program code in there that's "hidden" from HuC, and execute it before his HuC program starts up.

He can do this because he's found that Hudson's IPL code always loads that block into memory at $3000.

So just by setting the IPLJMP address to $3080 instead of PCEAS's default $4070, he can run his own code before jumping to the normal PCEAS/HuC startup at $4070.

Since PCEAS is designed to always load the user's program at $4000, and execute it at $4070, then this technique doesn't cause any problems.

While this is a neat trick, I'm 100% sure that the leftover space had a different purpose ... one that I've not seen mentioned before.

That is as a way for a developer to create their own IPL code that gets control of the system as-soon-as-possible after boot.

If you set the IPLBLN value to $00, which means that you don't want the IPL to load any game code at all, then you can set IPLJMP to $0080 (NOT $3080), and the IPL will jump to the 1920 bytes of code that you've put in the 2nd sector.

That's not a *lot* of code space ... but it is enough to create your own IPL program that then loads up your game code however and from wherever you wish.

Some obvious things that you could do with this are to create a startup logo that animates while the main game code is loading, or to immediately jump to a totally different boot loader if you find that you are running on a System Card 2.0 instead of a System Card 3.0.

It could also be used to create a complicated loading system that would be hard to crack/copy ... although that idea had more value in the 1980s and 1990s, before Mednafen and other emulators were written.

****

Because Hudson make you specify an IPLJMP address that is relative to the start of the sector, rather than just setting it to $3080, then I'm pretty sure that they didn't want developers to always rely on that sector getting loaded at $3000.

Console manufacturers tend not to like developers relying on the undocumented "internal behavior" of the manufacturer's system, and so I'm guessing that if you took advantage of this capability, then you would not be allowed to just assume that you code would run at $3080.

Luckily, it is easy to copy your code from whatever address the sector is loaded at, to a fixed location in memory and then just run it from there.

Here is a small example startup that would do that (copying the code to $2680 to run there) ...

```
                org     $3080

                bsr     *+2
                pla
                dec     a
                sta     <_al
                plx
                stx     <_ah
                ldy     #$12
                sta     [_ax],y
                txa
                iny
                sta     [_ax],y
                tii     $0000,$2680,$0780
                jmp     $2680+$1b

code_at_269b:   nop
```

Of course, 25 years later on, we know that Hudson never did create a different IPL, and that the 2nd sector is always-and-forever going to be loaded at $3000 ... so we don't need to bother with the code above. But it was interesting to write it, just to see what it might look like.

« *Last Edit: January 17, 2016, 04:29:06 PM by elmer* »

Report to moderator Logged

TheOldMan

Kongo Zilla

Posts: 956

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

« **Reply #95 on:** January 17, 2016, 08:55:37 PM »

Quote

More fun with the IPL...

Set the GRPBLK_ stuff and load your graphics and a palette before anything actually runs...
Set the ADPBLK_ stuff and OpenMode, and play an adpcm tune (or whatever) before loading....

Both of those also require adjusting the IPLBLK_ record number so the program will load (set it to start record of games actual code)

Stash your actual exec address (original IMLJMP, default = $4070) at the start of the free area and jump to it from the custom code area. Game should run like normal

Or for even more fun, use the custom code area to do a cd_read via mpr....and load as much as needed from the cd....(you still have to specify length, etc for cd_read)

The only big problem with this kind of stuff is HuC keeps a list of offsets for overlays. Those have to be

patched if you use the graphics/sound stuff, since the game won't start at sector 2 on disc, and HuC assumes that it will. Somewhere I have a tool that patches all that stuff, if I can ever find it again. (And someone wants to host it)

....And the final piece of the puzzle, elmer. Use the last 3 sectors of the program code (or add 3 sectors to it) and load those in the custom code. (Or, if you prefer, set the size in bytes and call cd_read with _dh = 0: that forces a byte->sector conversion) Then you can jump to a small routine that will load those last sectors into $2800.....

I *think* that will let you load 6K into the ram bank, and all of the cd card memory. Well, except for the load routine extension....which is < 64 bytes, iirc.

Report to moderator Logged

Bonknuts

Punchy Pedro

Posts: 1934

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

« **Reply #96 on:** January 18, 2016, 01:04:37 PM »

Quote

Ahh, so this is what you guys were talking about. I've used IPL to load a few graphic (and I think ADPCM) things like a splash screen, but I mostly just use it for the CD version detection routine; basically just a boot loader for the SCD or ACD program. I just jump into system ram and load all SCD/ACD banks and then jump to my exec address.

So what you guys are talking about, is similar? Jump into system ram ($3000 range) and execute detection and call CD load routines from there? While animation and/or sound is playing?

Report to moderator Logged

http://pcedev.wordpress.com/

elmer

Punchy Pedro

Posts: 2137

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

« **Reply #97 on:** January 18, 2016, 03:23:26 PM »

Quote

Quote from: Bonknuts on January 18, 2016, 01:04:37 PM

> So what you guys are talking about, is similar? Jump into system ram ($3000 range) and execute detection and call CD load routines from there? While animation and/or sound is playing?

Your small "Boot Loader" is the normal way that games are written, IMHO.

Here is my understanding of the options for getting a logo displayed and potentially animating it ...

