na_th_an|Sinclair User|Join Date: Feb 2003|Posts: 978

I'm creating a game and I have several "states" for my main sprite, such as jump, facing left, facing right, etcetera. All those states require a different graphic to be rendered.

I've understood that the Move_Sprite_* functions have a parameter which is how many bytes ahead (or before if the number is negative) the next frame to be rendered is. The problem is that I don't have a way to know which frame will be next as that will depend on the player's reaction, so I need a way to change to whichever frame I need when I want.

What I am doing is deleting the sprite and re-creating it again with the new frame, some like this:

Code:

```
int animation_shift () {
        int res = 0;

        /* First we wipe up the sprite */

        sp_MoveSprAbs(sp_phantomas, spritesClip, 0, 24, 32, 0, 0);// IMPORTANT: First move it off-screen.
        sp_DeleteSpr(sp_phantomas);

        if (phantomas.facing == 0){
                if (phantomas.sal){
                        sp_phantomas = sp_CreateSpr(sp_OR_SPRITE, 3, phantomas_3_a, 1, TRANSPARENT);
                        sp_AddColSpr (sp_phantomas, phantomas_3_b, TRANSPARENT);
                        sp_AddColSpr (sp_phantomas, phantomas_3_c, TRANSPARENT);
                } else if (phantomas.frame == 0) {
                        sp_phantomas = sp_CreateSpr(sp_OR_SPRITE, 3, phantomas_1_a, 1, TRANSPARENT);
                        sp_AddColSpr (sp_phantomas, phantomas_1_b, TRANSPARENT);
                        sp_AddColSpr (sp_phantomas, phantomas_1_c, TRANSPARENT);
                } else {
                        sp_phantomas = sp_CreateSpr(sp_OR_SPRITE, 3, phantomas_2_a, 1, TRANSPARENT);
                        sp_AddColSpr (sp_phantomas, phantomas_2_b, TRANSPARENT);
                        sp_AddColSpr (sp_phantomas, phantomas_2_c, TRANSPARENT);
                }
        } else {
                if (phantomas.sal) {
                        sp_phantomas = sp_CreateSpr(sp_OR_SPRITE, 3, phantomas_6_a, 1, TRANSPARENT);
                        sp_AddColSpr (sp_phantomas, phantomas_6_b, TRANSPARENT);
                        sp_AddColSpr (sp_phantomas, phantomas_6_c, TRANSPARENT);
                } else if (phantomas.frame == 0) {
                        sp_phantomas = sp_CreateSpr(sp_OR_SPRITE, 3, phantomas_4_a, 1, TRANSPARENT);
                        sp_AddColSpr (sp_phantomas, phantomas_4_b, TRANSPARENT);
                        sp_AddColSpr (sp_phantomas, phantomas_4_c, TRANSPARENT);
                } else {
                        sp_phantomas = sp_CreateSpr(sp_OR_SPRITE, 3, phantomas_5_a, 1, TRANSPARENT);
                        sp_AddColSpr (sp_phantomas, phantomas_5_b, TRANSPARENT);
                        sp_AddColSpr (sp_phantomas, phantomas_5_c, TRANSPARENT);
                }
        }
}
```

I was wondering if that was too slow and if I should be doing this in a different way.

I've been thinking about this for a while but I can't come with a better solution.

Alcoholics Anonymous|Manic Miner|Join Date: Dec 2002|Location: Canada|Posts: 1,717

Quote:
On 2005-09-25 16:04, na_th_an wrote:
I'm creating a game and I have several "states" for my main sprite, such as jump, facing left, facing right, etcetera. All those states require a different graphic to be rendered.

I've understood that the Move_Sprite_* functions have a parameter which is how many bytes ahead (or before if the number is negative) the next frame to be rendered is. The problem is that I don't have a way to know which frame will *be next as that will depend on the player's reaction, so I need a way to change to whichever frame I need when I want.*

What I am doing is deleting the sprite and re-creating it again with the new frame, some like this:

...

I was wondering if that was too slow and if I should be doing this in a different way.

Yes, this will be slowish. Creation and deletion of sprites does a lot of background work, including memory allocation and freeing + setting up linked lists for all the 8x8 sprite chars in the sprite.

The examples I left on the example page used an array of memory displacements to the next animation frame for each frame the sprite is in and, as you say, this won't work for your situation because the next frame could be one of any number of sprites. BUT, you don't have to deal with displacements -- you can deal with absolute memory addresses of each sprite frame.

So, eg, you could have something like this:

Code:

```
int currentFrame;
int nextFrame;
currentFrame = 0;
...

if (ph.sa1)
   nextFrame = pha_sa1;
else
   nextFrame = pha_sa2;

sp_MoveSprAbs(sp_ph, clip, nextFrame - currentFrame, row, col, cpix, rpix);
currentFrame = nextFrame;
```

pha_sa1 and pha_sa2 would be addresses in memory of the first row of a sprite graphic definition.

If you have a lot of animation frames to consider, you can wind up with spaghetti "if" code. It is possible in these cases to make the code shorter, faster and easier to follow by perhaps using a state machine to take care of sprite animation or, by expanding on the idea above, taking better advantage of how the sprite graphics are laid out in memory and using the memory addresses themselves to store state.

The latter probably doesn't make any sense to anyone except me . The state machine idea is probably better anyway and will allow you to get rid of a lot of the extra variables you'd otherwise need.

na_th_an|Sinclair User|Join Date: Feb 2003|Posts: 978

Let me see if I understand what you are explaining...

