Chapter 11 QuickDraw™ & Graphics

From: Larry Rosenstein, Apple Computer, Inc.
Subject:Copybits and the colortable

In article <3084@ur-cc.UUCP> tonyg@merlin.cvs.rochester.edu (Tony Giaccone) writes:

> So if that's the case why doesn't the copybits install a copy of the
> current color table into the PICT2. Or, if it does how come PICTViewer
> doesn't use that Color Table when it displays the file? It certainly seems
> to use the color tables in other PICT2 files. I don't get it.

If you were to dump the picture, I think you would find that your custom color table was present. A pixmap always has an associated color table.

What happens is that the CopyBits from your picture to the screen follows the normal Color QuickDraw rules. That is, Color QuickDraw maps the desired colors to the closest ones that the display can show. Color QuickDraw never modifies the current color table in the process of drawing. If you were using a 24-bit display, then you would get the desired colors, because all colors are achievable.

This is a known weakness in the PICT format. What programs needs to do is get the "appropriate" color table for the picture, set up that palette, and then draw the picture. The problem is that there is no standard for getting the "appropriate" color table.

Some programs store the CLUT as a separate resource, and read that resource to construct the palette. (You could use the resource type understood by the Palette Manager.)

Another approach is to scan through the picture and build up a CLUT based on the colors actually used. For a bitmap image this is OK, since the CLUT is stored in the pixmap. For an arbitrary picture it is not as easy to do.

-
Larry Rosenstein, Apple Computer, Inc.,Object Specialist

●●●

From: Tom Dowdy
Subject: Piccomment and technote 175

I've watched lots of interesting postings on peoples theories on how to make this work, but I haven't yet seen what I consider the best answer.

In article <UZB_yoW00UzxQ1R4wY@andrew.cmu.edu> ba0k+@andrew.cmu.edu (Brian Patrick Arnold) writes:

> [Recommends use of :]
> "SetLineWidth" described in Tech Note #175.

This is a correct way to get thin lines from a LaserWriter. Note that this PicComment is only currently interpreted on the LaserWriter and you gain nothing on other printers.

BUT:
There is a way to get thin lines and device positioning from *all* printer drivers. This is to use the PrGeneral calls GetRslData and SetRsl, as documented in Tech Note 128. When you make these calls, your app is said to be "imaging at device resolution."

This means that within your print record you will see the page expressed in terms of the resolution that you specified. This means that you can then place/draw objects within the coordinate space of the device, rather than the coordinate space of 72 dpi, which is what normally happens.

The reason this approach is better is that owners of LaserWriter SCs, DeskWriters (and this is a growing number), ImageWriter LQs, and even lowly ImageWriters will be able to draw things to best possible output for their device. Everyone is happy! And you didn't have to write any PostScript, or mess with the Enlarge/Reduce box in the dialog, or even use PicComments.

Please note that you do not need to use these calls if you application is simply trying to get "smooth looking circles" from the LaserWriter. The drivers take care of that for you. You only need to make this call if you need to draw lines at device resolution, or you need to be able to place dots or objects within the coordinate space of the device.

If you'd like to make these calls, please refer to the Tech Note for information on when/how to call GetRslData and SetRsl. Those of you with Volume V of Inside Mac can also find the information there in the chapter on the Printing Manager, the information looks the same, although in general I tend check the Tech Notes first because they are updated more often.

Tom Dowdy Internet: dowdy@apple.COM

●●●

From:Dave ?
Subject:Sync Drawing (with Code)

To do synched drawing on a Mac II, you want to use a technique analogous to that of watching TickCount() on the MacPlus, but you have to make your own VBL counter. This should be done with VBLTask tied to the video slot so you really get video synching.

To illustrate this, here is some code (in Think C):

Dave

```
/*  my globals  */
extern long     VBLcounter;
extern short    videoSlot;
extern VBLTask  drawtask;

/*************************************************************

   Before we do anything, we have to know which slot the
   card is in.  Use this routine.  (See Start Manager in
   IM V.)

*/

FindVideoSlot()
{
   DefVideoRec defvidrec;

   GetVideoDefault(&defvidrec);
   videoSlot = defvidrec.sdSlot;
}

/*************************************************************

   Then we have to install a VBL Task.  This is the one
   I'll use.  To use global variables you must setup the
   A5 register.  To do this in MultiFinder requires
   a special trick, which is demonstrated with the
   CacheA5 function.

*/

pascal void ScreenDrawTask()
{
/*
*   WARNING! - this is self-modifying code, but it is
*   necessary to run under MultiFinder...see Tech Note 180
*/
   long    oldA5;

   asm {
       move.l  a5,oldA5
```

```
        move.l  #0,a5   ; this will be replaced with
                        ; CurrentA5 by the CacheA5 function at runtime
    }
```

```
   VBLcounter += 1;
   drawtask.vblCount = 1;
   asm {
       move.l  oldA5,a5
   }
}

pascal void CacheA5(addr)
long    addr;
/*
*   This function will place CurrentA5 into the beginning of task
*   functions which start as:
*   pascal void FooTask()
*   {
*       long    junk;
*
*       asm {
*           move.l  a5,junk
*           move.l  #0,a5   ;this will be replaced by CurrentA5
*           ....
*
*   See Tech Note #180
*/
{
       asm {
           move.l  addr,a0
           add.w   #2,a0   ; move past JMP instruction
           move.l  (a0),a0
           add.w   #0xa,a0 ; move past first part of function
           move.l  CurrentA5,(a0)
                           ; puts CurrentA5 in (*addr)() dc.l location
   }
}


/*****************************************************************

   This function actually sets up the VBL Task.

*/

StartCounter()
{
   CacheA5(ScreenDrawTask);
   VBLcounter = 0;
   drawtask.vblAddr = ScreenDrawTask;
   drawtask.qType = vType;
   drawtask.vblCount = 1;
   drawtask.vblPhase = 0;
   SlotVInstall(&drawtask,videoSlot);
}


/*****************************************************************

   This function removes the VBL Task.

*/

RemoveCounter()
{
   SlotVRemove(&drawtask,videoSlot);
}
```

```
/*****************************************************************
```

```
   This function holds everything until it's
   time to draw. Note that it could be replaced by
   a #define macro (make VBLjunk a global).

*/

WAIT_FOR_SCREEN()
{
   long VBLjunk;

   VBLjunk = VBLcounter;
   do {} while ( VBLcounter == VBLjunk );
   /* this is used to synch drawing to screen */
}

/*****************************************************************

   Now, putting this altogether into a program

*/


main()
{

   /* do your init stuff */

   FindVideoSlot();
   StartCounter();

   /* now, whenever you want to draw something, do this... */
   WAIT_FOR_SCREEN();
   DrawMyThing();

   /* when you're done...*/

   RemoveCounter();

}
```

●●●

From: oster@dewey.soe.berkeley.edu (David Phillip Oster)
Subject: THINK C & 32-bit quickdraw (interfaces for 32bit quickdraw)

As my way of saying thank you, here are the interfaces for 32-bit Quickdraw for THINK C. You may want to save this file and upload it,since it contains tabs and fairly long lines.:

--- David Phillip Oster -master of the ad hoc odd hack.

```
(cut here, call this QuickDraw32Bit.h).
/* let's get a real juicy 32 bit discussion going based on this maybe !!
  by bhamlin@well.sf.ca.us (Brian M. Hamlin)
  from oster@well.sf.ca.us (David Phillip Oster)
*/

/* I think that's it - check it out, write something fun. Remember: You
       don't have to have a 32bit video card, just Sys 6.0.3 or greater on
       a color-capable machine and the 32bit Quickdraw INIT in your system
       folder to use these calls !
```

```
*/
/* hoo-boy, well ANYWAY thanks to Sam Roberts for assembler tweaking. */
```

```
#ifndef _QuickDraw32Bit_
#define _QuickDraw32Bit_

#ifndef _Quickdraw_
#include <Quickdraw.h>
#endif

#ifndef _Color_
#include <Color.h>
#endif

/* New Constants for 32-Bit QuickDraw */

#define ditherCopy 64            /* Dither mode for Copybits */
#define RGBDirect 16             /* 16 & 32 bits/pixel pixelType value */


/* New error codes */

#define rgnOverflowErr -147                  /* Region accumulation failed. Resulting
region may be currupt */
#define pixmapTooDeepErr -148          /* Pixmap is not 1-bit/pixel for BitmapToRegion */
#define insufficientStackErr -149/* QuickDraw could not complete the operation */
#define cDepthErr -157                       /* invalid pixel depth passed to NewGWorld or
UpdateGWorld */


#define      pixPurgeMask        1
#define      noNewDeviceMask           2

/* Flag bits passed to or returned by Offscreen routines */

enum  {
       pixPurgeBit = 0,
       nowNewDeviceBit = 1,
       pixelsPurgeableBit = 6,
       pixelsLockedBit = 7,

       mapPixBit = 16,                 /* set if color table mapping occurred */
       newDepthBit = 17,         /* set if pixels were scaled to a different depth */
       alignPixBit = 18,         /* set if pixels were realigned to screen alignment */
       newRowBytesBit = 19,      /* set if pixmap was reconfigured in a new rowBytes */
       reallocPixBit = 20,       /* set if offscreen buffer had to be reallocated */
       clipPixBit = 28,          /* set if pixels were or are to be clipped */
       stretchPixBit = 29,       /* set if pixels were or are to be stretched/shrinked */
       ditherPixBit = 30,
       gwFlagErrBit = 31
};

typedef enum {
       pixPurge = 1L << pixPurgeBit,
       nowNewDevice = 1L << nowNewDeviceBit,
       pixelsPurgeable = 1L << pixelsPurgeableBit,
       pixelsLocked = 1L << pixelsLockedBit,
       mapPix = 1L << mapPixBit,
       newDepth = 1L << newDepthBit,
       alignPix = 1L << alignPixBit,
       newRowBytes = 1L << newRowBytesBit,
       reallocPix = 1L << reallocPixBit,
       clipPix = 1L << clipPixBit,
       stretchPix = 1L << stretchPixBit,
       ditherPix = 1L << ditherPixBit,
       gwFlagErr = 1L << gwFlagErrBit
```

```
}GWorldFlag;
```

```
typedef long GWorldFlags;

/* Type definition of a GWorldPtr */

typedef CGrafPtr GWorldPtr;


/* Function Prototypes (necessary to get automatic type coercion -- see LSC User's Manual
p.125)
*/
pascal OSErr  BitmapToRegion(RgnHandle, BitMap *);
pascal QDErr  NewGWorld(GWorldPtr *, short, Rect *, CTabHandle, GDHandle, GWorldFlags);
pascal Boolean      LockPixels(PixMapHandle);
pascal void         UnlockPixels(PixMapHandle);
pascal GWorldFlags  UpdateGWorld(GWorldPtr *, short, Rect *, CTabHandle, GDHandle,
GWorldFlags);
pascal void         DisposeGWorld(GWorldPtr);
pascal void         GetGWorld(CGrafPtr *, GDHandle *);
pascal void         SetGWorld(CGrafPtr, GDHandle);
pascal void         CTabChanged(CTabHandle);
pascal void         PixPatChanged(PixPatHandle);
pascal void         PortChanged(GrafPtr);
pascal void         GDeviceChanged(GDHandle);
pascal void         AllowPurgePixels(PixMapHandle);
pascal void         NoPurgePixels(PixMapHandle);
pascal GWorldFlags  GetPixelsState(PixMapHandle);
pascal void         SetPixelsState(PixMapHandle, GWorldFlags);
pascal Ptr          GetPixBaseAddr(PixMapHandle);
pascal QDErr  NewScreenBuffer(Rect *, Boolean, GDHandle *, PixMapHandle *);
pascal void         DisposeScreenBuffer(PixMapHandle);
pascal GDHandle     GetGWorldDevice(GWorldPtr);

#endif
```

●●●

From: oster@dewey.soe.berkeley.edu (David Phillip Oster)
Subject:Re: THINK C & 32-bit quickdraw (.c) (interfaces for 32bit quickdraw)

As my way of saying thank you, here are the interfaces for 32-bit Quickdraw for THINK C. You can save this, compile it, then get at all the 32-bit quickdraw traps from THINK C. You may want to save this file and upload it, since it contains tabs and fairly long lines.:

