Subject: A proposed design for big bitmaps (for Maiko Color (aka Kaleidoscope))
From: shih:mv:envos

Here's a proposed design for big bitmaps (for Maiko Color). This message is also filed as BigBitmaps.TEdit, under {Pogo:MV:envos}<DSUNLISP>Documents>Development>Color>, and {Eris}<lispcore>internal>doc>.

There are basically two alternatives, either change allocblock to allow blocks bigger than 65K, or to change bitblt (and perhaps many other functions) to allow a new datatype, BIGBM.

The following describes the second BIGBM alternative.

DESIGN MOTIVATION

The color code currently uses ALLOCPAGEBLOCK to create a non-GC'able large bitmap for the color screen. Windows on that screen, I believe point to that screen bitmap, but Window "backing bitmaps" (the thing that holds what is behind an open window, and what is inside a closed window), is a separate bitmap.

The problem is that the backing bitmap currently cannot be as big as the window would like it to be.

The BIGBM design would keep the noncollectible color screen bitmap, but would allow windows to have BIGBM backing bitmaps. In addition, this design is more likely to be portable backwards (e.g. into Medley1.0 or Medley1.1) since no existing system datatypes need to be changed (unlike the plan to change ALLOCBLOCK).

Since XAIE does not (I believe) allow (currently) DIG operations on closed windows, then there are no DIG operations (except BITBLT) which need to occur on BIGBM backing bitmaps.

Therefore, a BIGBM need merely be a GC'able datatype which supports BITBLT between itself and all other legal datatypes (principally windows and bitmaps, but possible streams (e.g. Interpress), and possible other datatypes).

Longer term, having a generalized BITBLT will be useful, for the following reasons:

Color - 8 bpp bitmaps will begin to push Medley's 32Mb address limitation, and 24 bpp bitmaps certainly will (the screen alone will take up 3Mb = 10% of the address space!).

Remote Bitmaps - there may be applications which need bitmaps outside of the address space (color above for example). Nick Brigg's Maple color processor for the 1186 had a generalized remote bitblt, which "did the right thing" when source and destination bitmaps were both remte (e.g. remote bitblt).

XWindow Medley- might require remote bitmaps.

NonSun Medley - might require remote bitmaps, especially because it is doubtful the SunOS system call MMAP exists elsewhere. When not, the screen itself will need to be remote.

BIGBM DESIGN

A straightforward design would be to have a BIGBM simply be a collection of ordinary bitmaps ("slices of the big bitmap"), and then generalize BITBLT to handle the collection (by repeated bitblts).

More elaborate designs would allow BIGBMs to be a collection of any BitBlt'able objects (streams, windows, bitmaps, bigbitmaps), or perhaps collections of raw blocks (to save some of the BITBLT initialization overhead).

Probably at least one dimension of the slices should be a multiple of the LCD (least common denominator) of any TEXTURE (currently 16), so that textures do not show seams across the slices.

Probably also the slices should be "short and fat" (e.g. the BIGBM is made up of several rows of bitmaps), since the ucode inner loop runs in the X direction, but this partially depends on the application.

For example, font bitmaps are short and *very* wide. If large color fonts need a BIGBM font bitmap, then each character would cut across several slices, slowing down each character. For fonts, column slices might be better. Note that each character is also more "coherent" in memory this way, which might improve paging behaviour.

Having BIGBMs be composite objects (rather than a large coherent allocblock) might also improve GC behaviour, because a BIGBM can be allocated out fragmented free space. There may not be enough coherent free space to allocate a large allocblock, due to fragmentation.

Another design elaboration would be to provide full Imagestream capabilities onto BIGBMs, to that (DSPCREATE dest) will work on BIGBMs.

BIGBM IMPLEMENTATION

There is currently a draft implementation of BIGBMs on {Eris}<lispcore>internal>library>BIGBM. It has several limitations:

0. It is intended to be a subfunction of BITBLT (which explains the limits 1 & 2 below)
1. It doesn't handle clipping (I'm not sure, but clipping might be handled generically in BITBLT by changing the srcex, srcey, destx, desty, width, & height args).
2. It doesn't handle default arguments (BitBlt does this).
3. It currently only makes the slices 16 bits tall (it should make the slices as big as possible).
4. Its not very efficient. It only walks the slices linearly (the first relevent slice could be found by algebra), and it doesn't stop when the last relevent slice is bltted.
5. Because of 3 & 4, for large bitmaps its roughly 2x slower than normal bitmap blts. For small bitmaps, it may be much worse. This may be OK though for backing color windows, because color windows are large.

Mostly, this code needs to be folded into BITBLT & BLTSHADE, at the appropriate spots, and optimized if necessary.