When you create a sprite, you specify where in memory its graphics are. So let's imagine I'm using your technique. Let's say that I define the sprite to be standing facing right.

If I press the "left" key I need the sprite to face left:

Code:

```
int currentFrame;
int nextFrame;
```

```
currentFrame = 0;
```

At the beginning currentFrame=0. Ok. Then I press left, then I change a variable p.facing from 1 to 0 (for example). Then something like this:

Code:

```
if (p.facing)
   nextFrame = s_p_left;
else
   nextFrame = s_p_right;

sp_MoveSprAbs(sp_ph, clip, nextFrame - currentFrame, row, col, cpix, rpix);

currentFrame = nextFrame;
```

But, as far as I have understood the MoveSpr* way of working, that would print the sprite using the original sprite facing right and then it will prepare the sprite definition to print the sprite facing left in the next frame. I.e. the graphic would be 1 frame behind the actual action.

Am I right? It's not a big deal (I doubt the player even notices), but I need to understand this completely as I'm trying to write a tutorial.

This is the problem, I have several animation sequences - i.e. walking left, walking right, falling down, jumping left, jumping right... I already use a FSM to keep track of the movement, but how using it to determine which offset I have ot pass to the rendering function surpasses me. 'Cause I would have to "guess the future" - which is not possible.

I'm having a hard time to understand how this works Wouldn't have it been easier to just specify an absolute address of the graphic to be printed, or at least specify an offset (like now) but to the graphic to be printed *now* and not *in the next frame*?

'Cause maybe the problem is that I understood it badly: As far as I have understood, if you pass a "24" (for example) what you do is to print the graphic as defined in the creation of the sprite, and, in the next call, you'll print what's 24 bytes ahead, and so on. If that 24 bytes offset were added before drawing instead of after doing it, what you have explained would apply perfectly - or maybe I'm mixing everything up

My first game using z88dk+splib2 explored a few things, and this aspect (different animation sequences) is what I have to solve before writing a proper tutorial series.

Thanks for your time.

Alcoholics Anonymous|Manic Miner|Join Date: Dec 2002|Location: Canada|Posts: 1,717

I explained that very badly... here's more detail of what I was thinking:

Code:

```
main()
{
   int currentFrame, nextFrame;
   ...

   // create the sprite; on creation, sprite
   // is automatically placed offscreen
   sp_phantomas = sp_CreateSpr(sp_OR_SPRITE, 3, phantomas_1_a, 1, TRANSPARENT);
   sp_AddColSpr (sp_phantomas,  phantomas_1_b, TRANSPARENT);
   sp_AddColSpr (sp_phantomas, phantomas_1_c, TRANSPARENT);
   // place it in middle of screen initially
   sp_MoveSprAbs(sp_phantomas, clip, 0, 15, 11, 0, 0);

   // currentFrame always holds address of current sprite graphic used
   currentFrame = nextFrame = phantomas_1_a;

   ...

   /* main game loop */

   while (game_on) {

      // update screen -- everything drawn at once
      sp_UpdateNow();

      // do collison detection
      // read joysticks

      // move & animate sprite

      // nextFrame is set to address of
      // next sprite graphic to display

      // nextFrame = currentFrame here,
      // so it is okay if the following
      // if block does not assign a new
      // value to nextFrame

      if (...)
         nextFrame = sp_phantomas_2_a;
      else if (....)
         nextFrame = sp_phantomas_3_a;
      (etc)

      // now move the sprite and draw with
      // graphic selected above

      sp_MoveSprAbs(sp_phantomas, clip, nextFrame - currentFrame, row, col, cpix, rpix);
      currentFrame = nextFrame;

   }  /* game loop */
```

The idea is that the variable currentFrame holds the absolute address of the sprite graphic currently being displayed. The animation loop selects a new absolute address for the next sprite graphic to display and stores it in nextFrame. The call to sp_MoveSprAbs then uses "nextFrame - currentFrame" as the displacement to the next graphic to display. currentFrame is set to nextFrame right after the call to store the address of the new graphic being displayed (and which will be drawn on next call to sp_UpdateNow).

Quote:
If I press the "left" key I need the sprite to face left:
Code:

int currentFrame;
int nextFrame;
currentFrame = 0;

I messed this part up so see if the above makes sense.

Quote:
But, as far as I have understood the MoveSpr way of working, that would print the sprite using the original sprite facing right and then it will prepare the sprite definition to print the sprite facing left in the next frame. I.e. the graphic would be 1 frame behind the actual action.*

On the next call to sp_UpdateNow, the sprite will be drawn with updated graphics and position. The call to sp_MoveSprAbs does not itself change the display (I know you know this but others are reading too).

Here's what sp_MoveSprAbs does:

- mark the area the sprite currently occupies for update
- update coordinates
- modify sprite graphic pointers given the displacement passed in
- mark the area the sprite now occupies (with the new coordinates) for update.

The following call to sp_UpdateNow redraws all areas marked for update (in this case, the area where the sprite was originally and the area where the sprite was moved to).

Quote:
This is the problem, I have several animation sequences - i.e. walking left, walking right, falling down, jumping left, jumping right... I already use a FSM to keep track of the movement, but how using it to determine which offset I have ot pass to the rendering function surpasses me. 'Cause I would have to "guess the future" - which is not possible.