>--- David Phillip Oster -master of the ad hoc odd hack.

```
(cut here, call this QuickDraw32Bit.c).
#include <QuickDraw32Bit.h>

/* let's get a real juicy 32 bit discussion going based on this maybe !!
  by bhamlin@well.sf.ca.us (Brian M. Hamlin)
  from oster@well.sf.ca.us (David Phillip Oster)
*/

/* BitmapToRegion -
*/
pascal OSErr BitmapToRegion(region, bmap)
       RgnHandle    region;
       BitMap       *bmap;{

       asm {
              SUBQ.L #2,SP
```

```
MOVE.L region(A6),-(SP)
MOVE.L bmap(A6),-(SP)
DC.W   0xA8D7
MOVE.W (SP)+,D0
```

```
	}
}

/* NewGWorld -
*/
pascal QDErr NewGWorld(offscreenGWorld, pixelDepth, boundsRect, cTable, aGDevice, flags)
	GWorldPtr		*offscreenGWorld;
	short			pixelDepth;
	Rect			*boundsRect;
	CTabHandle		cTable;
	GDHandle		aGDevice;
	GWorldFlags		flags;{

	asm {
		SUBQ.L #2,SP
		MOVE.L offscreenGWorld(A6),-(SP)
		MOVE.W pixelDepth(A6),-(SP)
		MOVE.L boundsRect(A6),-(SP)
		MOVE.L cTable(A6),-(SP)
		MOVE.L aGDevice(A6),-(SP)
		MOVE.L flags(A6),-(SP)
		MOVEQ  #0,D0
		DC.W   0xAB1D
		MOVE.W (SP)+,D0
	}
}

/* LockPixels -
*/
pascal Boolean LockPixels(pm)PixMapHandle	pm;{
	asm {
		SUBQ.L #2,SP
		MOVE.L pm(A6),-(SP)
		MOVEQ  #1,D0
		DC.W   0xAB1D
		MOVE.B (SP)+,D0
	}
}

/* UnlockPixels -
*/
pascal void UnlockPixels(pm)PixMapHandle	pm;{
	asm {
		MOVE.L pm(A6),-(SP)
		MOVEQ  #2,D0
		DC.W   0xAB1D
	}
}

/* UpdateGWorld -
*/
pascal GWorldFlags UpdateGWorld(offscreenGWorld, pixelDepth, boundsRect, cTable, aGDevice,
flags)
	GWorldPtr		*offscreenGWorld;
	short			pixelDepth;
	Rect			*boundsRect;
	CTabHandle		cTable;
	GDHandle		aGDevice;
	GWorldFlags		flags;{

	asm {
		SUBQ.L #4,SP
		MOVE.L offscreenGWorld(A6),-(SP)
```

```
MOVE.W pixelDepth(A6),-(SP)
MOVE.L boundsRect(A6),-(SP)
MOVE.L cTable(A6),-(SP)
```

```
                MOVE.L aGDevice(A6),-(SP)
                MOVE.L flags(A6),-(SP)
                MOVEQ  #3,D0
                DC.W   0xAB1D
                MOVE.L (SP)+,D0
        }
}

/* DisposeGWorld -
*/
pascal void DisposeGWorld(offscreenGWorld)GWorldPtr offscreenGWorld;{
        asm {
                MOVE.L offscreenGWorld(A6),-(SP)
                MOVEQ  #4,D0
                DC.W   0xAB1D
        }
}

/* GetGWorld -
*/
pascal void GetGWorld(port, gdh)CGrafPtr *port;GDHandle *gdh;{
        asm {
                MOVE.L port(A6),-(SP)
                MOVE.L gdh(A6),-(SP)
                MOVEQ  #5,D0
                DC.W   0xAB1D
        }
}

/* SetGWorld -
*/
pascal void SetGWorld(port, gdh)CGrafPtr port;GDHandle gdh;{
        asm {
                MOVE.L port(A6),-(SP)
                MOVE.L gdh(A6),-(SP)
                MOVEQ  #6,D0
                DC.W   0xAB1D
        }
}

/* CTabChanged -
*/
pascal void CTabChanged(ctab)CTabHandle ctab;{
        asm {
                MOVE.L ctab(A6),-(SP)
                MOVEQ          #7,D0
                DC.W           0xAB1D
        }
}

/* PixPatChanged -
*/
pascal void PixPatChanged(ppat)PixPatHandle    ppat;{
        asm {
                MOVE.L ppat(A6),-(SP)
                MOVEQ  #8,D0
                DC.W   0xAB1D
        }
}

/* PortChanged -
*/
pascal void PortChanged(port)GrafPtr port;{
```

```
asm {
       MOVE.L port(A6),-(SP)
       MOVEQ  #9,D0
```

```
                DC.W   0xAB1D
        }
}

/* GDeviceChanged -
*/
pascal void GDeviceChanged(gdh)GDHandle gdh;{
        asm {
                MOVE.L gdh(A6),-(SP)
                MOVEQ  #0x0A,D0
                DC.W   0xAB1D
        }
}

/* AllowPurgePixels -
*/
pascal void AllowPurgePixels(pm)PixMapHandle pm;{
        asm {
                MOVE.L pm(A6),-(SP)
                MOVEQ  #0x0B,D0
                DC.W   0xAB1D
        }
}

/* NoPurgePixels -
*/
pascal void NoPurgePixels(pm)PixMapHandle pm;{
        asm {
                MOVE.L pm(A6),-(SP)
                MOVEQ  #0x0C,D0
                DC.W   0xAB1D
        }
}

/* GetPixelsState -
*/
pascal GWorldFlags GetPixelsState(pm)PixMapHandle pm;{
        asm {
                SUBQ.L #4,SP
                MOVE.L pm(A6),-(SP)
                MOVEQ  #0x0D,D0
                DC.W   0xAB1D
                MOVE.L (SP)+,D0
        }
}

/* SetPixelsState -
*/
pascal void SetPixelsState(pm, state)PixMapHandle pm;GWorldFlags state;{
        asm {
                MOVE.L pm(A6),-(SP)
                MOVE.L state(A6),-(SP)
                MOVEQ  #0x0E,D0
                DC.W   0xAB1D
        }
}

/* GetPixBaseAddr -
*/
pascal Ptr GetPixBaseAddr(pm)PixMapHandle pm;{
        asm {
                SUBQ.L #4,SP
                MOVE.L pm(A6),-(SP)
```

```
MOVEQ  #0x0F,D0
DC.W   0xAB1D
MOVE.L (SP)+,D0
```

```
	}
}

/* NewScreenBuffer -
*/
pascal QDErr NewScreenBuffer(globalRect, purgeable, gdh, offscreenPixMap)
	Rect		*globalRect;
	Boolean			purgeable;
	GDHandle	*gdh;
	PixMapHandle	*offscreenPixMap;{

	asm {
		SUBQ.L #2,SP
		MOVE.L globalRect(A6),-(SP)
		MOVE.B purgeable(A6),-(SP)
		MOVE.L gdh(A6),-(SP)
		MOVE.L offscreenPixMap(A6),-(SP)
		MOVEQ  #0x10,D0
		DC.W   0xAB1D
		MOVE.W (SP),D0
	}
}

/* DisposeScreenBuffer -
*/
pascal void DisposeScreenBuffer(offscreenPixMap)PixMapHandle offscreenPixMap;{
	asm {
		MOVE.L offscreenPixMap(A6),-(SP)
		MOVEQ  #0x11,D0
		DC.W   0xAB1D
	}
}

/* GetGWorldDevice -
*/
pascal GDHandle GetGWorldDevice(offscreenGWorld)GWorldPtr offscreenGWorld;{
	asm {
		SUBQ.L #4,SP
		MOVE.L offscreenGWorld(A6),-(SP)
		MOVEQ  #0x12,D0
		DC.W   0xAB1D
		MOVE.L (SP)+,D0
	}
}
```

●●●

From: oster@dewey.soe.berkeley.edu (David Phillip Oster)
Subject: Re: How do I get the slot of the main screen?

In article <334@6sigma.UUCP> blm@6sigma.UUCP (Brian Matthews) writes:

> >I can't figure out how to get the slot number of the main screen.

Re-read the graphic device chapter of Inside Mac Vol 5. Them write code similar to:

```
   GDHandle curgd;

   if(world->hasColorQD){
      for(curgd = GetDeviceList();curgd != NIL;curgd = GetNextDevice(curgd)){
         if(TestDeviceAttribute(curgd, screenDevice) &&
```

```
TestDeviceAttribute(curgd, screenActive) &&
TestDeviceAttribute(curgd, mainScreen)){
    return curgd;           /* found it */
}
```

```
            }
        }
    }
}
```

--- David Phillip Oster

●●●

From: austing@Apple.COM (Glenn L. Austin)
Subject:Re: Drawing to an off screen bitmap (with code)

In article <8912070531.AA07560@cadman.nyu.edu> deragon@CADMAN.NYU.EDU (John Deragon) writes:

> Hi folks,
> I did a small animation program on my Macintosh Plus,
>it steps through drawing each frame. The problem is, it is damn slow.
>So, faced with the fact that I cant afford a Mac IIcx, I would like to
>know how I could draw frames to an offScreen Bitmap. I am using LSC 4.0.

Drawing into an off-screen bitmap is really quite easy, if you follow a few rules:

1) The rowBytes *MUST* be even, otherwise the odd rows will be on an odd byte, just begging for address error!
2) Create a GrafPort for the new bitmap by calling OpenPort on a newly created GrafPort structure.
3) Save the current port and restore it after drawing to the off-screen port.

Some sample code (off the top of my head) in C:

```
BitMap  bm;  /* my off-screen bitmap */
GrafPort gp; /* my "off-screen grafport */
GrafPtr  oldPort; /* the current grafport */

GetPort(&oldPort); /* get the current grafport */
bm.rowBytes = 20; /* 160 pixels / 8 = 20 bytes */
SetRect(&bm.bounds, 0, 0, 160, 160); /* set the bounds (remember l,t,r,b!) */
bm.baseAddr = NewPtrClear(20 * 160); /* create a new ptr and zero the mem */
if (bm.baseAddr != nil)
{
  OpenPort(&gp); /* create a new, temporary grafport */
  SetPortBits(&bm); /* set the portBits to our bitmap */
  BlockMove((Ptr) &bm.bounds, (Ptr) &gp.portRect, sizeof(Rect)); /* copy the
           rect from bounds to portRect */
  RectRgn(gp.visRgn, &bm.bounds); /* change the visRgn to match */
  SetPort(&gp); /* use our port */

  /* Draw in the port just like you would draw on the screen! */

  SetPort(oldPort); /* restore the old port before disposing our ours! */
  ClosePort(&gp); /* so that the port memory is released *
  /* The bitmap is still around and valid and contains a bitmap of your */
  /* drawing */

  DisposPtr(bm.baseAddr); /* dispose of our off-screen bitmap memory */
}
```

It couldn't be easier! In fact, if you specify the port returned from the PrOpenPage with a SetPort, you are actually drawing into an off-screen bitmap that gets sent to the printer!

Glenn L. Austin

•••

From: oster@dewey.soe.berkeley.edu (David Phillip Oster)
Subject: Re: Color Quickdraw and copybits glitches (ps to colorqQD with Code)

Once again, here is a short example of using color quickdraw. This program takes a text file called "bush.ps" which is a postscript 16-gray level bitmap of president bush, reads it in and displays it in a window. This example is missing a few resources, and the actual "bush.ps" file.

```
/* ShowBush.c - short program to display a postscript picture of Bush.
     in color quickdraw. You also get Bush on your clipboard.
  by David Phillip Oster.

  Copy and use this in any way you see fit.

    Basically an example of the pain you have to go through to actually
    use color quickdraw.

    In old quickdraw, you had some data and a window, and you CopyBit()s the data to
       the window.

    In color quickdraw, you have as your source:
       a.) the data.
       b.) a color table.
       c.) a "pixmap" that connects the data to the color table.
    as your destination, you have:
       a.) a color window
       b.) a palette, that informs the window manager of which colors you'll be
           using, so the color manager can change the color table in the video
           board to make those color available.

    This program works best with at least 16 colors!

*/
#include <ColorToolbox.h>

#define HEIGHT 200
#define WIDTH  200

#define NIL 0L
#define NOT !

SysEnvRec *world;
Handle       myPixels;
WindowPtr    myWin;
PixMapHandle myPixMap;
CTabHandle      myCTable;
PaletteHandle myPm;

enum {
    READERROR = 1,
    NEEDSCOLORQD, /* this demo needs color quickdraw */
    NOPIXMAP,
    NOPIXELS,
    NOCTABLE,
    NOPALETTE,
    PIXDATACHANGED
};

enum {
    UNDOI = 1,
    COPYI = UNDOI + 1
};
```

```
main(){
    EventRecord myEvent;
```

```
    Init();
    for(;;){
        if(GetNextEvent(everyEvent, &myEvent)){
            HandleEvent(&myEvent);
        }
    }
}


Init(){
    PaletteHandle oldPm;

    FlushEvents(everyEvent, NIL);
    InitGraf( (Ptr) &thePort);
    InitFonts();
    InitWindows();
    InitCursor();
    InitDialogs(NIL);
    InitMenus();
    TEInit();

    world = (SysEnvRec *) NewPtr(sizeof(SysEnvRec));
    SysEnvirons(1, world);
    if( NOT world->hasColorQD){
        Error(NEEDSCOLORQD, "");
    }

    myWin = GetNewCWindow(128, NIL, -1L);                          /* set up the window */
    SetPort(myWin);
    ClipRect(&thePort->portRect);

    myPixMap = NewPixMap();                                              /* allocate the
pixmap */
    if(NIL == myPixMap){
        Error(NOPIXMAP, "");
    }
    myPixels = NewHandle( ((long) HEIGHT/2)*WIDTH);          /* allocate the pixel data */
    if(NIL == myPixels){
        Error(NOPIXELS, "");
    }                                                                 /* allocate the color
table */
    myCTable = (CTabHandle) NewHandle(sizeof(ColorTable) + 15L*sizeof(ColorSpec));
    if(NIL == myCTable){
        Error(NOCTABLE, "");
    }
    InitPixels(myPixels);                                    /* set up the pixel data */
    InitCTable(myCTable);                                    /* set up the color table */

    (**myPixMap).bounds.left = 0;                       /* set up the pixmap */
    (**myPixMap).bounds.top = 0;
    (**myPixMap).bounds.right = WIDTH;
    (**myPixMap).bounds.bottom = HEIGHT;
    (**myPixMap).rowBytes = 0x8000 | (WIDTH/2);
    if(NIL != (**myPixMap).pmTable)
        DisposHandle((**myPixMap).pmTable);
    (**myPixMap).pmTable = myCTable;
    (**myPixMap).pixelSize = 4;

    myPm = NewPalette(16, myCTable, pmTolerant, 0);    /* set up the palette */
    if(NIL == myPm){
        Error(NOPALETTE, "");
    }
```

```
if(NIL != (oldPm = GetPalette(thePort))){
```

```
        DisposePalette(oldPm);
    }
    SetPalette(thePort, myPm, TRUE);
    ActivatePalette(thePort);
}

/* SwapMove - copy from src to dest, moving scan lines from top to bottom
    as you go (postscript images are upside down compared to quickdraw images)
*/
SwapMove(src, dest, cnt)Ptr src, dest;long cnt;{
    register short i, j;

    for(i = 0;i <200;i++){
        BlockMove(src + (i*(WIDTH/2)), dest  + ((199-i)*(WIDTH/2)), WIDTH/2);
    }
}

/* InitPixels- initialize our handle to the bush data
    SwapMove since Postscript is upside down from Mac.
*/
InitPixels(h)Handle h;{
    register Ptr p, pMax;
    register short i;
    Handle h2, GetBush();

    h2 = GetBush();
    SwapMove(*h2, *h, GetHandleSize(h));
}


/* InitCTable - initialize our color table to uniform gray
*/
InitCTable(ct)CTabHandle ct;{
    short i;

    (**ct).ctSeed = GetCTSeed();
    (**ct).ctFlags = 0;
    (**ct).ctSize  = 15;
    for(i = 0;i < 16;i++){
        (**ct).ctTable[i].value = i;
        (**ct).ctTable[i].rgb.red =
        (**ct).ctTable[i].rgb.green =
        (**ct).ctTable[i].rgb.blue  = (unsigned) i * 16*257;
    }
}

/* Error - print the error message
*/
Error(i, s1)short i;Str255 s1;{
    Str255 s;

    GetIndString(s, 128, i);
    ParamText(s, s1, "", "");
    Alert(129, NIL);
    ExitToShell();
}

/* HandleEvent - quit on mouse or key event
*/
HandleEvent(event)EventRecord *event;{
    switch(event->what){
    case mouseDown:
    case keyDown:ExitToShell();
```

```
    case updateEvt:  GoUpdate(event);    break;
    }
}
```

```
/* GoUpdate - handle update event, paste it to clipboard
*/
GoUpdate(event)EventRecord *event;{
    SetPort(event->message);
    BeginUpdate(thePort);
    CopyMyBits();
    EndUpdate(thePort);
}

/* CopyMyBits - put bits to screen, put bits to clipboard
*/
CopyMyBits(){
    Rect r, r2;
    PicHandle ph;

    HLock(myPixels);
    HLock(myPixMap);
    (**myPixMap).baseAddr = *myPixels;
    r.left = 0;
    r.top = 0;
    r.right = WIDTH;
    r.bottom = HEIGHT;
    r2 = r;
    r2.top *= 2;
    r2.left *= 2;
    r2.right *= 2;
    r2.bottom *= 2;

    ph = OpenPicture(&r);
    CopyBits(*myPixMap, &thePort->portBits, &r, &r, srcCopy, NIL);
    ClosePicture();
    CopyBits(*myPixMap, &thePort->portBits, &r, &r2, srcCopy, NIL);

    HUnlock(myPixMap);
    HUnlock(myPixels);

    ZeroScrap();
    HLock(ph);
    PutScrap(GetHandleSize(ph), 'PICT', *ph);
    SystemEdit(COPYI-1);    /* so multifinder will take the scrap */
    KillPicture(ph);
}

/* GetBush
*/
Handle GetBush(){
    OSErr  val;
    short  ref;
    Handle h;
    long   len;
    Str255 s;

    ref = 0;
    h = NIL;
    val = FSOpen("\pbush.ps", 0, &ref);
    if(noErr == val)val = GetEOF(ref, &len);
    if(noErr == val)h = NewHandle(len - 4);
    if(noErr == val && NIL != h){
        HLock(h);
        len = GetHandleSize(h);
        if(noErr == val)val = FSRead(ref, &len, *h);
        HexToBin(h);
```

```
    HUnlock(h);
}
if(0 != ref){
```

```
        FSClose(ref);
    }
    if(noErr != val){
        NumToString(val, s);
        Error(READERROR, s);
        if(NIL != h){
            DisposHandle(h);
            h = NIL;
        }
    }
    return h;
}

/* SkipHeader - advance p to point to data
*/
unsigned char *SkipHeader(p)register unsigned char *p;{
    while( NOT (p[0] == '\r' && p[1] == '0'))
        p++;

    return &p[1];
}

/* HexToBin - h is a locked handle containing text. turn it into binary
    Sorry about this. It is a quick and dirty hex ascii to binary routine.
*/
HexToBin(h)Handle h;{
    register unsigned char *src, *srcMax, *dest;

    src  = dest = (unsigned char *) *h;
    src = SkipHeader(src);
    srcMax = src + GetHandleSize(h);
    while(src < srcMax){
        if(*src >= '0' && *src <= '9'){
            *dest = (*src++ - '0') << 4;

            if(*src >= '0' && *src <= '9'){
                *dest++ |= *src++ - '0';
            }else if(*src >= 'A' && *src <= 'F'){
                *dest++ |= *src++ - 'A' + 10;
            }else if(*src >= 'a' && *src <= 'f'){
                *dest++ |= *src++ - 'a' + 10;
            }
        }else if(*src >= 'A' && *src <= 'F'){
            *dest = (*src++ - 'A' + 10) << 4;

            if(*src >= '0' && *src <= '9'){
                *dest++ |= *src++ - '0';
            }else if(*src >= 'A' && *src <= 'F'){
                *dest++ |= *src++ - 'A' + 10;
            }else if(*src >= 'a' && *src <= 'f'){
                *dest++ |= *src++ - 'a' + 10;
            }
        }else if(*src >= 'a' && *src <= 'f'){
            *dest = (*src++ - 'a' + 10) << 4;

            if(*src >= '0' && *src <= '9'){
                *dest++ |= *src++ - '0';
            }else if(*src >= 'A' && *src <= 'F'){
                *dest++ |= *src++ - 'A' + 10;
            }else if(*src >= 'a' && *src <= 'f'){
                *dest++ |= *src++ - 'a' + 10;
            }
        }else{
```

```
            src++;
        }
    }
```

```
    SetHandleSize(h, dest - (unsigned char *) *h);
}
```

