basix

COLLABORATORS			
	TITLE : basix		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		August 8, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

basix

1.1 2 TCS Basics

2 TCS Basics

This section purposes to provide all the means to use the tcs.library to a great extent and to help you familiarize with the basic concepts more formally defined in the techie section (which, however, you could need to have a look at for the most complex/custom/tricky things).

[some work to be done here...]

2.x
Performance
2.y
Some Considerations

1.2 2.x Performance

2.x Performance

Now I’ll let the numbers talk at my place.
We can look at performance from two different points of view:

2.x.1
Speed
2.x.2
Memory Needs

- all the figures in the sections above apply to the tcs.library

1.3 2.x.1 Speed

2.x.1 Speed

In this section are shown the results deriving from many tests based on a loop which executed the following piece of code 100 times:

```
 .fill   rept     16
         move.b   d1,(a1)+   ;write 1 pixel
         endr
         dbra     d3,.fill
```

to fill a 160x256 or 320x256 screen, depending on the resolution used (d3 holds the size in bytes of the screen).

MskPln	ChqrMode	HScrl	resolution	machine	secs	fps
OFF	OFF	OFF	HalfRes	A1200+Bz1230	2.320	43.10
OFF	OFF	ON	HalfRes	A1200+Bz1230	2.320	43.10
OFF	ON	OFF	HalfRes	A1200+Bz1230	2.389	41.86
OFF	ON	ON	HalfRes	A1200+Bz1230	2.406	41.56
ON	OFF	OFF	HalfRes	A1200+Bz1230	2.709	36.91
ON	OFF	ON	HalfRes	A1200+Bz1230	2.732	36.60
ON	ON	OFF	HalfRes	A1200+Bz1230	2.804	35.66
ON	ON	ON	HalfRes	A1200+Bz1230	2.853	35.40
OFF	OFF	OFF	FullRes	A1200+Bz1230	3.564	28.50
OFF	OFF	OFF	HalfRes	bare A1200	2.320	43.10
OFF	OFF	ON	HalfRes	bare A1200	2.320	43.10
OFF	ON	OFF	HalfRes	bare A1200	2.389	41.86
OFF	ON	ON	HalfRes	bare A1200	2.407	41.55
ON	OFF	OFF	HalfRes	bare A1200	2.710	36.90
ON	OFF	ON	HalfRes	bare A1200	2.733	36.59
ON	ON	OFF	HalfRes	bare A1200	2.804	35.66
ON	ON	ON	HalfRes	bare A1200	2.854	35.38
OFF	OFF	OFF	FullRes	bare A1200	13.913	7.19

(the Bz1230-IV was clocked at 50 Mhz and was equipped with 60 ns ram; FullRes conversion was done without Blitter's help)

The first thing that strikes our eyes is that the unxepanded A1200 perfor-med exactly like the powered-up one in all but one mode: FullRes.
This can be looked at as a "proof of quality" for TCS displays: they offer chunky and TrueColor-like screens for free. You may ask: if it's for free, why can't 50 fps be reached? The answer, concerning HalfRes modes, is that the program writes single bytes to the slow CHIP ram, so the 24-bit bus is badly used. Using the same loop as above, but writing longwords instead of bytes, the figures become:

```
+----------+--------+-------+-----------+-------------+-------+--------+
```

ChqrMode	MskPln	HScrl	resolution	machine	secs	fps
ON	ON	ON	HalfRes	A1200+Bz1230	0.710	140.83
ON	ON	ON	HalfRes	bare A1200	0.710	140.83

in fact, considering the corresponding number of the bigger table, we have that: 2.853/4 = 0.713, which is quite close to 0.710.

With regard to FullRes, we have already discussed the reason of such a big drop of performance; now let’s see how much a 320x256 FullRes screen costs to the Amiga (doing _nothing_ else than showing the screen):

machine	secs	fps	
A4000+CS060	1.369	73.05	(old values)
A1200+Bz1230	2.550	39.20	
bare A1200	9.270	10.13	

(the CSII-060 was clocked at 50 Mhz and had a 70 ns simm)

You could think that there’s a discrepancy here; by looking at the first table, one might expect that the time taken by FullRes operations should be equal to the time elapsed for filling a screen in FullRes mode subtracted by the time taken for filling in HalfRes (MskPln and ChqrMode OFF) mode multiplied by 2 (HalfRes screen is half of FullRes’):

- bare A1200 : 13.913-2.320*2 = 9.273
- A1200+Bz1230: 3.564-2.320*2 = -1.076

the first result is perfectly consistent, whereas the second is without doubt impossible! The simple reason is that on the expanded A1200 the screen to be filled is located in FAST ram, therefore the actual time is much less than 2.320*2 seconds (3.564-2.550 = 1.014 secs).

I must admit that 39 fps on a Bz1230-IV is not much, but I can’t really imagine a better way of implementing the routines which executes the conversion needed by FullRes displays; anyway, you *do* have a couple of ways to go faster: redraw only the areas that actually need to be updated or... reduce the display dimensions (currently a display size of 256x252 allows to reach 50+ fps)!!! ;)

We can close this paragraph with a note: the extra DMA fetch for MskPln and ChqrMode steals not so many CHIP ram bus cycles, so it isn’t worth turning those options off considering the extremely poor quality of the deriving screen modes (scrolling, instead, most of the times is of no use, so turn it OFF).