I think you overcomplicated what sp_MoveSprAbs is doing, but hopefully the above explains it. Just use your FSM to select the next graphic to display as above and it'll be fine

Quote:
I'm having a hard time to understand how this works Wouldn't have it been easier to just specify an absolute address of the
graphic to be printed, or at least specify

I am revisiting splib now as it's being integrated into z88dk so I will look at this again. The animation idea came late on in the implementation of splib2 and that was the easiest way to add it with zero runtime overhead.

Quote:
*an offset (like now) but to the graphic to be printed *now* and not *in the next frame*?*
Well the next frame thing is central to how this sprite library works -- you do all drawing at once with sp_UpdateNow, then move sprites to new locations based on user (or program) control using calls to sp_MoveSpr and then update everything again on the next sp_UpdateNow.*

Most sprite engines do things this way, so your consternation might just be a misunderstanding. The simpler ones draw things immediately to the screen, but I think the better ones have to queue up screen draws to sync with the raster in order to eliminate flicker.

This library does not do that, but instead effectively queues up all draws to avoid redrawing the same areas of the screen over and over, and to minimize flicker. It will also draw *all* graphics in a particular char square into an 8-byte buffer before copying the final result to the screen.

The next version of splib that is going into z88dk will be slightly different and will hopefully address the sprite shear you may notice on larger sprites or when there are a lot of sprites in the same rows. It will also introduce windows, which should be pretty cool

I'll take this opportunity to plug what's been added to z88dk recently (available through cvs only and not entirely tested):

- Abstract data types (heap, linked lists, hash table, queue, stack)
- Block memory allocator
- Generic IM2 library
- quicksort + binary search added to stdlib
- Input library for reading keyboard, mice and joysticks

All written in assembler. It would be nice to see more authors out there share their code this way...particularly with some of the nice sprite engines I've seen on this forum.

na_th_an|Sinclair User|Join Date: Feb 2003|Posts: 978

Awsesome, thanks. Now I understand it. My problem #1 was that I hadn't got what MoveSprAbs was doing exactly. Now all is cleared up. I thought for some reason that the new animation frame would appear 1 frame later than it actually does. Once I understood how the function works - no probs from now I find it pretty straightforward.

Thanks for your explanation, it has been very informative.

I have to give the CVS a try and attempt to compile it. I've seen you added inb and outb support to stdlib and I need that I was fiddling with z88dk and in my future include coding a paging library for the 128K spectrums and attempt to use it from z88dk+splib2. With some luck, I'll be coding the first 128K game with AY music and bank switching using z88dk

Alcoholics Anonymous|Manic Miner|Join Date: Dec 2002|Location: Canada|Posts: 1,717

Quote:
On 2005-09-27 08:46, na_th_an wrote:
I find it pretty straightforward.

Hah, good. If you have issues with available memory, there are a couple of things you can do with the sprite definitions:

- Every sprite graphic is defined with alternating MASK and GRAPHIC bytes. If the sprites are OR,XOR or LOAD type they don't make use of the mask byte. What you an do is interleave the sprite definitions of the OR/XOR/LOAD bytes so that the graphics of one sprite sit in the mask bytes of the graphics of another sprite. This way you fit two sprites into the space of one.
- Sprites that can be placed at pixel precision horizontally (as opposed to being confined on exact character cells) need that extra blank column on the right of the sprite definition. You can avoid explicitly creating that extra blank column on the right by redirecting the graphic pointers in that column to point at sp_NullSprPtr. There is an example of this being done on the tutorial page (it involves a call to sp_IterateSpr).

Quote:
was fiddling with z88dk and in my future include coding a paging library for the 128K spectrums and attempt to use it from z88dk+splib2. With some luck, I'll be coding the first 128K game with AY music and bank switching using z88dk

I look forward to seeing this If it comes to it, it won't be completely straightforward to force the assembler to confine code to certain pages -- this will probably require you to come up with a compile script as opposed to using the default zcc. z80asm is not >64K aware but it does allow map generation that can generate defines for locations of subroutines that would be useful.

I am actually looking at the beep code now and will clean that up a bit, the main reason for doing so is to get rid of the di/ei in there so that it doesn't mess with the program. Hopefully people won't find it necessary to include their own assembly beep routines once that's done. I'd also like to include a Wham and Orfeus player for music -- I may have to be careful re the license z88dk is distributed under and the rights of the copyright owners so I may have to write a new player from scratch that accepts compiled music in those formats. AY is on the radar too but may be a while.

na_th_an|Sinclair User|Join Date: Feb 2003|Posts: 978

Thanks for the tips, I'll take them in account.

About the paging stuff: Yeah, my idea was configuring splib2 and z88dk to fit everything below C000h so I can use the last slot to page in graphics and game data.

If I define my sprites to be in a position, it doesn't matter which page is paged in as long as there's sprites in there, so splib2 would be perfectly usable.

About music, one of the developers in CEZGS (where I belong to) has managed to make Vortex tracker-generated code to work alongside splib2 just placing the calls needed to "keep the music updated" in the interrupt service routine, and it works like a charm. In moggy, I used Wham! tunes but I had to alter the player code to eliminate every di/ei, otherwise the game went nuts.

Yeah, I used the ROM BEEP routine but somewhat like a black box. A BEEP routine will be a nice addition to the library. The music player is also very interesting.