●●●

From: oster@dewey.soe.berkeley.edu (David Phillip Oster)
Subject: Re: Finding the size of the screen under Multifinder/Color QD (gdevices)

In article <10139@saturn.ucsc.edu> sirkm@ssyx.ucsc.edu (Greg Anderson) writes:

> >TN #117 says to look at the QuickDraw variable screenbits.bounds to
> >find the size of the screen, but on my Mac IIx, this rectangle is
> >always (0,0,0,0) when MultiFinder is installed (color or b&w).

Tim Maroney is right, it should return the bounding rectangle of the CRT that has the menu bar. It always has for me.

> >First question: what _is_ a gDevice? Is there one & only one gDevice
> >per monitor (/other display device) hooked up to the Mac, or might there
> >be multiple devices referencing a single monitor? If the later, why?

There should only be one at a time. Some video boards change the number of pixels they display depending on what mode they are in: i.e, they might be 640x480 in 8-bit per pixel mode, by 1024x1204 withe one bit per pixel. In addition, since graphic devices hold the inverse color table data structures (used for color matching when you do a CopyBits of a pixmap with one color table to a pixmap with a different table) there may be graphic devices that aren't connected to screens.

> >The reason I wish to know the size of the screen is simply to center my
> >dialogs on the screen. Given this purpose, should I just look at one
> >gDevice (namely, the one returned by GetGDevice)? Once I have this handle,
> >how should I determine the screen bounds? Get the PixMap from
> >(*gDev)->gdPMap and then look at pixMap.bounds?

use screenBits.bounds.

Here _is_ a use for gDevices:

I redefine the zoom box, so that windows zoom to the full size of the CRT they are on, rather than Apple's default. There is nothing more annoying than working with a two-monitor system, clicking on the zoom box of a window, and have it leap to the other, smaller montior.

Here is a code fragment:

```
/* GetBiggestZoom - set r to the dimensions of the largest
   extern SysEnvRec world;

   biggdh = NIL;
   if(world.hasColorQD){
       for(gdh = GetDeviceList(); gdh != NIL; gdh = GetNextDevice(gdh)){
           if(TestDeviceAttribute(gdh, screenDevice) &&
               TestDeviceAttribute(gdh, scrnActive)){
               if(biggdh == NIL || (**gdh).gdRect.bottom - (**gdh).gdRect.top >
                           (**biggdh).gdRect.bottom - (**biggdh).gdRect.top){
                   biggdh = gdh;
               }
           }
       }
   }
   if(biggdh == NIL){
       *r = screenBits.bounds;
       r->top += GetMBarHeight();
   }else{
       *r = (**biggdh).gdRect;
```

```
if(TestDeviceAttribute(biggdh, mainScrn)){
```

```
            r->top += GetMBarHeight();
        }
    }
}
```

--- David Phillip Oster

●●●

From: trebor@biar.UUCP (Robert J Woodhead)
Subject: Re: Bitmap not showing up in scrapbook (cliprect)

legg@sirius.ucs.adelaide.edu.au (Christian Legg) writes:

> A friend of mine is having problems with displaying a bitmap that he has
>saved in the scrapbook. He uses copybits to copy the bitmap into a PICT
>resource which he then saves into the scrapbook. The scrapbook indicates
>in the lower right hand corner that the item is a PICT, but does not display
>it.

Thou hast been bitten by that most egregious of monsters, the Immense ClipRect. When you create the grafport that you draw into to generate the picture, always set it's cliprect to just surround the area that will be drawn. Otherwise you get a huge cliprect (-32768,-32768,32767,32767) and the scrapbook and other programs go a little bonkers because of it.

--
Robert J Woodhead, Biar Games, Inc. !uunet!biar!trebor | trebor@biar.UUCP

●●●

From: tim@hoptoad.uucp (Tim Maroney)
Subject: Re: **Finding the size of the screen under Multifinder/Color QD**

In article <10139@saturn.ucsc.edu> sirkm@ssyx.ucsc.edu (Greg Anderson) writes:

>>>TN #117 says to look at the QuickDraw variable screenbits.bounds to
>>>find the size of the screen, but on my Mac IIx, this rectangle is
>>>always (0,0,0,0) when MultiFinder is installed (color or b&w).

In article <9429@hoptoad.uucp> tim@hoptoad.UUCP (Tim Maroney) writes:
>>Uh, I really don't think so. Check again.

In article <1102@urbana.mcd.mot.com> willcox@urbana.mcd.mot.com
(David A Willcox) writes:
>This sounds a bit like what happened when I tried to use screenBits
>from within an XCMD. The problem, of course, was that I hadn't called
>InitGraf() (I hope that's the right name) from within my XCMD, and I
>was seeing a local copy of screenBits that had not been initialized.
>(I'm using Think C 3.0.)

When I first read this, I though, "Of course!" But now, I'm not sure this makes sense. The XCMD should be called using HyperCard's A5, and its QD globals are initialized already. Maybe the problem is that LSC has a bug that looks for the QD globals off A4 if you are linking a code resource? That wouldn't be too surprising. But the LSC code resource glue doesn't mess about with the caller's A5, which should remain constant throughout.

>I tried calling InitGraf() within my XCMD, and that gave me valid
>values for screenBits, but things rapidly went to hell on return to
>HyperCard, since HyperCard's quickdraw globals didn't work any more.
>(Obvious, in hindsight.)

Sort of. This seems to be compatible with the A4-bug hypothesis,assuming that the code generated was using offsets from

A4 and you passed a pointer to thePort which was also A4-relative.

>Can anyone tell me, is there a better way to get at global variables

>such as screenBits from within an XCMD?

Tell me, did you try the ugly way of accessing QD globals from assembly language, from IM I-163? I bet it would work.
--
Tim Maroney, Mac Software Consultant, sun!hoptoad!tim, tim@toad.com

●●●

From: oster@dewey.soe.berkeley.edu (David Phillip Oster)
Subject: Re: Finding the size of the screen under Multifinder/Color QD (with C code)

In article <10164@saturn.ucsc.edu> sirkm@ssyx.ucsc.edu (Greg Anderson) writes:

>I know it's illegal to call InitGraf() from within XCMD's, but can I
>call it from within a DA?

No, you can't call it from a DA either.

>If I cannot call InitGraf() from within my DA, then how can I find the
>size of the screen?

Here is how you access the quickdraw globals from a non-application, assuming that _some_ application has set them up by calling InitGraf: (A safe assumption for D.A.s, CDEFs, MDEFs, WDEFs, CDEVs, & FKEYs)

```
/* QuickDraw - access to globals from a code resource
       brought to you by David Phillip Oster
       (oster@well.sf.ca.us  or oster@dewey.soe.berkeley.edu)
       You may use this as you see fit.
*/
typedef struct QuickDraw {
       LongInt      randSeed;
       BitMap screenBits;
       Cursor arrow;
       Pattern      dkGray;
       Pattern      ltgray;
       Pattern      gray;
       Pattern      black;
       Pattern      white;
       GrafPtr      thePort;
} QuickDraw;


/* DisableSelf - this is from a CDEF I wrote, it dims the control.
*/
DisableSelf(ch)ControlHandle ch;{
       QuickDraw *qd;
       Rect r;

       qd = (QuickDraw *) ( *(Byte **) CurrentA5 - (sizeof(QuickDraw) -
sizeof(GrafPtr)) );
       r = (**ch).contrlRect;
       PenMode(patBic);
       PenPat(qd->gray);   /* <-- Note, a QD global! */
       PaintRect(&r);
       PenNormal();
}
```

David Phillip Oster

●●●

From: tim@hoptoad.uucp (Tim Maroney)

Subject: **Re: Marquee Help? (with Code)**

In article <18304@dartvax.Dartmouth.EDU> sean@eleazar.dartmouth.edu (Sean P. Nolan) writes:

> >I saw on here recently a passing mention that someone was using code that
> >produced the MacPaint-like Marquee around a selection. The Johnny Carson theme
> >comes to mind. But I have been trying, off-and-on, to figure out how to do
> >just that, given a selection rectangle and a nifty slot to do its thing in
> >my Event loop.
> >
> >If anybody has the ah-ha insight on how to do this, I'd really appreciate a
> >note. Currently I shoved the problem aside and just InvertRect the selection,
> >which is uuuuuuuugly.

Might as well just post dwb's code that I got from a local BBS:

```
;
; (c) Copyright 1987, David W. Berry
;
              case      object
              string    pascal

              load   'MacStuff'

              export    MarqueeRect, MoveMarqueeRect

patterns      record    entry,decr
marqueePat       dc.l      $0f87c3e1, $f0783c1e
shiftPat      dc.l      $88442211, $88442211
              endr

;
;   Frame rectangle, using marquee pattern.  The rect is drawn in xor mode.
;   This procedure should be called to draw the marquee initially, then
;   MoveMarqueeRect should be called repeatedly to move the pattern.
;   Finally, call MarqueeRect again to erase the marquee.  The display
;   will end up the same as it was before.
;

MarqueeRect      proc      export

Frame         record    {a6link},decr
rect      ds.l      1
return        ds.l      1
a6link        ds.l      1
state         ds.b      psRec
locals        equ       *
              endr

              with      Frame

              link      a6,#locals
              pea       state(a6)
              _GetPenState

              _PenNormal

              pea       patterns.marqueePat
              _PenPat

              move.w    #patXor,-(a7)
              _PenMode
```

```
move.l    rect(a6),-(a7)
_FrameRect
```

```
            pea       state(a6)
            _SetPenState

            unlk      a6
            rts

            endproc

;
;   Frame rectangle, using pattern that moves the marquee.  Then shift
;   the pattern and the marquee pattern, to keep the two in sync.
;   This procedure must be called repeatedly to achieve the effect of
;   motion.
;

MoveMarqueeRect  proc   export

Frame       record {a6link},decr
rect      ds.l   1
return      ds.l   1
a6link      ds.l   1
state       ds.b   psRec
locals      equ    *
            endr

            with   Frame

            link   a6,#locals

            pea    state(a6)
            _GetPenState

            _PenNormal

            pea    patterns.shiftPat
            _PenPat

            move   #patXor,-(a7)
            _PenMode

            move.l rect(a6),-(a7)
            _FrameRect

            pea    state(a6)
            _SetPenState

    ; shift the marquee pattern 1 slot to the left
            lea    patterns.marqueePat,a0
            bsr.s  rotate

    ; likewise the shift pattern
            lea    patterns.shiftPat,a0
            bsr.s  rotate

            unlk   a6
            rts

rotate:             move.l          (a0),d1
            move.l    4(a0),d0
            rol.l     #8,d0
            rol.l     #8,d1
            move.b    d0,d2
            move.b    d1,d0
```

```
move.b    d2,d1
move.l    d1,(a0)
move.l    d0,4(a0)
```

```
            rts

            endproc

            end
```

End of quoted code marquee.a.