1.4 2.x.2 Memory Needs

```
2.x.2   Memory Needs



Are Tricky-Color screens memory-greedy?
Well, sort of.

First we have to discover how to calculate the quantity of memory required
for a screen ScrWd pixels wide and ScrHt pixels tall (each pixel is inten-
ded to be directly addressable):

 - in HalfRes mode we need to allocate several planes of size PlnSz =
   ScrWd*ScrHt bytes in CHIP ram; the number of planes (indicated from
   now on with PlnsNmb) is 4 if MskPln is OFF, 5 otherwise; the final oc-
   cupancy is therefore PlnSz*PlnsNmb bytes in CHIP ram, 0 bytes in FAST.
 - in FullRes mode we need to allocate a buffer in FAST ram (if available)
   of ScrSz = ScrWd*ScrHt bytes, plus 4 planes of DsplWd*4*DsplHT bytes
   in CHIP ram (planes in CHIP ram don't need to be as large as the buffer
   in FAST because its data have to be converted and then written to the
   CHIP planes - obviously we don't need to convert more data than the
   visible area can hold); additionally, for Blitter-assisted FullRes
   conversion we need one more buffer of ScrWd*ScrHt bytes in CHIP ram.

Let's make some comparisons between similar screens in different display
modes (DsplWd=320, DsplHt=256):

 A 256 colors normal planar screen requires:

  pixels   pixels   colors    bytes   lines   planes   bytes      mem type
   320   x  256   x  256   ->  40   x  256  x   8    = 81920        CHIP

 A TCS MskPln'ed HalfRes screen requires:

  pixels   pixels   colors    bytes   lines   planes   bytes      mem type
   160   x  256   x  256   ->  160  x  256  x   5    = 204800       CHIP

 A TCS FullRes screen with CPU-only FullRes conversion requires:

  pixels   pixels   colors    bytes   lines   planes   bytes      mem type
   320   x  256   x  256   ->  160  x  256  x   4    = 163840+      CHIP
                           ->  320  x  256  x   1    =  81920=      FAST
                                                       -------
                                                       245760

 A TCS FullRes screen with Blitter-assisted FullRes conversion requires:

  pixels   pixels   colors    bytes   lines   planes   bytes      mem type
   320   x  256   x  256   ->  160  x  256  x   4    = 163840+      CHIP
                           ->  320  x  256  x   1    =  81920+      CHIP
                           ->  320  x  256  x   1    =  81920=      FAST
                                                       -------
```

```
                                                         327680
```

So the answer to the question is, unfortunately, a big YES!
A screen that normally would occupy just 80 kb, in HalfRes takes 200 kb and in FullRes 240 or 320 kb!!!

But this is not all, there's an even worse thing to consider.
Think about screens in HalfRes mode larger than the display for a moment: in theory, it would be wise to reserve the memory needed only for the VdoPlns, whereas the SlcPlns and MskPln, whose use is limited to the visible (and smaller) display area, could be kept 160x256 bytes large (something similar does happen in FullRes, instead).
We're not so lucky.
This is not possible in practice because the SlcPlns and MskPlns share the BLPxMOD registers with the VdoPlns, due to the way we arranged them:

```
 plane 5   MskPln
 plane 4   SlcPln1
 plane 3   SlcPln0
 plane 2   VdoPln1
 plane 1   VdoPln0
```

It's spontaneous to say: "So what?!? Re-arrange them!!! SlcPlns can have their own horizontal size: it's enough to assign them to planes 2 and 4!

```
 plane 5   MskPln
 plane 4   SlcPln1
 plane 3   VdoPln1
 plane 2   SlcPln0
 plane 1   VdoPln0
```

See?!? Now VdoPlns belong to playfield 1 and SlcPlns to playfield 2, thus can be sized indipendently! We have to give up just on MskPln (unless activating a 6th DMA-hungry plane), but at least we've gained something!"

This would be a rather smart solution but... sigh! We are missing something here: VdoPln0 and VdoPln1 *must* belong to two different playfields for the TCS method is based on this!!! In fact, they must be shifted by one LORES pixel, that is equal to saying they need a different value in the two nibbles (each belonging to a different playfield) of BPLCON1's lowest byte.

- double and triple buffering obviously require 2 and 3 times as much as needed by the VdoPlns, respectively
- Cross Playfield requires two more planes in CHIP ram

1.5 2.y Some Considerations

2.y   Some Considerations

If we look at the speed, TCS surely offers a blistering fast way of plotting dots: in fact a single access per pixel is ideal to avoid the otherwise many CPU wait states due to Amiga’s CHIP ram bus architecture.
The real bad side is the horizontal resolution limitation in HalfRes mode, which can be overcome only with the expensive FullRes method that negatively influences the performance.

So far, maybe RGBH is the best method as it stands in the middle of RGBW/RGBM (pale/bright and imprecise) and RGB332 (dark but exact).
If I were to draft a general chart I’d scribble dowm:

pos	mode	unique colors	features
1st	RGBH	256	bright, quite smooth and varied colors
2nd	RGBM	217	bright, quite smooth and varied colors
3rd	RGBW	175	pale, quite smooth colors; ease of use
4th	RGBP	256	strong colors (no white)
5th	RGBS	256	very strong colors (no white)
6th	RGB332	256	dark, extra smooth colors; ease of use

The best choice, though, can be done only considering the picture(s) to display and their original palette(s) (in particular, if you don’t need bright colors, RGB332 is in absolute the best choice!)