I'm still a newb on this - but this is fascinating and I am willing to learn everything. My project involving paging and stuff won't be started in a while (I have to finish some things before - there are a couple of games I must finish and release as my final rendition to compiled BASIC), but be sure that I'll keep you posted.

na_th_an|Sinclair User|Join Date: Feb 2003|Posts: 978

I have a simple question now:

When you are drawing a sprite which is not TRANSPARENT, I've noticed that it colours the bottom and right squares even if the sprite is positioned at a tile-perfect position (i.e. it's not drawing anything to the bottom and right squares). Is there any way to prevent this? I have my enemies walking through black areas so I though I could colour them; they are moving following an horizontal path with a fixed "y" coordinate which is a multiple of eight, but the coloured tiles beneath get tinted by the sprites when there's no use for it (they are not needed to tint my sprites).

I can guess that the answer is "no", but just in case

EDIT: I think I sorted it, but I leave the question and the auto-answer here just in case it's useful for somebody else:

As the enemies have fixed paths, I can define a clipping rectangle for each one of them. Would this speed up things somewhat? Would it take so much memory? (I'm using 3 enemies per screen).

Alcoholics Anonymous|Manic Miner|Join Date: Dec 2002|Location: Canada|Posts: 1,717

On 2005-09-28 16:46, na_th_an wrote:
When you are drawing a sprite which is not TRANSPARENT, I've noticed that it colours the bottom and right squares even if the sprite is positioned at a tile-perfect position (i.e. it's not drawing anything to the bottom and right squares). Is there any way to prevent this? I have my enemies walking through black areas so I though I could colour them; they are moving following an horizontal path with a fixed "y" coordinate which is a multiple of eight, but the coloured tiles beneath get tinted by the sprites when there's no use for it (they are not needed to tint my sprites).

EDIT: I think I sorted it, but I leave the question and the auto-answer here just in case it's useful for somebody else:

As the enemies have fixed paths, I can define a clipping rectangle for each one of them. Would this speed up things *somewhat? Would it take so much memory? (I'm using 3 enemies per screen).*

Actually the clipping rectangle thing is a good solution. Sprites are always clipped against a rectangle (usually the full screen rectangle sp_ClipStruct) so using another smaller one affects nothing negatively speed-wise. It will slightly speed things up since that extra row/col of the sprite won't be drawn. 'Course the clipping rectangle must always be confined to the screen... if it extends past the screen boundaries nasty things will happen.

There are other things that can be done:

- If the sprite is always drawn at an exact horizontal char coordinate (requiring no shifting), as might be the case for an enemy that only moves vertically, you don't have to add that blank column on the right of the sprite definition. The blank column on the right will not be drawn and the sprite will be smaller to boot, making it slightly faster to draw.
- Likewise if the sprite is drawn at exact vertical character coordinates (requiring no vertical shifting) you don't need to have a blank character square at the bottom of each sprite's column (nor the 7 blank pixel rows above each column). The sprite's height can be made smaller by one and the empty row at bottom will not be drawn.

If the cases above don't fit, you can use the clipping rectangle as an easy option.

Another option that requires a deeper understanding of the mechanics of splib:

Every sprite is broken into character cell size pieces described by a struct sp_CS:

Code:

```
struct sp_CS {
   uchar *next_in_spr;  /* big endian!! */
   uchar *prev;         /* big endian!! */
   uchar spr_attr;      /* sprite type (bits 6..7) | sprite plane (bits 0..5) */
   uchar *left_graphic;
   uchar *graphic;
   uchar hor_rot;
   uchar colour;        /* attribute in spectrum mode, threshold in hi-colour mode */
   uchar *next;         /* big endian!! */
   uchar unused;
};
```

You'll notice that one of the characteristics of a sprite char in this structure is the colour. An easy way to gain access to these pieces is by using the sp_IterateSprChar function.

If you're sprite is 3x3 chars in size it's graphics are defined as follows:

ABC
DEF
GHI

ADG was the first column made when sp_CreateSpr was called then BEH and CFI were added with two sp_AddColSpr calls.

A call to sp_IterateSprChar will call a function you supply nine times, one time for each of those character cells. It calls your funtion passing in a "struct sp_CS" in column major order -- in this case in the order ADGBEHCFI.

In-game you could use something like the following to colour those blank squares transparent:

Code:

```
int N;

void colourTransparent(struct sp_CS *c)
{
   if ((N==2) || (N>=5))
      c->colour = TRANSPARENT;
   N++;
}

...

main()
{
   ...
   N=0;
   sp_IterateSprChar(mySprite, colourTransparent);
}
```

Notice how N is initialized to 0 and is increased inside "colourTransparent" so that it counts which char cell in the sprite is being examined. Inside "colourTransparent" N=0 for A, N=1 for D, etc.

I don't really like the idea of iterating over a sprite while in the game loop. Another option is to get addresses of the colour members in the "struct sp_CS" before the game loop begins and then using those addresses to set colours while in the loop:

Code:

```
int N;
uchar *gC, *gF, *gG, *gH, *gI;

void getAddresses(struct sp_CS *c)
{
   switch(N) {
      case 2:
         gG = &c->colour;
         break;
      case 5:
         gH = &c->colour;
         break;
      case 6:
         gC = &c->colour;
         break;
      case 7:
         gF = &c->colour;
         break;
      case 8:
         gI = &c->colour;
         break;
      default:
         break;
   }
   N++;
}

...
main()
{
    ...
    // sprite initialization
    N=0;
    sp_IterateSprChar(mySprite, getAddresses);
    ...

    while(game_on) {
       ...
       // now we colour those problematic
       // tiles transparent

       *gC = *gF = *gG = *gH = *gI = TRANSPARENT;
```

```
         ...
      }
}
```

I think I prefer the clipping

EDIT:
I'll just add here that if I were to change all the colours of a sprite frequently within the game loop, I'd enlist the help of an assembly function. Taking the same hypothetical 3x3 sprite as above as an example, I might do something like the following:

Code:

```
uchar *sprClrAddr[9];
uchar **temp;
uchar clrBlock[] = {
   INK_BLUE | PAPER_YELLOW,
   ... 9 times for the 9 chars in sprite
};

void storeClrAddr(struct sp_CS *c)
{
   *temp++ = &c->colour;
}

void colourSpr(uchar *colours, uchar **addr, int n)
{
   #asm
   ld hl,2
   add hl,sp
   ld a,(hl)    ; a = N
   inc hl
   inc hl
   ld e,(hl)
   inc hl
   ld d,(hl)    ; de = addr
   inc hl
   ld c,(hl)
   inc hl
   ld b,(hl)    ; bc = colours
   ex de,hl     ; hl = addr*

loop:
   ld e,(hl)
   inc hl
   ld d,(hl)    ; de = addr of spr attr
   inc hl
   ex af,af
   ld a,(bc)    ; get clr from clr block
   inc bc
   ld (de),a    ; write clr into struct sp_CS
   ex af,af
   dec a
   jp nz, loop
#endasm
}

...

main()
{
   ...
   // sprite initialization
   temp = sprClrAddr;
   sp_IterateSprChar(mySprite, storeClrAddr);

   ...

   while(game_on)
   {
      ...
      // now colour sprite with colours
      // stored in colour block

      colourSpr(clrBlock, sprClrAddr, 9);

     ...
   }
}
```

sp_IterateSprChar is used to write all the struct sp_CS.colour addresses into the sprClrAddr[] array. clrBlock[] holds the colours of 9 sprite chars. The function colourSpr quickly copies the colours in clrBlock[] to the addresses specified in sprClrAddrp[] in order to colour the entire sprite.

The assembler can be improved and I'm sure the rest of the code can be cleaned up to, but that's a way to quickly change the colours of a sprite fairly quickly.

Alcoholics Anonymous|Manic Miner|Join Date: Dec 2002|Location: Canada|Posts: 1,717

Quote:
On 2005-09-28 15:32, na_th_an wrote:
About the paging stuff: Yeah, my idea was configuring splib2 and z88dk to fit everything below C000h so I can use the last slot to page in graphics and game data.

If I define my sprites to be in a position, it doesn't matter which page is paged in as long as there's sprites in there, so splib2 would be perfectly usable.

You also have to keep in mind that splib2 has a large rotation table that by default sits in the top page, and this must be available when (and only when) sp_UpdateNow runs. It can be configured to move that table elsewhere or you could have a copy of it in every ram page.

Interrupts too you have to be careful with, making sure that the interrupt code (and any music player + music data) are paged in or are paged in when the interrupt routine runs.

All these details make it hard to fit all code below c000. You could also go with another model: you have 48k RAM during gameplay but you use the extra RAM as a RAMdisk, copying things to and from other ram pages.

Anyway it's not impossible whatever you wind up doing. You'll be the first to try it AFAIK so it will be a learning exercise, hopefully an interesting one!

Quote:
(where I belong to) has managed to make Vortex tracker-generated code to work alongside splib2 just placing the calls needed to "keep the music updated" in the interrupt service routine ... In moggy, I used Wham! tunes but I had to alter the player code to eliminate every di/ei, otherwise the game went nuts.

Yeah, I've also been using Soundtracker in an interrupt as well, and I've also looked at your Moggy code so I know what was going on there Santiago's programs also use their own beep code so I had to look at what was wrong with z88dk's built in stuff and I found it isn't easily useable as is in large part because of the di/ei stuff.

Quote:
I'm still a newb on this - but this is fascinating and I am willing to learn everything. My project involving paging and stuff won't be started in a while (I have to finish some things before - there are a couple of games I must finish and release as my final rendition to compiled BASIC), but be sure that I'll keep you posted.

Please do. What has really sucked me in with z88dk is that it is so easy to integrate your own assembler when you feel the need to and at the same time you can do plenty in C, which is fast enough to be useful. I think it can speed up development time many, many times and makes the whole experience more like programming in BASIC than assembler, where many projects are frustrated because of lack of time or all the minutae.

na_th_an|Sinclair User|Join Date: Feb 2003|Posts: 978

You are completely right. Before z88dk, I would have never considered stepping into assembly coding. Now I try to convert the most critial parts to assembly within the C skeleton, which is a good thing. We should try and get more people to try z88dk, I find it the most affordable way to create a nice, playable game.

Thanks for the post about my solved issue (with clipping rectangles), it was really illustrative. Indirectly, it made me understand lots of things, now I know your library more in depth - a fact that has opened lots of doors to me, trust me. Now I understand how things work. Interesting stuff being able to omit the third column if the sprite is gonna be in a tile-perfect horizontal position or remove the last 8 rows of zeros if the sprite's gonna be on a tile-perfect vertical position &lt;- this is the kind of things you know (I read the docs in your site from top to bottom) but you never think about when problems like this arise

Yeah, the paging stuff WILL be tricky. First of all, 'cause following my idea I would have to fit LOTS of things below C000h. Anyhow, that big structure you mention (rotation tables and stuff)... is only used by the update function? If so, it's not a "big" problem, I can always page in the main page 0 before updating. Paging doesn't take much time, so it wouldn't impact on the game.

So this is the idea: page in what you need when you need it. When I need to draw the screen, page in the page with the map and the tiles and render the screen. Then I need to place my sprites and move them, then I page in the page with the sprites, and call to MoveSprAbs and stuff. When I get to the point where I must update everything, I page in page 0 (or the one I had paged in when I initialized the library) and call to sp_UpdateAll.