Here's MPW C code I use for the same effect. I'm afraid it's rather environment-dependent, but you should still be able to break it to your will.

```
static void RotatePatterns(Pattern marquee, Pattern shift)
{   unsigned char c;
    c = marquee[0];
    BlockMove(marquee + 1, marquee, 7);
    marquee[7] = c;
    c = shift[0];
    BlockMove(shift + 1, shift, 7);
    shift[7] = c;
}

static void
click(WindowPtr window, DocStorage **storage, EventRecord *event,
     Boolean *changed, Boolean *selected)
{   Point start, now; Rect r, r2; short tmp;
    DocInfo doc; Pattern marquee, shift; PenState pen;
#pragma unused (changed)
    unselect(window, storage);
    GetDocInfo(window, &doc);
    start = now = event->where;
    GetPenState(&pen);
    PenNormal();
    BlockMove((Ptr)(*storage)->marquee, (Ptr)marquee, 8);
    BlockMove((Ptr)(*storage)->shift, (Ptr)shift, 8);
    PenPat(marquee);
    PenMode(patXor);
    SetRect(&r, start.h, start.v, now.h, now.v);
    FrameRect(&r);
    while (WaitMouseUp()) {
        PenPat(shift);
        FrameRect(&r);
        RotatePatterns(marquee, shift);
        GetMouse(&now);
        SetRect(&r2, start.h, start.v, now.h, now.v);
        if (r2.left > r2.right)
            { tmp = r2.left; r2.left = r2.right; r2.right = tmp; }
        if (r2.top > r2.bottom)
            { tmp = r2.top; r2.top = r2.bottom; r2.bottom = tmp; }
        if (EqualRect(&r, &r2)) continue;
        PenPat(marquee);
        FrameRect(&r);
        FrameRect(&r2);
        r = r2;
    }
    if (*selected = !EmptyRect(&r))
        OffsetRect(&r, GetCtlValue(doc.hScroll), GetCtlValue(doc.vScroll));
    else { PenPat(marquee); FrameRect(&r); SetRect(&r, 0, 0, 0, 0); }
    BlockMove((Ptr)marquee, (Ptr)(*storage)->marquee, 8);
    BlockMove((Ptr)shift, (Ptr)(*storage)->shift, 8);
    (*storage)->selRect = r;
    SetPenState(&pen);
```

```
}

static void idle(WindowPtr window, DocStorage **storage)
{   Pattern marquee, shift; Rect r; PenState pen; DocInfo doc;
```

```
    r = (*storage)->selRect;
    if (EmptyRect(&r)) return;
    GetDocInfo(window, &doc);
    GetPenState(&pen);
    PenNormal();
    BlockMove((Ptr)(*storage)->marquee, (Ptr)marquee, 8);
    BlockMove((Ptr)(*storage)->shift, (Ptr)shift, 8);
    PenPat(shift);
    PenMode(patXor);
    OffsetRect(&r, -GetCtlValue(doc.hScroll), -GetCtlValue(doc.vScroll));
    FrameRect(&r);
    RotatePatterns(marquee, shift);
    BlockMove((Ptr)marquee, (Ptr)(*storage)->marquee, 8);
    BlockMove((Ptr)shift, (Ptr)(*storage)->shift, 8);
    SetPenState(&pen);
}
```

And in Rez, the patterns are:

```
resource 'PAT#' (33) { {
       $"0f 87 c3 e1 f0 78 3c 1e" ,
       $"88 44 22 11 88 44 22 11"
} };
```

--
Tim Maroney, Mac Software Consultant, sun!hoptoad!tim, tim@toad.com

●●●

From: ari@eleazar.dartmouth.edu (Ari Halberstadt)
Subject: How to get the QD globals outside of an application

Several people have been curious about how to access the QuickDraw globals from within an XCMD or XFCN. Here's a little routine I wrote for THINK C which you can call when you initialize your XCMD or XFCN. If I remember correctly, MPW uses a single global structure to store the QuickDraw globals. Thus, one BlockMove would suffice to implement this routine in MPW.

```
/* QuickDraw global offsets off of A5 */
#define THEPORT     0            /* GrafPtr */
#define WHITE       THEPORT-8    /* Pattern */
#define BLACK       WHITE-8            /* Pattern */
#define GRAY        BLACK-8            /* Pattern */
#define LTGRAY         GRAY-8          /* Pattern */
#define DKGRAY         LTGRAY-8  /* Pattern */
#define ARROW       DKGRAY-68    /* Cursor */
#define SCREENBITS     ARROW-14  /* BitMap */
#define RANDSEED    SCREENBITS-4 /* long */

/* Setup the QuickDraw globals by getting them from their offsets off of A5 */
/* By Ari Halberstadt, 1990 */
static void
SetupQDGlobals()
{
    Ptr p;

    p = *((Ptr *) CurrentA5);
    thePort = (GrafPtr) (THEPORT + p);
    BlockMove(WHITE + p, white, sizeof(Pattern));
    BlockMove(BLACK + p, black, sizeof(Pattern));
    BlockMove(GRAY + p, gray, sizeof(Pattern));
    BlockMove(LTGRAY + p, ltGray, sizeof(Pattern));
    BlockMove(DKGRAY + p, dkGray, sizeof(Pattern));
    BlockMove(ARROW + p, &arrow, sizeof(Cursor));
```

```
    BlockMove(SCREENBITS + p, &screenBits, sizeof(BitMap));
    randSeed = (long) (RANDSEED + p); /*I hope this line is correct...*/
}
```

--
Ari Halberstadt

●●●

From: lsr@Apple.COM (Larry Rosenstein)
Subject: Re: Marquee Help?

```
>PenMode(patXor);          /* I'm not sure this is either the correct
```

If you are doing this in a bitmap, then you should use patCopy. patXOR might look better in a structured graphics program, where there is always a lot of white space. (If you use patCopy then to erase the marquee you have to redraw the contents of the rectangle.0

```
>PenPat(patterns[patNum++]);
>patNum &= 3;
>FrameRect(pRect);                    /* Draw a rectangle */
>while (TickCount() - t < 4) /* Wait a bit */
```

Another way to do this is to choose the pattern based on the value of TickCount % 4 (or TickCount % 8 if you use 8 patterns). The advantage is that the ants will move at the same speed regardless of the size of the rectangle. The disadvantage is that you may get jerky motion if your loop can't keep up.

--
Larry Rosenstein, Object Specialist

●●●

From: jmunkki@kampi.hut.fi (Juri Munkki)
Subject: Re: Finding the size of the screen under Multifinder/Color QD

In article <9530@hoptoad.uucp> tim@hoptoad.UUCP (Tim Maroney) writes:

```
>Nope.  The OpenPort strategy doesn't work on color systems, at least
>not in the simple way you seem to be suggesting.  If you just look at
>port.portBits.bounds, you'll be looking at something weird (a handle
>followed by two integers) on color systems.
```

Nope. You are confusing OpenCPort and OpenPort. OpenPort opens a regular grafport with a normal bitmap. I don't think that you bothered to look, but I did and even if I set my Mac II to 8 color, I get a rowBytes of 128 and a bounds of (0,0,480,640).

Creating a pixmap for offscreen grafports would have created an enormous amount of incompatibility. OpenPort fakes a bitmap that looks just as if you had a 1-bit screen. That's why the old MacPaint and SuperPaint painted all over the top of the screen. They got a rowBytes of 128 when it actually is 8 times that much. (Although the screen is 640 pixels wide, Apple throws away a few bytes and uses 1024 pixels per row of which only 640 are displayed.)

Juri Munkki jmunkki@hut.fi jmunkki@fingate.bitnet

●●●

From: beard@ux1.lbl.gov (Patrick C Beard)
Subject: Re: Finding the size of the screen under Multifinder/Color QD

In article <9530@hoptoad.uucp> tim@hoptoad.UUCP (Tim Maroney) writes:

```
#Nope.  The OpenPort strategy doesn't work on color systems, at least
#not in the simple way you seem to be suggesting.  If you just look at
```

```
#port.portBits.bounds, you'll be looking at something weird (a handle
#followed by two integers) on color systems.
#
#           if ((port->portBits.rowBytes & 0x8000) == 0)
#                       screen = port->portBits.bounds;
```

```
#       else {    PixMapHandle pm = ((CGrafPtr) port)->portPixMap;
#                 screen = (*pm)->bounds;
#       }
```

I've used the OpenPort strategy before and it does work. In fact, if you just open a classic GrafPort, you don't need to go through the tests you show above. I have also just used the portRect field of the GrafPort to give me the same information:

```
ScreenSize(Rect *r)
{
    GrafPort aPort;
    OpenPort(&aPort);
    *r = aPort.portRect;
    ClosePort(&aPort);
}
```

Another method is to just use the Window Manager port. Instead of the OpenPort call above, and instead of allocating a GrafPort on the stack you only need a GrafPtr.

```
ScreenSize(Rect *r)
{
    GrafPtr aPort;

    GetWMgrPort(&aPort);
    *r = aPort->portRect;
}
```

This seems to work on all systems I've tried.

Patrick Beard, Macintosh Programmer (beard@lbl.gov)

●●●

From: ech@cbnewsk.ATT.COM (Ned Horvath)
Subject: Re: Finding the size of the screen under Multifinder/Color QD

From article <10164@saturn.ucsc.edu>, by sirkm@ssyx.ucsc.edu (Greg Anderson):

> [quoted references to invalid screenbits.bounds deleted]
>
> I know it's illegal to call InitGraf() from within XCMD's, but can I
> call it from within a DA? It seems like I would have the same problems
> you experienced.
>
> If I cannot call InitGraf() from within my DA, then how can I find the
> size of the screen?
>
> >>Can anyone tell me, is there a better way to get at global variables
> >>such as screenBits from within an XCMD?
>
> Sort of -- make callbacks to HyperCard. This won't always get you what you
> want, but it can often be very helpful...

Assuming that the A5 you "inherit" is valid, and that InitGraf has been called by your host application (which had pretty well ALWAYS be true!), there are a couple of answers.

1. GrafPort myPort; OpenPort (&myPort);...your code here...ClosePort(&myPort); This gives you access to some of the variables you crave.

2. More generally, the low memory global CurrentA5 (long integer at 0x904) contains the address of the address of thePort (the last quickdraw variable) for the current application. So the following code snippet will get you a way to PEEK (never, never, NEVER poke!) at the QD globals:

```
struct qdvars {         /* note that the order is the reverse */
   long   randSeed; /* of that on IM I-204 */
   BitMap screenBits;
```

```
    Cursor arror;
    Pattern   dkGray;
    Pattern   ltGray;
    Pattern   gray;
    Pattern   black;
    Pattern   white;
    GrafPtr   thePort;
} *myGlobals;

    myGlobals = (struct qdvars *) (
        (**(long**)0x904) -             /* address of thePort */
        (long)&((struct qdvars *)0)->thePort  /* offset of thePort */
    );
```

I presume that if you're not using C you can figure out how to translate that nasty stuff into your own argot. The ugly part with 0x904 gets the address of thePort; the even more bizarre second term evaluates to a constant (at compile time) which is the offset of thePort in the full set of Quickdraw globals. You can now reference myGlobals->screenBits.bounds, for example.

The second example works whenever the host application has called InitGraf(), even at interrupt time; most APPLs call InitGraf within micro seconds after launch, so you should be safe.

=Ned Horvath=

●●●

From: oster@dewey.soe.berkeley.edu (David Phillip Oster)
Subject:Re: PICT resource to BitMap (with code)

In article <oZkwPji00WB5E_IkpG@andrew.cmu.edu> es2q+@andrew.cmu.edu (Erik Warren Selberg) writes:
>I'm having loads of problems trying to figure out how to grab a PICT
>resource, turn it into a bitmap, and then use CopyBits with it (actually, my
>problem is that I can't seem to turn said resource into a bitmap). Does

Write it yourself, it'll improve your mac programming skills, and there isno single system call to do it anyway. You'll do basically

```
BitMap *PictToBitMap(myPic)PicHandle myPic;{
    GrafPtr      savePort, myPort;
    GrafPort  myPortRec;
    static BitMap myBitMap; /* so we can return it */
    Rect      r;

    GetPort(&savePort);
    myPort = &myPortRec;
    OpenPort(myPort);
    SetPort(myPort);
    myBitMap.bounds.top = myBitMap.bounds.left = 0;
    myBitMap.bounds.right = BIGENOUGHFORTHIS;
    myBitMap.bounds.bottom = ALSOBIGENOUGH;
    /* must be big enough an even. */
    myBitMap.rowBytes = ((myBitMap.bounds.right + 15) / 16) * 2);
    myBitMap.baseAddr = NewPtr(myBitMap.rowBytes * (long) myBitMap.bounds.bottom);
    SetPortBits(&myBitMap);
    /* next 2 lines necessary is PICT bigger than Mac's screen. */
    ClipRect(&myPort->portRect);
    CopyRgn(myPort->clipRgn, myPort->visRgn);

    r = (**myPic).picFrame;/* 'cause it might move during draw */
    DrawPicture(myPic, &r);
    SetPort(savePort);
```

```
    ClosePort(myPort);
    return &myBitMap;
}
```

The above is from memory, and may have errors in it. The caller is responsible for deallocating my BitMap.baseAddr.