1) Normal Game flow (no undocumented assumptions are needed)..

```
Hudson IPL
  Load graphic to VRAM
  Load ADPCM to CD RAM
  IPL displays logo in VRAM and starts playing the ADPCM jingle
  The logo does not animate at this point
  Load boot code to $2680/$3000/$4000 or wherever (MPR2/MPR3/MPR4/MPR5/MPR6)
  IPLJMP is set to somewhere in the area that you just loaded

Developer's Boot Code
  Check CD/SCD/ACD and decide what to load
  Animate logo while loading up the 3rd stage
  Load and Run 3rd stage to display an Error Message or start the Main Menu.
```

2) Custom IPL Game flow (no undocumented assumptions are needed).

```
Hudson IPL
  Load graphic to VRAM
  Load ADPCM to CD RAM
  IPL displays logo in VRAM and starts playing the ADPCM jingle
  IPLJMP is set to $0080, no assumption is made about where the IPL is loaded
  IPL jumps to $3080 as soon as the ADPCM starts playing

Developer's Boot Code
```

```
Copy $0780 bytes of boot code to somewhere specific (say $2680)
Jump to the code's new location
Start the logo animation
Check CD/SCD/ACD and decide what to load, and start loading it
Run 3rd stage once it has loaded, or when the animation has finished
```

~~3) TheOldMan's flow (assumes IPL is loaded from $2800-$37FF, which is undocumented).~~

~~- Hudson IPL~~
~~ - Load graphic to VRAM~~
~~ - Load ADPCM to CD RAM~~
~~ - IPL displays logo in VRAM and starts playing the ADPCM jingle~~
~~ - The logo does not animate at this point~~
~~ - Load HuC boot code to $4000 (MPR2/MPR3/MPR4/MPR5/MPR6 or a subset)~~
~~ - IPLJMP is set to $3080 (within the IPL sectors that the System Card loaded)~~

~~- Developer's Boot Code~~
~~ - Code at $3080 runs and bounces the logo.~~
~~ - Code jumps to $4070 to run the regular HuC startup.~~

[EDIT]
Upon further clarification we've been having a bit of a misunderstanding, and TheOldMan is actually using a variant of Method 2, but without relocating the code, and so is still assuming that it won't be overwritten when he loads up the HuC Boot Code.
That's my *current* understanding, anyway. 😉
[END EDIT]

Now, there's not really very much difference between the first 2 if your Boot Code is fairly small, but the Custom IPL option does potentially offer the developer some (small) advantages if the Boot Code is tiny, and the 3rd-stage load is very long.

I don't see *any* advantage to the 3rd method, since TheOldMan could just as easily achieve *exactly* the same end result without having to rely on the undocumented knowledge of where the System Card loads the 2 IPL sectors.

a) If he isn't moving the location of the HuC code in the ISO, then he could just set the load addresses (IPLSTA) to $3800, set the execution address (IPLJMP) to $3880, and set the CD start sector (IPLBLK) to 1 instead of 2.

b) If he has to mess around with the PCEAS ISO output anyway in order to inject the logo and sound sectors, and so has to move the HuC Boot Code location, then it would be easy to just add an extra sector on the front and change the load and execution addresses as above.

c) Just modify HuC's "startup.asm" to include the logo bouncing code and run it before the rest of the startup procedure.

Judging by the flow of PMs ... TheOldMan and I have somewhat differing opinions on the matter. 😉

« *Last Edit: January 18, 2016, 07:14:32 PM by elmer* »

Report to moderator Logged

TheOldMan
Kongo Zilla
Posts: 956

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc. Quote
« **Reply #98 on:** January 18, 2016, 06:48:33 PM »

> **Quote**
>
> Ahh, so this is what you guys were talking about. I've used IPL to load a few graphic (and I think ADPCM) things like a splash screen, but I mostly just use it for the CD version detection routine; basically just a boot loader for the SCD or ACD program. I just jump into system ram and load all SCD/ACD banks and then jump to my exec address.

Yeah, that's pretty much what we are talking about.
And debating (he from an asm, me from a Huc point of view) whether it's "better" to use the ipl empty space or load part of the program into Ram and execute it there.

We agree we disagree, mostly. Though I do have to say I could see loading initialized data and possibly kernel/irq functions into ram as a good thing. Can't do that with Huc (well, I could, but it's a lot of work)

And a note for anyone else playing with this: IIRC, it's not a background kind of operation. (At least on the HuC side). Your graphics load, and display. Then (I think) the screen blanks while the program is loading. So, alas, no loading progress screen

Report to moderator Logged

Bonknuts

Punchy Pedro

Posts: 1934

Re: Graphic, Sound, & Coding Tips / Tricks / Effects / Etc.

Quote

« **Reply #99 on:** January 18, 2016, 08:48:30 PM »

Quote from: TheOldMan on January 18, 2016, 06:48:33 PM

> (At least on the HuC side). Your graphics load, and display. Then (I think) the screen blanks while the program is loading. So, alas, no loading progress screen

That must be an huc thing. I don't remember what project it was, but I did have the logo static on screen with an adpcm jingle, while I did stuff (and CD loading) in the background.

Report to moderator Logged

http://pcedev.wordpress.com/

Quick Reply

With *Quick-Reply* you can write a post when viewing a topic without loading a new page. You can still use bulletin board code and smileys as you would in a normal post.

Post Preview Spell Check