The more I think about this, the easier will be when I start the project. But things will be slow as this will be a big CEZGS project and there are many things to sort out and to design - mainly tons of graphics and a nice framework where it's easy to design map data and place the baddies and stuff.

EDIT: I forgot commenting about a couple of things. You mentioned using RAM as a RAM disk to copy stuff when needed. That made me wonder if is there any way to access the 128K ramdisk from z88dk. It would be great news - a easy way to store stuff. I've been told that z88dk programs are capable to open files for read/write in a floppy in the Spectrum +3, so I was wondering if it was possible to do the same in the RAM disk.

Alcoholics Anonymous|Manic Miner|Join Date: Dec 2002|Location: Canada|Posts: 1,717

Quote:
On 2005-09-29 09:14, na_th_an wrote:
The more I think about this, the easier will be when I start the project. But things will be slow as this will be a big CEZGS project and there are many things to sort out and to design - mainly tons of graphics and a nice framework where it's easy to design map data and place the baddies and stuff.

It's definitely doable and you'll probably have to get a pencil and paper out to map where everything goes. If you need some info later on, I'd be happy to let you know what has to be in memory when. By the time your project is underway there may be an improved lib available too.

Quote:
EDIT: I forgot commenting about a couple of things. You mentioned using RAM as a RAM disk to copy stuff when needed. That made me wonder if is there any way to access the 128K ramdisk from z88dk. It would be great news - a

easy way to store stuff. I've been told that z88dk programs are capable to open files for read/write in a floppy in the Spectrum +3, so I was wondering if it was possible to do the same in the RAM disk.

Yep, there is +3 disk access, tape and microdrive access in z88dk now. The ZXVGS port (Spectrum with Jarek Adamski's enhancements; the SE is a ZXVGS compatible machine, eg) can also do CF and IDE things.

But nothing with the 128's Basic RAMdisk. A RAMdisk is very easy to implement and I think most authors would prefer just to ldir stuff from one bank to another, but there might be some merit in supporting a RAMdisk abstraction for machines with > 64K. Might have to think about that one...

Another idea I can leave to germinate here is that some data compression functions will be added in the not-too-distant future. The first implementation will include a static and a dynamic Huffman encoder / decoder. These are fairly quick functions. For the decoder, you would simply specify where the compressed data begins and then call a function that would return one byte at a time uncompressed. Might fit well with the ramdisk notion.

na_th_an|Sinclair User|Join Date: Feb 2003|Posts: 978

In a couple of to-be-released games I use the pucrunch compression alongside the decompressor which was ported from C64's 6902 to MSX's Z80 and then it was adapted to be callable from Sinclair BASIC by one of my pals (iforeve). The compressor is like magical with the ratios it gets, and the decompressor is fairly fast. I don't have an usable compressor that runs on a Z80 (I compress the files in my PC then import the raw data to the emulator), but I'm pretty sure that in the pucrunch homepage they featured one. Maybe you are interested in this as the co/dec routines for Sprite Pack.

I don't know what you have in mind to enhace the library (that sounds like excellent news, tho'), but I have a few suggestions First of all, you should add some "drawtile" function which would draw a variable-sized static tile (i.e. like a Sprite but just blit it to the screen and forget about it: it won't move). Yesterday I discovered that you just have 1 8x8 bitmap per character to build tile data, that makes only 96 8x8 blocks. If you could use a set, print with it, and select another (like when switching codepages using 23606 and 23607), that wouldn't be a problem, but I discovered yesterday that the BASIC trick of "switch to the graphics charset, draw the background; switch to the font charset, print the text" don't work 'cause when you change the charset, what's already printed on screen also changes.

Another good adition would be the possibility of configuring the library to accept a fixed size for sprites. I.e. imagine you compile a fixed version of the library where you define that sprites will always be 16x16. That library version would be much faster 'cause the fixed sprite dimmensions and the fact they are powers of two would take away lots of calculation.

I have (yet) another question: If I draw something to the screen externally - I mean, for example, using some MC located somewhere which decompresses a screen into video memory @ 4000h, will the next call to sp_UpdateAll() take this in account? If not, how could I feed that data to the library so it takes it in account? It's 'cause I want to use some fixed background in my games which I would decompress to the screen and then draw the tiles and render the game over it with splib2.

na_th_an|Sinclair User|Join Date: Feb 2003|Posts: 978

Okay, sorted it out The trick is never invalidate the things you don't want to be removed

Now I have my great score board.

Alcoholics Anonymous|Manic Miner|Join Date: Dec 2002|Location: Canada|Posts: 1,717

Quote:
On 2005-09-30 08:49, na_th_an wrote:
pucrunch compression alongside. The compressor is like magical with the ratios it gets, and the decompressor is fairly fast. I don't have an usable compressor that runs on a Z80 (I compress the files in my PC then import the raw data to the emulator), but I'm pretty sure that in the pucrunch homepage they featured one.

Definitely interested in this, thanks! Also found links to bitblaster and chrust. I'll see if I can modify these (or rewrite them somewhat) to allow byte at a time decompression. I'd like to use these things as a compression/decompression layer so you could, eg, save compressed data to disk or make use of compressed data without decompressing the whole thing into memory. Something like having your AY music compressed and calling these routines to decompress just enough info into a small buffer to set registers in the next interrupt. Same thing with sprite data: have starting points in the compressed file and only decompress those sprites being used.