Handling color PICTs is left as an exercise to the reader. Oh, the above will _work_, but if you want the color table of the returned pixmap to be minimal (i.e., only as deep as necessary, with only the colors that are actually used by the PICT.) you either need to 1.) allocate a 32-bit deep grafport, draw the picture into that, then search its pixmap to see what colors it used, if it used less than 256 colors, but more than 16, build an 8 bit deep grafport, with an appropriate color table, and CopyBits from the 32-bit deep one to the 8-bit one, discard the 32-bit one, and return the 8. or 2.)

Write a parser for PICTs, search it for color tables and 16-bit or 32-bit deep pixmaps, and figure out the final color table yourself. This has the advantage that, at the cost of re-implementing color quickdraw yourself, it can be made to work even on a MacPlus, and the disadvantage that it is likely to break the next time Apple changes quickdraw. I would go with method 1, or just blow off the whole question and handle all color pixmaps as 32-bit deep.

David Phillip Oster

●●●

From: rcfische@polyslo.CalPoly.EDU (Raymond C. Fischer)
Subject: Re: Writing large PICT files???

In article <22163.25d040f6@kuhub.cc.ukans.edu> brownrigg@kuhub.cc.ukans.edu writes:
>Can someone describe for me how to create a PICT file consisting of only
>a bitMap?

This is from memory, so check the calls against Inside Mac.
Assuming the existence of a GrafPort that has your BitMap in it ...

```
SetPort (your GrafPtr);
OpenPicture
CopyBits (thePort^.portBits, thePort^.portBits,
     thePort^.portRect, thePort^.portRect,
     srcCopy, NIL);
ClosePicture
```

This will create a PICT that has nothing but the bitmap in it. To turn this into a PICT file, create a file of type PICT, write 512 bytes of zeros, then write entire handle contents returned from the OpenPicture/ClosePicture.

Make sure that at some time after opening the port and BEFORE calling OpenPicture, you set the ClipRgn of the port. One way to do this is

```
ClipRect (thePort^.portRect);
```

If you have only a BitMap and no GrafPort, then open a GrafPort, set the portBits to the bitmap record (using SetPortBits?), set the portRect (as in: thePort^.portRect := thePort^.portBits.bounds), and set the ClipRgn. The do the above CopyBits and then close the port;

If the GrafPort is a CGrafPort, this will produce a type 2 PICT (color) which won't work with any Mac that doesn't have color quickdraw.

Of course, GrafPort is synonymous with WindowRecord in this example.
Any questions?

Ray Fischer rcfische@polyslo.calpoly.edu

●●●

From: ephraim@think.com (Ephraim Vishniac)
Subject: Re: How's the screen cleaned after _SysError?

In article <6959@internal.Apple.COM> lsr@Apple.COM (Larry Rosenstein) writes:
>In article <34252@news.Think.COM> ephraim@Think.COM (Ephraim Vishniac)

> >writes:
> >> How is the screen restored after a call to _SysError?
>
> >There's a low memory global called DSAlertRect. In GetMouse, if the high
> >bit of DSWndUpdate is 1, then the system refreshes the rectangle stored in
> >DSAlertRect.

GetMouse doesn't do this on my Mac II (6.02, Multifinder off). I disassembled it and found that the only subroutine call was to a journaling routine which exits immedidately if journaling is off.

Also, DSWndUpdate seems to be $FF most of the time (i.e., the high bit is always set).

> >Since this doesn't seem to be documented, use at your own risk.

Not documented? Not so! Please refer to "Low Memory in Alphabetical Order" dated 12 Apr 85. (You did keep every scrap of paper from the Macintosh Software Supplements, right?) DSWndUpdate is at least as well documented as the famous MrMacHook. Notice the line that reads:

DSWndUpdate .EQU $15D ;01 GNE not to paintBehind DS AlertRect? [byte]

This suggests that it's GetNextEvent, not GetMouse, and painting will occur when the bit is cleared, not set. I haven't actually tried this out yet - it took me a while to find the right documentation.

--
Ephraim Vishniac ephraim@think.com ThinkingCorp@applelink.apple.com

●●●

From: jackiw@cs.swarthmore.edu (Nick Jackiw)
Subject: Optimizing Copybits (good Information with code)

[This is a repost of some comments of mine that appeared here last month.]

In Inside Mac V and in various places in the techNotes, it states that CopyBits is optimized for cases in which rowBytes is a multiple of four and in which the bitmaps are aligned. I recently did some testing on a (wide) sample of arbitrary bitmaps, and came up with the following interesting timings:

- Worst case ranges from about 102-105% of average case.
  (Which is to say: the average case is about as bad as you can get)
- Best case (perfect alignment) ranges from about 54-72% of average case.
  (i. e. Alignment has a *substantial* payoff.)

Remember, alignment (keeping the bit-index the same between corresponding bits in the source bitmap and the destination) is only useful under several conditions:

- You're always copying part of the source into one and only one corresponding part of the destination.
- The source and destination are at the same bitdepth, if you're working in color.

- For any given copy, the srcRect and destRect are of the same dimension (though not necessarily the same coordinates).

A good example of when alignment is useful: storing an offscreen bitmapcontaining the image to be painted into a window. Alignment will only change (and the offscreen map be recomputed) if the window is moved.

A good example of when alignment won't help at all: storing the image of a spaceship offscreen which can be copyBitted anywhere into the window. ("Anywhere"=any of 32 bit positions=alignment will only help you in 1/32 of your copies. Of course, you could store 32 differently-aligned spaceship copies offscreen, and choose the appropriate one to blit based on the dest rect.)

The following function might be useful for cases when you want to create an offscreen bitmap for copying to a window. Specify the window (which must already be created and in its proper location) in destWindow, and the bounds of the desired offscreen window (in the local coordinate system of destWindow). InitAlignedBitmap will generate the bitmap, tacking enough "slack" onto the left edge of the bitmap (i. e. by extending the bounds) to guarantee that any bit in theBits is aligned with the corresponding bit in the window's local coordinate system. This will guarantee you best case performance.

Lastly, a note or two:

- If User moves the window, you'll need to recompute and redraw the bitmap. You can dispose the old one and call this function again, if you like, or you can modify this one to SetPtrSize on the existing baseAddr ptr. You may want to change the code to generate handles, instead of ptrs, but remember to lock and derefence them before drawing. Growing a window doesn't change the alignment (e. g. local [0,0] is still in the same place as it was pre-grow; therefore so is offscreen [0,0].)

- Color doesn't complicate things (multiply by pixDepth before determining rowBytes, and don't forget to set the high bit), but multiple screens and multiple pixel depths do. In that you can only be in alignment with one of the multiple screens at a time (or only *guarantee* that you'll be so), you'll have to choose between aligning for the screen which is deepest (which you'll want to do if your offscreen bitmap should be complexly colored), or for the screen which has the maximum destWindow area on it (which will optimize for speed over color fidelity). Once,you've figured out which GDevice to align to, plug in its pixMap for screenBits, below.

Enjoy.

```
function InitAlignedBitMap(var theBits:Bitmap;destWindow:WindowPtr):boolean;

{By Nick Jackiw}
{Allocates storage for a bitmap the bounds of which have been set before}
{calling.  The bitmap will be long-word aligned with the coordinate system}
{of the bit image which contains the destWindow.}

 var
  oldPort: GrafPtr;
  aPoint: point;
  bitOffset: integer;  {# of bits we need to expand the offscreen map by}

begin
 GetPort(oldPort);     {Switch to desired port, for local->global conversion}
 SetPort(destWindow);
 aPoint := theBits.bounds.topLeft;
 LocalToGlobal(aPoint);

 bitOffset := (aPoint.h - ScreenBits.bounds.left) mod 32;
 if bitOffset < 0 then
  bitOffset := 32 + bitOffset;

 with theBits.bounds do
  left := left - bitOffset;{Expand dest bitmap by appropriate # of bits}

 with theBits do
  with bounds do
   begin
    rowBytes := ((right - left + 31) div 32) * 4;
    baseAddr := NewPtr(rowBytes * (bottom - top));
    InitAlignedBitMap := baseAddr <> nil
   end;
 SetPort(oldPort)
end;
```

--

Nick Jackiw | Visual Geometry Project | Math Department

●●●

From: jackiw@cs.swarthmore.edu (Nick Jackiw)
Subject: Re: Fast Copybits suggestions needed, other questions

chris@ADMS-RAD.Unisys.COM (Chris Sterritt) writes:

> Someone a few weeks ago posted quite a long and informative post
> on making CopyBits go really fast. I didn't keep it then (in my almost
> infinite stupidity :-), so I'd like a copy now.

Other e-mail requests suggest to me that a repost would be welcome.
I'll post it separately.

> In case it doesn't answer these questions, I'll ask them now:
> (Background: I'm working on a game with animation, that I'd like
> to go as fast as possible).
>
> The game is (for now) black and white, will CopyBits do the
> right thing if it's run on a 4 or more bits deep screen (i.e., copying a
> 1-bit bitmap)? Or should I special-case the code for color?

Special-case. For every B&W pixel you draw on a four-bit screen, Color Quick- draw is going to have to convert it into a 4-bit pixel, and map it to the appropriate color. Though these might seem trivial, multiplied by a zillion pixels they amount to an incredible delay.

The cheesy thing to do would be to insist that user run in B&W mode. Better would be, at start-up, to allocate an offscreen pixmap at the current depth of the screen, draw all of your b&w bitmaps onto it, and then dispose the 1-bit bitmaps. Then use the curdepth pixmap as your library in the actual game. This keeps the task of translating depths isolated in your initialization code, not duplicated every time you draw an object onscreen.

--
Nick Jackiw

●●●

From:-Kelesi
Subject:How does one set up an offscreen buffer

>How does one set up an offscreen buffer so that CopyBits may be used to
>refresh the window when responding to an update event?

This is really a rather simple procedure. The idea is to draw into an offscreen port and then CopyBits the wanted information to the desired window. To setup an offscreen port for drawing, you can do something like this...

In Pascal (LSP):

```
Function SetupPort : GrafPtr;
var
 tmpPort, myPort : GrafPtr;
begin
 GetPort(tmpPort);
   { Need to allocate memory for the port }
 myPort := GrafPtr(NewPtr(sizeof(GrafPort));
   { -- Error checking for nil pointer goes here -- }
   { Create the internal structures for the port }
 OpenPort(myPort);
   { Now we need to set aside enough memory to draw into }
```

```
with myPort^.portBits do
  begin
    baseAddr := NewPtr(rowBytes * (bounds.bottom - bounds.top));
      { -- Error checking for nil pointer goes here -- }
```

```
        { Don't need to initialize any of the rest of the fields in }
        { the port because Quickdraw has done all we need already. }
    end;
 { And return the resulting port, voila, an offscreen graph port! }
 SetPort(tmpPort);
 SetupPort := myPort;
end;
```

or, for them (LS)C buffs...

```
#define myBM myPort->portBits
GrafPtr SetupPort()
{
    GrafPtr tmpPort;
    register GrafPtr myPort;

    GetPort(&tmpPort);
    myPort = (GrafPtr) NewPtr(sizeof(GrafPort));
        /* -- Error checking for nil pointer goes here -- */
        /* Create the internal structures for the port */
    OpenPort(myPort);
        /* Now we need to set aside enough memory to draw into */
    myBM.baseAddr = NewPtr(myBM.rowBytes * (myBM.bounds.bottom -
        myBM.bounds.top));
        /* -- Error checking for nil pointer goes here -- */
        /* And return the resulting port, voila, an offscreen graph port! */
    SetPort(tmpPort);
    return myPort;
}
#undef myBM
```

There are several problems with the program as it stands above. 1) it does not have any error conditions. 2) It eats up about 6K of the heap everytime it's called (or more if your doing color, but if that's the case, you'll _need_ to look at Macintosh Technote #120)

Notice that I saved the current port on entry, this is because I didn't really want SetupPort to change my current graph port. If you don't care, you can remove these lines (but you may never know when you may need them, I spent several hours tracking down a single missing SetPort once...not a pleasant experience.)

You can save memory by passing a window to the procedure and then resizing the port to the window size. If you do this though, you have to resize the port everytime you resize the window, something of a bother. Enough of this, let's describe how to draw, etc.

The procedure to draw into the newly created port may look something like this (in Pascal):

```
Procedure DrawInMyPort( port : GrafPtr );
var
 tmpPort : GrafPtr;
begin
 GetPort(tmpPort);
 SetPort(port);
   { various Quickdraw calls here }
 SetPort(tmpPort);
end;
```

And in C...

```
DrawInMyPort(port)
GrafPtr port;
{
    GrafPtr tmpPort;
```

```
GetPort(&tmpPort);
SetPort(port);
    /* various Quickdraw calls here */
SetPort(tmpPort);
```

```
}
```

Pretty easy huh? The way the port is setup, you have the entire 'screen' to draw into, we'd have to do a few more calls if the window are 'larger' than the screen (e.g. paint programs). Finally, we want to do an update event, drawing from the offscreen port to a window:

```
Procedure UpdateWind( wind : WindowPtr; port : GrafPtr );
var
 tmpPort;
begin
 GetPort(tmpPort);
 SetPort(wind);
 BeginUpdate(wind);
 CopyBits(port^.portBits, wind^.portBits, port^.portRect,
   wind^.portRect, srcCopy, nil);
 EndUpdate(wind);
 SetPort(tmpPort);
end;
```

or

```
#define NIL (0L)
UpdateWind(wind, port);
WindowPtr wind;
GrafPtr port;
{
   GrafPtr tmpPort;

   GetPort(&tmpPort);
   SetPort(wind);
   BeginUpdate(wind);
   CopyBits(&port->portBits, &wind->portBits, &port->portRect,
       &wind->portRect, srcCopy, NIL);
   EndUpdate(wind);
   SetPort(tmpPort);
}
```

These are all pretty much basic routines here but they should work. If you have any more questions, you might refer to the Macintosh technical notes. (You might be able to find the most recently distributed tn's on comp.mac.sources (comp.sources.mac?)) Specifically, TN#120 (which I know was uploaded recently) Hope this helps!