Quote:
but I have a few suggestions

Suggestions always welcome The lib is not perfect and cannot do everything. Because of the underlying display algorithm used it may not be able to do what you want it to either, even with improvements. Neither will any other sprite lib be able to do everything, but then that is why we need many libs to choose from.

splib is just the beginning as far as I am concerned; I've got two others in development too: the first is a 64x48 full colour zx81 mode with sprites and graphics primitives. Every pixel can be any colour. This will be added shortly after the next splib as it's fairly close to being finished. The 2nd still isn't fleshed out, but its aim is to be able to do something like Green Beret and it is still far off. It would be great to see more people share their code in the form of libs as well...

Quote:
First of all, you should add some "drawtile" function which would draw a variable-sized static tile (i.e. like a Sprite but just blit it to the screen and forget about it: it won't move).

A couple things you can look at:

- Build your large tiles from sp_PrintString strings. These strings can contain control codes to change relative print position, so you can print these "large tile" strings at any point on screen and have them clipped by sp_PrintString automatically. This won't change the fact there are only 256 8x8 tiles to build these tile strings from...
- Use load sprites for the background. I had originally thought of using these to enable "backgrounds" that your main sprite can walk behind.

Quote:
Yesterday I discovered that you just have 1 8x8 bitmap per character to build tile data, that makes only 96 8x8 blocks.

Yeah, tiles are 8x8 udg characters, but there are 256 of them (codes 0..255). The sp_PrintAt functions will print them correctly, but the sp_PrintString function hijacks some codes for control characters. With sp_PrintString you can precede codes with an esc character so they aren't hijacked.

Quote:
If you could use a set, print with it, and select another (like when switching codepages using 23606 and 23607), that wouldn't be a problem, but I discovered yesterday that the BASIC trick of "switch to the graphics charset, draw the background; switch to the font charset, print the text" don't work 'cause when you change the charset, what's already printed on screen also changes.

Yep, that's right. The only time the tile graphics are referenced is inside sp_UpdateNow when the screen is drawn. Whatever the tile array holds at that moment is what is used as the background tile.

It was essentially a memory saving technique that was employed -- the entire background screen is made up of (up to) 256 udgs, or a maximum memory requirement 1/3 of the Spectrum screen, and can be much less if not all the tiles are used. The alternative was to store an entire Spectrum-size screen in memory (6912 bytes) for background and that was quite a bit on top of all the other tables and whatnot that have to be there.

I am using a codepage idea for different windows in the next version and I will consider using a 16-bit tile id instead of 8-bit: if the id is &lt;=255, it's a tile drawn like usual, else it's a pointer to the tile's graphic description. I'll see how it goes.

Quote:
Another good adition would be the possibility of configuring the library to accept a fixed size for sprites. I.e. imagine you compile a fixed version of the library where you define that sprites will always be 16x16. That library version would be much faster 'cause the fixed sprite dimmensions and the fact they are powers of two would take away lots of calculation.

With this library the size doesn't matter as everything is about character cells. The sprites are broken into character pieces and those are associated with a char cell on screen -- this is how the lib can completely draw a single character cell composed of background and all sprites all at once. It's also why it's able to do arbitrary clipping to char cell boundaries so easily.

However, I am doing things that will speed it up (I don't know how much yet) and will also make it appear to speed up. One of the bad things the lib does is update the screen top to bottom, left to right. If you've got ten sprites of height two in the same char row, the top row of all ten sprites will be drawn before the bottom row is drawn.... it is probable that you'll see this on screen as shear and this makes the lib seem slower than if five of the sprites were completely drawn (top and bottom) and five weren't updated yet.

Quote:
I have (yet) another question: If I draw something to the screen externally - I mean, for example, using some MC located somewhere which decompresses a screen into video memory @ 4000h, will the next call to sp_UpdateAll() take this in account? If not, how could I feed that data to the library so it takes it in account? It's 'cause I want to use some fixed background in my games which I would decompress to the screen and then draw the tiles and render the game over it with splib2.

As you found out, this is not something that splib can do well.

If the area is never drawn to (invalidated by) splib, it will not be updated and the area will remain intact. You can ensure this by using appropriate clipping rectangles.

You can also sp_Validate the area before each call to sp_UpdateNow to make sure it is not drawn to, but the first option is preferable.

Another thing you can do is recompile splib to use only a portion of all the rows on-screen. It can be configured, eg, to use only rows 4-20 or some other portion. The coordinate (0,0) is always mapped to the top left corner of the area used by splib. Less memory will be necessary to keep track of the smaller screen area and draw times will improve slightly.

If you want splib's sprites to float over an arbitrary background, it can't be done the way things are now. That background must either be composed of tiles or load sprites unfortunately. With sprites, it's possible to set the graphics pointed at to some arbitrary point in memory which could contain a screen image or somesuch. Not exactly elegant though.

na_th_an|Sinclair User|Join Date: Feb 2003|Posts: 978

First of all, thank you very much for your long posts. I'm really starting to know your lib in & out. I'm saving this post - it's the perfect companion to the tutorial you have on your site.

Quote:
On 2005-10-02 04:18, Alcoholics Anonymous wrote:

Definitely interested in this, thanks! [...] I'll see if I can modify these (or rewrite them somewhat) to allow byte at a time decompression. [...]