-Kelesi

●●●

From:Written by John Olsen to go CCW, optimized by Mike Morton, and unknown
Subject: Rotate Bit Map CCW (full source code)

```
int RotateBMapCCW(srcBMap, dstBMap)
   BitMap *srcBMap, *dstBMap;
/* Given a source and destination bitMap, rotate it CCW 90°.
* Bitmap can be of any size, will also align on word boundaries.
* Written by John Olsen, optimized by Mike Morton, adapted to C by me.
*
* This routine has a little problem. The width of the destBMap is based
* on its rowBytes, not its bounds rect, for speed reasons… but this
* pretty much guarantees garbage in the no-man's land between the
* bounds.right and the rowBytes. This causes a problem when counting
* pixels in the caller routine, so the caller must take pains to
* reclear this area.
*
```

```
* Returns FALSE if memory allocation problems, else returns TRUE.
*
* See MacTutor, V.4-N.11-p.86 for more details.
*/
```

```
{   int    colmCount;
    long   currentWord;
    long   lowerLeft;
    int    wordCount;

    asm
    {  movem.l   d3-d7/a2-a5,-(sp)          ; save registers

       ; Initialize destination bitMap, etc.
       ;   a0 = result of _NewPtr call         | d0 = size of bitMap allocate
       ;  a1 = ptr to srcBMap                | d1 = width / 2 (source)
       ;  a2 = ptr to dstBMap                   | d2 = height / 2 (source)
       ;  a3 = dest bits                        | d3 = left | top
       ;  a4 = copy of dest bits                | d4 = ctrPt.h
       ;  a5 = not used                      | d5 = ctrPt.v

       move.l srcBMap(a6),a1                    ; point to the source BitMap
       move.l dstBMap(a6),a2                    ; point to the destination BitMap

       ; Compute height of source BitMap rounded up to nearest word.
       move.w bds + bo(a1),d7               ; bottom into d7
       sub.w     bds + to(a1),d7            ; bottom - top = height, put in d7
       move.w d7,d0                         ; save copy of height for later

       ; rnd = (height + 15) / 16, follows:
       addi.w #15,d7                        ; add #15 to height, put in d7
       lsr       #4,d7                      ; divide by 16
       lsl       #1,d7                      ; Multiply by 2 = rowBytes of dstBMap

       ; d7 now contains rowBytes of dstBMap. Now compute the dstRect for
       ;  dstBMap.
       ;
       ;  Compute the center point of srcRect, note the center points are
       ;  not necessarily the same in the vertical direction relative to
       ;  srcBMap becuz of rounding that follows, (i.e., height of srcBMap
       ;  is directly proportional to rowBytes of dstBMap).
       lsr       #1,d0                      ; divide height by 2
       move.w d0,d2                         ; make a copy of height/2 for later
       move.w bds + to(a1),d5               ; get 'top' of srcRect
       add.w     d0,d5                      ; d5 contains vert compnt of ctr pt

       ; Compute the width of srcRect & put in d4
       move.w bds + ri(a1),d3               ; get right and put in d3
       sub.w     bds + le(a1),d3            ; right - left = width, put in d3
       lsr       #1,d3                      ; divide width by 2
       move.w d3,d1                         ; make a copy of width/2 for later
       move.w bds + ri(a1),d4               ; get right of srcRect
       sub.w     d3,d4                      ; d4 contains horiz compnt ctr pt

       ; Now get center point and compute bounds for dstBMap.
       ;  Height and width are reversed now for the 2 BitMaps.
       ;
       ;     left      := ctrPt.h - height / 2;
       ;     top       := ctrPt.v - width / 2;
       ;  right     := ctrPt.h + height / 2;
       ;     bottom := ctrPt.v + width / 2;
       ;
       ;  Put data in direct to avoid overhead of SetRect.
       ;  Compute 'top'.
       move.w d5,d3                             ; get a copy of ctrPt.v and put in d3
       sub.w     d1,d3                          ; ctrPt.v - width / 2 = 'top' in d3
       swap      d3                             ; move 'top' to hiword
```

```
; Compute 'left'.
move.w d4,d3                            ; get a copy of ctrPt.h & put in loword d3
```

```
        sub.w     d2,d3                         ; ctrPt.h - height / 2 = 'left' in loword d3

        ; We now have 'top' in hiword & 'left' in loword d3.
        ;  Compute 'bottom'.
        move.w d5,d6                            ; get a copy of ctrPt.v in d3
        add.w     d1,d6                         ; ctrPt.v + width / 2 = 'bottom' in d3
        swap      d6                            ; move 'bottom' to hiword

        ; Compute 'right'.
        move.w d4,d6                            ; get a copy of ctrPt.h and put in loword d6
        add.w     d2,d6
        ; We now have 'bottom' in hiword & 'right' in loword d6.
        ;
        ; Assign data to dstBMap structure directly.
        move.w d7,rB(a2)                  ; put in rowBytes of dstBMap
        move.l d3,bds + topLOff(a2)          ; put the coord pair in direct
        move.l d6,bds + botROff(a2)          ; put the coord pair in direct
        move.w d7,d0                      ; put rowBytes in d0
        mulu      rB(a1),d0               ; multiply rowBytes src*dst = d0
        lsl       #3,d0                   ; multiply d0 by 8 for size of dstBMap
        _NewPtr CLEAR                     ; allocate size of dstBMap (a2 pts to it)
        cmp.l     #0,a0                   ; succesful?
        bne       @cont                   ; yes, continue
        clr.l     d0                      ; no, return FALSE & …
        return;                              ; …exit
cont:
        move.l a0,bA(a2)                  ; move a0 to baseAddr field of dstBMap

        ;-------------------------------------------------------------------------------
        ; a0 = not used                       | d0 = rowBytes of srcBMap
        ; a1 = points to bits of srcBMap      | d1 = width / 2 (source)
        ; a2 = ptr to dstBMap                     | d2 = height / 2 (source)
        ; a3 = dest Bits                       | d3 = current word scrBMap
        ; a4 =                                    | d4 = copy of width of srcBMap
        ; a5 = not used                       | d5 = current dstBMap
        ; a6 = not used                       | d6 = insideLoop count (current bit)
        ; a7 = not used                       | d7 = rowBytes of dstBMap
        ;-------------------------------------------------------------------------------
        ; ****************************************************************
        ; *
        ; *       Start of actual rotation of bitMaps
        ; *
        ; ****************************************************************
        ; Assumptions:
        ; height of destination bitMap = rowBytes of srcBMap
        ;

        ; initialize d0 to contain the pointer to the srcBMap
        move.w rB(a1),d0                  ; get rowBytes of srcBMap

        ; get regs pointing to bits
        move.l (a1),a1                          ; a1 pts to BITS of srcBMap
        move.l (a2),a3                          ; a3 pts to BITS of dstBMap

        ; Compute the lower left corner of the dstBMap - this maps to
        ; the upper left corner of the srcBMap and will be used as base
        ; from which to offset into the dstBMap when we need to set a bit.
        clr.l     d1                      ; clear d1
        move.w d0,d1                      ; load rowBytes of srcBMap
                                          ; (rowBytes of srcBMap = ht of dstBMap)
        lsl.w     #3,d1                   ; multiply by 8 to get ht of dstBmap when rot'd
        mulu.w d7,d1                      ; multiply by rowBytes(dstBMap) = #words in
        sub.w     d7,d1                   ; dstBMap less one row
```

```
       add.l     a3,d1                      ; add the computed offset to the baseAddr of
dstBMap
       move.l d1,lowerLeft                  ; init lower left

       ;---------------------------------------------------------------------------
       ;   a0 = not used                    |   d0 = rowbytes of srcMap
       ;   a1 = points to bits of src bitMap|   d1 = SCRATCH
       ;   a2 = currentWord                 |   d2 = colmCounter
       ;   a3 = dest bits                      |   d3 = current srcMap word
       ;   a4 = SCRATCH                     |   d4 = colmBitLoop counter
       ;   a5 = not used                    |   d5 = current word destMap
       ;   a6 = not used                    |   d6 = insideLoop count (current bit)
       ;   a7 = not used                    |   d7 = rowBytes of dest bitMap
       ;---------------------------------------------------------------------------
       ;

       ; init wordCount to zero
       ;
       move.w #0,wordCount              ; set wordCount to one before we start
       ;
       ; init colmLoop counter
       ;
       move.w d7,d2                     ; get rowBytes(dest)
       lsr       #1,d2                  ; divide by 2 - d2 now contains the outer counter
       move.w d2,colmCount              ; put the max value away for reference
       moveq.l   #0,d2                     ; set d2 to zero since we use it for the counter

colmLoop:
       ;
       ; this is the outside loop
       ;
       move.w #15,d4                    ; init colmBitLoop counter to 15
       move.l lowerLeft,a2              ; a2 is currentWord, init it

colmBitLoop:
       ;
       ; this is the loop that does the bit counting in the destMap - it is really a
       ; misuse of the 'dbra' instruction since it is not an independent loop but it
       ; is a quite cheap method of doing our counting
       ;

       cmp.w     #7,d4                  ; are we doing high byte or low?
       bgt       @doByte1               ; skip if high byte
       addq      #1,a2                  ; else point to low byte

doByte1:
rowLoop:

       ;
       ; loop across each row in the srcMap and keep count each time we do a row
       ; this 'rowCount' will used to compute the offset from 'lowerLeft'
       ;
       move.w (a1)+,d3                  ; get a word to rotate into d3
       addq.w #2,wordCount              ; add 1 for each word ---jdo---

       add.w     d3,d3                  ; test next bit in the current source word
       bcc.s     @noBitToChange            ; if bit's not set, skip to next
       bset      d4,(a2)                   ; set proper bit in high or low byte in
currentWord

noBitToChange:
       sub.l     d7,a2                  ; subtract rowBytes(destMap) from current word
```

```
        ; Do the same as above, fifteen more times.
```

```
       add.w     d3,d3
       bcc.s     @1
       bset      d4,(a2)
@1  sub.l     d7,a2

       add.w     d3,d3
       bcc.s     @2
       bset      d4,(a2)
@2  sub.l     d7,a2

       add.w     d3,d3
       bcc.s     @3
       bset      d4,(a2)
@3  sub.l     d7,a2

       add.w     d3,d3
       bcc.s     @4
       bset      d4,(a2)
@4  sub.l     d7,a2

       add.w     d3,d3
       bcc.s     @5
       bset      d4,(a2)
@5  sub.l     d7,a2

       add.w     d3,d3
       bcc.s     @6
       bset      d4,(a2)
@6  sub.l     d7,a2

       add.w     d3,d3
       bcc.s     @7
       bset      d4,(a2)
@7  sub.l     d7,a2

       add.w     d3,d3
       bcc.s     @8
       bset      d4,(a2)
@8  sub.l     d7,a2

       add.w     d3,d3
       bcc.s     @9
       bset      d4,(a2)
@9  sub.l     d7,a2

       add.w     d3,d3
       bcc.s     @10
       bset      d4,(a2)
@10 sub.l     d7,a2

       add.w     d3,d3
       bcc.s     @11
       bset      d4,(a2)
@11 sub.l     d7,a2

       add.w     d3,d3
       bcc.s     @12
       bset      d4,(a2)
@12 sub.l     d7,a2

       add.w     d3,d3
       bcc.s     @13
       bset      d4,(a2)
```

```
@13 sub.l     d7,a2

       add.w     d3,d3
```

```
       bcc.s     @14
       bset      d4,(a2)
@14 sub.l     d7,a2

       add.w     d3,d3
       bcc.s     @15
       bset      d4,(a2)
@15 sub.l     d7,a2

       ;
       ; at this point we've just completed a word from the srcMap
       ;
       cmp       wordCount,d0           ; cmp wordCount to rowBytes-are we done with a row ?
       bne       @rowLoop               ; no - do another word

       cmp.w     #7,d4                  ; did we do high byte or low?
       bgt       @dobyte2               ; skip if high byte
       subq      #1,a2                  ; else undo the fudge to a2
dobyte2:

       ;
       ; we just completed a row so add 2 to the count
       ;
       move.w #0,wordCount              ; ---jdo--- reset wordCount to zero

       ;
       ; the following use of dbra keeps track of our destMap bit for us
       ;
       move.l lowerLeft,a2              ; re-init currentWord
                                        ; set the currentWord to the lowerLeft(current)
                                        ; lowerLeft is moved over (right) by a word each
                                                        ; time we do a row in the srcMap
       dbra      d4,@colmBitLoop        ; go thru the bits (15->0) in the
                                        ; destMap - then exit and do it
                                           ; again for rowBytes/2 times

       ;
       ; were done counting thru the bits of a destMap word - at this point we have
       ; actually finished a column in the destMap. The column is a multiple of
       ; 16 bits wide (and being at least 1 colm) and as tall as the destMap is
       ; (which is also equal to rowBytes of the srcMap)
       ;
       addq.l #2,lowerLeft           ; this moves us into the next
                                     ; 'column' of bits in the destMap
       addq.w #1,d2                  ; add 1 to the colmLoop counter
       cmp       colmCount,d2        ; are we done yet ?
       bne       @colmLoop           ; no - do it again
                                     ; else we're done

       move.l #1,d0                           ; return TRUE, all ok
       movem.l (sp)+,d3-d7/a2-a5        ; restore registers, bye!
   }
} /* RotateBMapCCW */


int RotateBMapCW(srcBMap, dstBMap)
   BitMap *srcBMap, *dstBMap;
/* Given a source and destination bitMap, rotate it CW 90°.
* Bitmap can be of any size, will also align on word boundaries.
* Written by John Olsen to go CCW, optimized by Mike Morton,
* and adapted to C by me.
*
* Revised 2/26/89 by me to do CW rotation.
```

*

```
* This routine has a little problem. The width of the destBMap is based
* on its rowBytes, not its bounds rect, for speed reasons… but this
* pretty much guarantees garbage in the no-man's land between the
* bounds.right and the rowBytes. This causes a problem when counting
* pixels in the caller routine, so the caller must take pains to
* reclear this area.
*
* Returns FALSE if memory allocation problems, else returns TRUE.
*
* See MacTutor, V.4-N.11-p.86 for more details.
*/
{   int    colmCount;
    long   currentWord;
    long   lowerLeft;
    int    wordCount;
    int    srcHeight;    /* pixel height of source bitMap */

    asm
    {   movem.l   d3-d7/a2-a5,-(sp)          ; save registers

        ; Initialize destination bitMap, etc.
        ;    a0 = result of _NewPtr call      | d0 = size of bitMap allocate
        ;   a1 = ptr to srcBMap                  | d1 = width / 2 (source)
        ;   a2 = ptr to dstBMap             | d2 = height / 2 (source)
        ;   a3 = dest bits                       | d3 = left | top
        ;   a4 = copy of dest bits               | d4 = ctrPt.h
        ;   a5 = not used                           | d5 = ctrPt.v

        move.l srcBMap(a6),a1             ; point to the source BitMap
        move.l dstBMap(a6),a2             ; point to the destination BitMap

        ; Compute height of source BitMap rounded up to nearest word.
        move.w bds + bo(a1),d7               ; bottom into d7
        sub.w     bds + to(a1),d7            ; bottom - top = height, put in d7
        move.w d7,d0                         ; save copy of height for later

        ; rnd = (height + 15) / 16, follows:
        addi.w #15,d7                        ; add #15 to height, put in d7
        lsr       #4,d7                      ; divide by 16
        lsl       #1,d7                      ; Multiply by 2 = rowBytes of dstBMap

        ; d7 now contains rowBytes of dstBMap. Now compute the dstRect for
        ;  dstBMap.
        ;
        ;  Compute the center point of srcRect, note the center points are
        ;  not necessarily the same in the vertical direction relative to
        ;  srcBMap becuz of rounding that follows, (i.e., height of srcBMap
        ;  is directly proportional to rowBytes of dstBMap).
        lsr       #1,d0                   ; divide height by 2
        move.w d0,d2                      ; make a copy of height/2 for later
        move.w bds + to(a1),d5            ; get 'top' of srcRect
        add.w     d0,d5                   ; d5 contains vert compnt of ctr pt

        ; Compute the height of srcRect & put in srcHeight - rcy 2/26/89
        move.w bds + bo(a1),d3            ; get bottom and put in d3
        sub.w     bds + to(a1),d3         ; bottom - top = height, put in d3
        move.w d3,srcHeight                  ; store here, need later…

        ; Compute the width of srcRect & put in d4
        move.w bds + ri(a1),d3               ; get right and put in d3
        sub.w     bds + le(a1),d3            ; right - left = width, put in d3
        lsr       #1,d3                      ; divide width by 2
        move.w d3,d1                         ; make a copy of width/2 for later
```

```
move.w bds + ri(a1),d4              ; get right of srcRect
sub.w     d3,d4                     ; d4 contains horiz compnt ctr pt
```

```
        ; Now get center point and compute bounds for dstBMap.
        ;  Height and width are reversed now for the 2 BitMaps.
        ;
        ;       left   := ctrPt.h - height / 2;
        ;       top    := ctrPt.v - width / 2;
        ;       right  := ctrPt.h + height / 2;
        ;       bottom := ctrPt.v + width / 2;
        ;
        ;  Put data in direct to avoid overhead of SetRect.
        ;  Compute 'top'.
        move.w d5,d3                        ; get a copy of ctrPt.v and put in d3
        sub.w     d1,d3                     ; ctrPt.v - width / 2 = 'top' in d3
        swap      d3                        ; move 'top' to hiword

        ; Compute 'left'.
        move.w d4,d3                        ; get a copy of ctrPt.h & put in loword d3
        sub.w     d2,d3                     ; ctrPt.h - height / 2 = 'left' in loword d3

        ; We now have 'top' in hiword & 'left' in loword d3.
        ;  Compute 'bottom'.
        move.w d5,d6                           ; get a copy of ctrPt.v in d3
        add.w     d1,d6                     ; ctrPt.v + width / 2 = 'bottom' in d3
        swap      d6                        ; move 'bottom' to hiword

        ; Compute 'right'.
        move.w d4,d6                        ; get a copy of ctrPt.h and put in loword d6
        add.w     d2,d6                     ; ctrPt.h + height / 2 = 'right' in loword d6

        ; We now have 'bottom' in hiword & 'right' in loword d6.
        ;
        ; Assign data to dstBMap structure directly.
        move.w d7,rB(a2)                       ; put in rowBytes of dstBMap
        move.l d3,bds + topLOff(a2)       ; put the coord pair in direct
        move.l d6,bds + botROff(a2)       ; put the coord pair in direct
        move.w d7,d0                    ; put rowBytes in d0
        mulu      rB(a1),d0             ; multiply rowBytes src*dst = d0
        lsl       #3,d0                 ; multiply d0 by 8 for size of dstBMap

        _NewPtr CLEAR                   ; allocate size of dstBMap (a2 pts to it)
        cmp.l     #0,a0                 ; succesful?
        bne       @cont                 ; yes, continue
        clr.l     d0                    ; no, return FALSE & …
        return;                            ; …exit
cont:
        move.l a0,bA(a2)            ; move a0 to baseAddr field of dstBMap

        ;-------------------------------------------------------------------------------
        ; a0 = not used                              | d0 = rowBytes of srcBMap
        ; a1 = points to bits of srcBMap             | d1 = width / 2 (source)
        ; a2 = ptr to dstBMap                           | d2 = height / 2 (source)
        ; a3 = dest Bits                              | d3 = current word scrBMap
        ; a4 =                                           | d4 = copy of width of srcBMap
        ; a5 = not used                              | d5 = current dstBMap
        ; a6 = not used                              | d6 = insideLoop count (current bit)
        ; a7 = not used                              | d7 = rowBytes of dstBMap
        ;-------------------------------------------------------------------------------
        ; ****************************************************************
        ; *
        ; *       Start of actual rotation of bitMap
        ; *
        ; ****************************************************************
        ; Assumptions:
        ; height of destination bitMap = rowBytes of srcBMap
```

```
;
; initialize d0 to contain the pointer to the srcBMap
```

```
        move.w rB(a1),d0                ; get rowBytes of srcBMap

        ; get regs pointing to bits
        move.l (a1),a1                      ; a1 pts to BITS of srcBMap

        ; The original CCW rotate started with the upper leftmost word of the
        ; srcBMap and went contiguously down thru memory. To go CW instead I have
        ; to do the opposite, start at the end and go up.
        ;
        ; The a3 reg points to the word to get next and in this case I must set it
        ; to the end, so what does the 'end' mean here? It means use the actual
        ; pixel height of the srcBMap, as determined by the bounds, multiplied by
        ; the srcBMap.rowBytes, added to the start, which will make a3 point to 1
        ; word past the end. Since the 1st fetch does a pre-decrement, this points
        ; us at the last real word, huzzah-huzzah…
        ;
        ; 1 prob, remains, the orginal CCW routine assumed
        ; - rcy 2/26/89
        clr.l     d1
        move.w d0,d1                    ; load rowBytes of srcBMap
        mulu.w srcHeight,d1             ; times height of srcBMap = total wrds + 1
        add.l     d1,a1                 ; a1 now pts 1 word beyond end of srcBMap

        move.l (a2),a3                      ; a3 pts to BITS of dstBMap

        ; Compute the lower left corner of the dstBMap - this maps to
        ; the lower right corner of the srcBMap and will be used as base
        ; from which to offset into the dstBMap when we need to set a bit.
        clr.l     d1                    ; clear d1
        move.w d0,d1                    ; load rowBytes of srcBMap
                                        ; (rowBytes of srcBMap = ht of dstBMap)
        lsl.w     #3,d1                 ; multiply by 8 to get ht of dstBmap when rot'd
        mulu.w d7,d1                    ; multiply by rowBytes(dstBMap) = #words in
        sub.w     d7,d1                 ; dstBMap less one row
        add.l     a3,d1                 ; add the computed offset to the baseAddr of dstBMap
        move.l d1,lowerLeft             ; init lower left

        ;-------------------------------------------------------------------------------
        ;   a0 = not used                                       |   d0 = rowbytes of srcMap
        ;   a1 = points to bits of src bitMap |   d1 = SCRATCH
        ;   a2 = currentWord                                    |   d2 = colmCounter
        ;   a3 = dest bits                                      |   d3 = current srcMap word
        ;   a4 = SCRATCH                                        |   d4 = colmBitLoop counter
        ;   a5 = not used                                       |   d5 = current word destMap
        ;   a6 = not used                                       |   d6 = insideLoop count (current bit)
        ;   a7 = not used                                       |   d7 = rowBytes of dest bitMap
        ;-------------------------------------------------------------------------------
        ;

        ; init wordCount to zero
        ;
        move.w #0,wordCount             ; set wordCount to one before we start
        ;
        ; init colmLoop counter
        ;
        move.w d7,d2                    ; get rowBytes(dest)
        lsr       #1,d2                 ; divide by 2 - d2 now contains the outer counter
        move.w d2,colmCount             ; put the max value away for reference
        moveq.l   #0,d2                     ; set d2 to zero since we use it for the counter

colmLoop:
        ;
```

```
       ; this is the outside loop
       ;
       move.w #15,d4                    ; init colmBitLoop counter to 15
       move.l lowerLeft,a2              ; a2 is currentWord, init it

colmBitLoop:
       ;
       ; this is the loop that does the bit counting in the destMap - it is really a
       ; misuse of the 'dbra' instruction since it is not an independent loop but it
       ; is a quite cheap method of doing our counting
       ;

       cmp.w     #7,d4                  ; are we doing high byte or low?
       bgt       @doByte1               ; skip if high byte
       addq      #1,a2                  ; else point to low byte

doByte1:
rowLoop:

       ;
       ; loop across each row in the srcMap and keep count each time we do a row
       ; this 'rowCount' will used to compute the offset from 'lowerLeft'
       ;
       move.w -(a1),d3               ; get a word to rotate into d3 - rcy 2/26/89
       addq.w #2,wordCount           ; add 1 for each word ---jdo---

       lsr.w     #1,d3               ; test next bit in the current source word - rcy
2/26/89
       bcc.s     @noBitToChange         ; if bit's not set, skip to next
       bset      d4,(a2)                ; set proper bit in high or low byte in currentWord

noBitToChange:
       sub.l     d7,a2               ; subtract rowBytes(destMap) from current word

       ; Do the same as above, fifteen more times.
       lsr.w     #1,d3
       bcc.s     @1
       bset      d4,(a2)
@1  sub.l     d7,a2

       lsr.w     #1,d3
       bcc.s     @2
       bset      d4,(a2)
@2  sub.l     d7,a2

       lsr.w     #1,d3
       bcc.s     @3
       bset      d4,(a2)
@3  sub.l     d7,a2

       lsr.w     #1,d3
       bcc.s     @4
       bset      d4,(a2)
@4  sub.l     d7,a2

       lsr.w     #1,d3
       bcc.s     @5
       bset      d4,(a2)
@5  sub.l     d7,a2

       lsr.w     #1,d3
       bcc.s     @6
       bset      d4,(a2)
```

```
@6 sub.l     d7,a2
```

```
        lsr.w     #1,d3
        bcc.s     @7
        bset      d4,(a2)
@7  sub.l     d7,a2

        lsr.w     #1,d3
        bcc.s     @8
        bset      d4,(a2)
@8  sub.l     d7,a2

        lsr.w     #1,d3
        bcc.s     @9
        bset      d4,(a2)
@9  sub.l     d7,a2

        lsr.w     #1,d3
        bcc.s     @10
        bset      d4,(a2)
@10 sub.l     d7,a2

        lsr.w     #1,d3
        bcc.s     @11
        bset      d4,(a2)
@11 sub.l     d7,a2

        lsr.w     #1,d3
        bcc.s     @12
        bset      d4,(a2)
@12 sub.l     d7,a2

        lsr.w     #1,d3
        bcc.s     @13
        bset      d4,(a2)
@13 sub.l     d7,a2

        lsr.w     #1,d3
        bcc.s     @14
        bset      d4,(a2)
@14 sub.l     d7,a2

        lsr.w     #1,d3
        bcc.s     @15
        bset      d4,(a2)
@15 sub.l     d7,a2

        ;
        ; at this point we've just completed a word from the srcMap
        ;
        cmp       wordCount,d0          ; cmp wordCount to rowBytes-are we done with a row ?
        bne       @rowLoop              ; no - do another word

        cmp.w     #7,d4                 ; did we do high byte or low?
        bgt       @dobyte2              ; skip if high byte
        subq      #1,a2                 ; else undo the fudge to a2
dobyte2:

        ;
        ; we just completed a row so add 2 to the count
        ;
        move.w #0,wordCount             ; ---jdo--- reset wordCount to zero

        ;
        ; the following use of dbra keeps track of our destMap bit for us
```

```
;
move.l lowerLeft,a2                ; re-init currentWord
```

```
                                        ; set the currentWord to the lowerLeft(current)
                                        ; lowerLeft is moved over (right) by a word each
                                        ; time we do a row in the srcMap
        dbra      d4,@colmBitLoop       ; go thru the bits (15->0) in the
                                        ; destMap - then exit and do it
                                        ; again for rowBytes/2 times

        ;
        ; were done counting thru the bits of a destMap word - at this point we have
        ; actually finished a column in the destMap. The column is a multiple of
        ; 16 bits wide (and being at least 1 colm) and as tall as the destMap is
        ; (which is also equal to rowBytes of the srcMap)
        ;
        addq.l #2,lowerLeft             ; this moves us into the next
                                        ; 'column' of bits in the destMap
        addq.w #1,d2                    ; add 1 to the colmLoop counter
        cmp       colmCount,d2          ; are we done yet ?
        bne       @colmLoop             ; no - do it again
                                        ; else we're done

        move.l #1,d0                    ; return TRUE, all ok
        movem.l (sp)+,d3-d7/a2-a5          ; restore registers, bye!
    }
} /* RotateBMapCW */
```