Wow - that sounds amazing. About the pucrunch decompressor, I have been playing with it discovering that it doesn't "compile" if you embed it in C code (i.e. Z80ASM can't assemble it). The assembler finds problems with some "ex bc, bc'" instructions and the like (maybe it doesn't accept the notation for the alternate set of registers - I've been researching on this but I haven't found proper documentation for Z80ASM). It assembles perfectly with Pasmo, tho'. I'll post the adapted decompressor here later, not at home at the mo.

Quote:
splib is just the beginning as far as I am concerned; I've got two others in development too: the first is a 64x48 full colour zx81 mode with sprites and graphics primitives. Every pixel can be any colour. This will be added shortly after the next splib as it's fairly close to being finished.

This is definitely interesting. VERY interesting. The way you'll get 4 colours out of every character cell - I don't know, but I will defintely doing something with it.

Quote:
The 2nd still isn't fleshed out, but its aim is to be able to do something like Green Beret and it is still far off.

Do you mean adding native support for pixel*pixel scrolling? Whoah man - that will make a lot of people's dreams come true.

Quote:
It would be great to see more people share their code in the form of libs as well...

Yeah. Sadly, most people think they are so 'leet to share their code. I don't understand that. We all learned to code looking at some else's code. To me, sharing all my sources is a kind of pay-back. That's why I'll be writing a complete tutorial when I manage to add paging to a splib2-driven engine, which I find that will be something really USEFUL.

Quote:
[...]are only 256 8x8 tiles to build these tile strings from...

Now I get it. Don't worry, 256 is far enough (it's 2.5 times what I "had" until now.

Now I understand how your library works. Really intelligent. I understand why it is the way it is and I think it is the better way to achieve fast and tight results.

Quote:
I am using a codepage idea for different windows in the next version and I will consider using a 16-bit tile id instead of 8-bit: if the id is &lt;=255, it's a tile drawn like usual, else it's a pointer to the tile's graphic description. I'll see how it goes.

That will be a good thing. It will double the amount of space taken - maybe this will be acceptable for some applications, but not for others. I don't know how difficult would it be to have that feature selectable at build time, i.e. you can build the lib with or without that.

Quote:
With this library the size doesn't matter as everything is about character cells.

Now I got it, thanks.

Quote:
However, I am doing things that will speed it up (I don't know how much yet) and will also make it appear to speed up. One of the bad things the lib does is update the screen top to bottom, left to right. If you've got ten sprites of height two in the same char row, the top row of all ten sprites will be drawn before the bottom row is drawn.... it is probable that you'll see this on screen as shear and this makes the lib seem slower than if five of the sprites were completely drawn (top and bottom) and five weren't updated yet.

So you are talking of altering the order the screen is drawn. I've read somewhere that ultimate used to update the screen bottom to top to somewhat use the retrace to eliminate shearing - I dunno how, but hey.

Quote:
If the area is never drawn to (invalidated by) splib, it will not be updated and the area will remain intact. You can ensure this by using appropriate clipping rectangles.

Quote:
If you want splib's sprites to float over an arbitrary background, it can't be done the way things are now. That background must either be composed of tiles or load sprites unfortunately.

Well, as you said, no library can do everything. Now I understand that if you wanted the library to do what I suggested, a whole 6912 bytes buffer should have to be used. No worries, I wanted this feature just to draw a nice score board and stuff - now I know how to use it, thanks.

As I said before, I'm starting to know this library in and out and that's great to know what you can do and how. And there are many things you can do to overcome de-facto library limitations, and that's great.

Thanks again for you efforts. My current game is like 10 times more complex library-usage-wise than moggy - and all thanks to you.

Greetings.

Alcoholics Anonymous|Manic Miner|Join Date: Dec 2002|Location: Canada|Posts: 1,717

Quote:
About the pucrunch decompressor, I have been playing with it discovering that it doesn't "compile" if you embed it in C code (i.e. Z80ASM can't assemble it). The assembler finds problems with some "ex bc, bc'" instructions and the like (maybe it doesn't accept the notation for the alternate set of registers - I've been researching on this but I haven't found proper documentation for Z80ASM).

Yeah that's exactly right. z80asm has some great features but it has quirks too. Instructions with an apostrophe in them cause trouble for the parser (actually I don't know if it's the C compiler or the assembler that causes the problem) so instead of "ex af,af'" use "ex af,af".

Quote:
Quote:
development too: the first is a 64x48 full colour zx81 mode with sprites and graphics primitives. Every pixel can be any colour.
This is definitely interesting. VERY interesting. The way you'll get 4 colours out of every character cell - I don't know,

but I will defintely doing something with it.

It requires timing synced to the raster, so all sprite draws, line draws, etc are written as fixed-length (timewise) pieces that the programmer organizes into a display list that is executed during the interrupt. With code guaranteed to run at regular intervals, it may even be possible to play digitized music and sound effects in game. That might be interesting

Quote:
*Do you mean adding native support for pixel*pixel scrolling?*
That's the goal.

Quote:
So you are talking of altering the order the screen is drawn. I've read somewhere that ultimate used to update the screen bottom to top to somewhat use the retrace to eliminate shearing - I dunno how, but hey.
Yeah I will be changing the draw order, doing chars in sprite order rather than position order. The drawing bottom up thing ensures you only have shear at one pixel row on display but only works if you have everything drawn in a back buffer and are copying or if you actually draw completed graphics one pixel row at a time.

Quote:
As I said before, I'm starting to know this library in and out and that's great to know what you can do and how. And there are many things you can do to overcome de-facto library limitations, and that's great.

Thanks again for you efforts. My current game is like 10 times more complex library-usage-wise than moggy - and all thanks to you.
Glad to hear!