●●●

From: jmunkki@kampi.hut.fi (Juri Munkki)
Subject: Re: Aligning bitmaps? (with code)

In article <3791@ncsuvx.ncsu.edu> jnh@ecemwl.UUCP (Joseph N. Hall) writes:

> >What is the most straightforward procedure for ensuring that a window's
> >portBits begin on a word boundary? What I would like to do is check after
> >the user creates or resizes a graphics window and shift the window horizontally
> >(if necessary) so that it will be aligned for later updates from an offscreen
> >bitmap.
> >
> >Also, how much speed difference is there between an aligned and unaligned
> >one-to-one CopyBits? I assume that performance is no better for an
> >odd-byte-aligned bitmap than it is for a totally unaligned bitmap, right?

The speed difference is visible. I assume that you have an offscreen bitmap that you regularly copy into a window. The best solution for alignment is to shift the bitmap instead of forcing the window. You will only waste a few bytes for the extra margin necessary.

Here are a few code fragments:

```
/*
>> AdjustVidPort adjusts the offscreen bitmap so that
>> CopyBits can always work in 32 bit alignment.
>>
>> Call AdjustVidPort whenever your window has moved.
>>
>> The method was originally suggested by Larry Rosenstein
>> from Apple's Advanced Technology Group.
*/
void   AdjustVidPort()
{
   BitMap temporary;
   GrafPtr   saved;
```

```
GetPort(&saved);
SetPort(&VidPort);
```

```
/*  Save old port location:
*/
   temporary=VidPort.portBits;

/*  Align port with magic formula:
*/
   MovePortTo((HMARGIN-VTWind->portBits.bounds.left) & 31,0);

/*  Align image:
*/
   CopyBits( &temporary,         &VidPort.portBits,
          &VidPort.portRect,     &VidPort.portRect,
          0,srcCopy);

   SetPort(saved);
}

/*  Open an offscreen grafport for VT100 screen
**  Make it 32 pixels wider than absolutely necessary.
*/
   RowBytes= ((ts.hsize*ts.charwidth+47)/16)*2;
...
   VidPort.portBits.bounds.left=0;
   VidPort.portBits.bounds.right=ts.hsize*ts.charwidth+32;
   VidPort.portBits.bounds.top=0;
   VidPort.portBits.bounds.bottom=ts.vsize*ts.lineheight;

   VidPort.portRect=VidPort.portBits.bounds;
   VidPort.portRect.right-=32;
   RectRgn(VidPort.visRgn,&VidPort.portRect);
   RectRgn(VidPort.clipRgn,&VidPort.portRect);
```

The above code makes sure that all your drawing is long word aligned. My terminal program doubled speed in cases where the window was badly positioned. All this has no effect on color screens, because I'm using a B&W bitmap even when the screen is in color. You would have to make small changes to apply the same technique to color pixmaps.

Juri Munkki jmunkki@hut.fi jmunkki@fingate.bitnet

●●●

From:Larry Rosenstein
Subject: Re: Aligning bitmaps?

In article <15288@dartvax.Dartmouth.EDU> earleh@eleazar.dartmouth.edu (Earle R. Horton) writes:

> In article <3791@ncsuvx.ncsu.edu> jnh@ecemwl.UUCP (Joseph N. Hall) writes:
> >What is the most straightforward procedure for ensuring that a window's
> >portBits begin on a word boundary? What I would like to do is check after
> >the user creates or resizes a graphics window and shift the window horizontally
> >(if necessary) so that it will be aligned for later updates from an offscreen
> >bitmap.
>
> I don't know what you mean by straightforward, but ANDing the
> horizontal coordinate with NOT-0x0F before calling MoveWindow() should
> work fine. Resize and create should not be a problem, but moving the

As someone noticed, Hypercard grids the window position and shows this by gridding the gray outline.

Another approach is to perform the alignment offscreen by allocating an offscreen bitmap that is 16 bits wider than necessary. Then you use a similar calculation to Earl's and align the offscreen buffer before copying it on the screen. I used

this technique in the old MacApp Paint program and it worked very well. (And it looked much better than doing unaligned CopyBits.) In this case, you don't have to worry about non-standard window behavior.

I'm positive that alignment is important for 1-bit deep bitmaps on a 68000-based machine. (It might be that on a 68020 machine it's not necessary.) As I recall, it required 1/2 second on a MacPlus to align a MacPaint image offscreen. I did this by calling CopyBits with the same bitmaps as source and destination.

Also, notice that what counts is the global coordinates of the destination. If you use my trick of aligning the offscreen bitmap, then you will need to compute the global coordinates of your window. To do this first call SetPort(yourWindow), assign (0, 0) to a Point, and use LocalToGlobal.

Finally, I'm not sure that a word boundary is optimal alignment for Color Quickdraw. I think you might want to use longword alignment. I haven't done any stuff with offscreen pixmaps, so I don't know for sure.

> and the screen depth is greater than one. Byte alignment seems to be
> all that's necessary. Where did you get your information that word
> alignment is needed?

I don't know about color, but for 1-bit deep images, word alignment alows QuickDraw to fetch and store longwords at a time (except for the edges). If you are only byte aligned then it may have to fetch things a byte at a time. (On a 68000 you can't fetch and store a word on an odd boundary; on a 68020 you can fetch and store on odd byte boundaries, but you pay a speed penalty.)

> I don't notice any difference between aligned or non-aligned
> BitMaps with a black and white screen. On the other hand, these are
> subjective results since I haven't done formal timing.

On a MacPlus it is noticeable, although I haven't done any formal measurements, either.

Larry Rosenstein, Apple Computer, Inc. Object Specialist

●●●

From:earleh@eleazar.dartmouth.edu (Earle R. Horton)
Subject: Re: Aligning bitmaps?

In article <3967@internal.Apple.COM> lsr@Apple.COM (Larry Rosenstein) writes:
>In article <15288@dartvax.Dartmouth.EDU> earleh@eleazar.dartmouth.edu
>(Earle R. Horton) writes:
...
>Finally, I'm not sure that a word boundary is optimal alignment for Color
>Quickdraw. I think you might want to use longword alignment. I haven't
>done any stuff with offscreen pixmaps, so I don't know for sure.
...
>> I don't notice any difference between aligned or non-aligned
>> BitMaps with a black and white screen. On the other hand, these are
>> subjective results since I haven't done formal timing.
>
>On a MacPlus it is noticeable, although I haven't done any formal
>measurements, either.

OK, I broke out the old Timex, and actually timed some big screen animation on the Mac II using 1, 2, and 4 bits screen depth with big offscreen PixMaps. My conclusion after about a half hour of testing was that the only alignment that makes any difference to Color QuickDraw CopyBits() is longword alignment. The performance increase when both PixMaps are longword-aligned is what I would call spectacular, too. It's about 2 to 1! There does appear to be little, if any, performance increase for byte or word alignment over random alignment, so I wasn't totally wrong before.

Sorry for any confusion my earlier posting might have caused.

Earle R. Horton

●●●

From: awd@dbase.UUCP (Alastair Dallas)
Subject: ThePort and TENew (problem with font)

The problem that had me tearing my hair out was incredibly simple as it turned out. I can keep it to myself, or I can share it at the risk of appearing inexperienced. Here it is--you won't care now, but maybe this will settle into a niche at the back of your brain and when you trip over this problem, you'll think of me.

I created an edit field with a call to TENew() and I set some text into it. Then, at update time, I call TEUpdate(). It didn't display anything. I made sure that the port was set correctly at update time. Still nothing. I did a FrameRect() before the TEUpdate()--I got the rect, but no text. I dumped the TERec--perfect. I fiddled with the clipRgn. Finally, in desperation, I single-stepped through TEUpdate().

Do you know why I wasn't seeing text? The port must be correct at TENew() time, not TEUpdate(). You can be set to anything and still update your TEs, but you have to make sure the port is right when you create the TE. In most apps, you switch ports when you open windows and your calls to TENew() come after that, so everything's fine. In my case, the port was wrong at init time, but I fixed it up later.

Hope it helps somebody somewhere sometime.

/alastair/

●●●

From: nicholas jackiw
Subject: Re: FatBits,thinbits and MapPt()

JON@wehi.dn.mu.oz (Jon Eaves) writes:

> I draw on the squash in the inverse colour (aaarrrgggh), 2 pixels get
> toggled horizontally and on the 'FatBits' version 1.5 fatPixels gets
> toggled.
>
> Why does this happen? I am pretty sure I am using MapPt() correctly
> and it works to draw the fatbits initially.
> ---

Why don't you try posting your code? Sounds like an arithmetic error someplace, not a MapPt problem.

One thing you might want to consider doing, which is tremendously easy, is to do all drawing to just one of your two pictures, and then use CopyBits to scale it to the other. CopyBits is a whole lot faster than doing a bunch of FillcRects on your FatBits image, and all of the scaling is automated by quickdraw.

The one disadvantage is you can't display the "grid" in your fatbits image, unless you superimpose it after CopyBitsing.

-Nick
--
nicholas jackiw \jackiw%campus.swarthmore.edu@swarthmr.bitnet

●●●

From: ftanaka@Apple.COM (Forrest Tanaka)
Subject: Re: 32-bit QuickDraw scaling behavior
Keywords: 32-bit Color QuickDraw scaling DrawPicture PenSize printing

In article <41276@apple.Apple.COM> ftanaka@Apple.COM (Forrest Tanaka) writes:

> The old characteristic of not scaling pen sizes when a picture was scaled
>was a bug in Quickdraw that's been fixed by 32-Bit Quickdraw. Since it's a
>bug fix, we thought that everyone would like it and no one would want to go
>back to the old way. So, there's no way right now to disable the fix aside
>from not using 32-Bit Quickdraw. It turned out that a few people depended on
>this bug, so there might be a way to disable pen size scaling in the future.
>I kind of doubt it, but it's not up to me.

Judging by the mail I've gotten, it looks like more people than I'd realized thought that this bug was pretty cool. So, I tried out something that I suspected might work for some people to fix the fix, and it does work.

If y'all turn to page 197 of Inside Macintosh I, you'll see a discussion about Quickdraw bottlenecks. By making your own bottlenecks, you can customize almost everything that Quickdraw does. When you call LineTo, StdLine is called to do the drawing. By providing your own routine to replace StdLine, you can make almost anything you want happen when LineTo is called. The same applies to all the others you see on the next couple of pages. Your StdLine replacement can even call StdLine, and it can even do things after calling StdLine--a non-breakable form of tail-patching if you want to think of it that way.

When you call DrawPicture, these bottlenecks are called to actually draw the contents of the Picture. So if the picture has a line in it, StdLine or your StdLine replacement will be called to draw the line. That's the key to fixing the fix.

What you can do is to create your picture in the usual way. Then before you call DrawPicture, replace the StdLine, StdRect, StdRRect, StdOval, StdArc, StdPoly, and StdRgn bottlenecks with your own bottlenecks. Except the StdLine case, the typical bottleneck will look something like:

```
if the verb is 'frame' then
  call PenSize(1,1)
call the standard bottleneck
```

That's it! It doesn't make sense for StdLine to take a verb, so you can just call PenSize(1,1) and then call the standard bottleneck in that case. What this is saying is that if the programmer or the picture calls Frame<X>, then set the pen size to 1 before calling the routine that actually draws the <X>.

The downside of this for some of you is that if you export your picture to another program, the pen sizes will be scaled again because they won't know to put in their own bottlenecks to set the pen size to 1 before drawing the objects. But, there is sort of a way around this even if you know what scale you want the result to be. In more pseudo-code, the operation could look something like this:

```
original picture = OpenPicture(small rectangle)
Draw some framed things
ClosePicture

Replace bottlenecks with the 'PenSize(1,1)' bottlenecks
scaled picture = OpenPicture (big rectangle)
DrawPicture (original picture, big rectangle)
ClosePicture
Set the bottlenecks back to what they were
```

Now, 'scaled picture' will contain a scaled version of 'original picture', but all the lines and frames will still have a pen size of 1 because you replaced the bottlenecks. Of course, if you scale 'scaled picture' larger, then the pen sizes will again be scaled unless you again replace the bottlenecks.

I doubt that this is practical in every case, but it sounds like some of you can do something like this.

Another option that I thought of as I was typing this is to patch PenSize. After giving it about three seconds of thought, I don't see what would be wrong with this. If any of you see a problem with this or the bottleneck method, or if you have a better way, please please please post it.

--
Forrest Tanaka Macintosh Developer Technical Support Apple Computer, Inc.

●●●

From: lsr@Apple.COM (Larry Rosenstein)
Subject: Re: Need help creating a Quickdraw picture

In article <7366@jarthur.Claremont.EDU> mwilkins@jarthur.Claremont.EDU (Mark Wilkins) writes:

> Every time I've done it copying from a bitmap to itself, it's given me an
> empty picture definition.

Perhaps QuickDraw only records operations that involve the current grafPort's portBits. If you copy from one offscreen bitmap to another then the operation wouldn't involve the portBits. You might try installing your offscreen bitmap into a grafPort and use that port for creating the picture.

Larry Rosenstein, Apple Computer, Inc.

●●●

From: kaufman@Neon.Stanford.EDU (Marc T. Kaufman)
Subject: Re: Still need help creating a QuickDraw picture

In article <2223@dftsrv.gsfc.nasa.gov> seiler@amarna.gsfc.nasa.gov writes:

```
... (stuff deleted)
>SetPort(windPtr);
>picHndl = OpenPicture(&(windPtr->portRect));
>CopyBits(&(windPtr->portBits), &(windPtr->portBits),
>        &(windPtr->portRect), &(windPtr->portRect), srcCopy, NULL);
>ClosePicture();

>no copybits opcode shows up, but the sequence: version op, version, header op,
>size, fBBox, <4 bytes reserved>, DefHilite, Clip, opEndPic occurs.
```

Let me try (after all, everyone else has :-)). I don't see why you are changing ports. Why not just do the OpenPicture on your screen grafPort? The picture traps will catch everything.

If you really want to use another port, you need to make sure that the portRect, clipRgn, and visRgn are correct. You do NOT need to worry about the portBits.bounds, and indeed, you do not need any bitMap/Pixmap actually allocated for a port that is only used for capturing a picture.

Marc Kaufman (kaufman@Neon.stanford.edu)

●●●

From: pierce@radius.com (Pierce T. Wetter III)
Subject: Re: Need help animating with 24 bit card

```
>I wrote a program about 2 years back which needed to
>change the color of circles on the screen at a very fast rate. I
>accomplished this by calling AnimatePallette. Of course this limited
>me to a maximum of 254 objects at any time due to our 8 bit card.  I
>now need to write a similar application which does the same thing except
>with many more objects... in the range of about a thousand.

>I thought it might be possible to do the same thing except with a
>24 bit card.
```

Funny you should ask about this, I just did this for PrecisionColor (radius'sMonitor calibration gizmo, that I am responsible for writing the software to)

You can animate colors on a 24 bit board, but there are some caveats:

1: You can only do 256 arbitrary colors at at time.
2: Other colors besides the one you wish to animate will animate as well.
3: For non arbitrary colors, you can animate 256 shades of red, 256 of blue and 256 of green.

The necessary technique is to use the gamma table to do your animation for you. All the 24-bit cards on the mac have an 8-bit by 8-bit table in front of the dac. One way to animate a color is to pick an entry in all three of these tables. Lets pick (0xFE,0xFE,0xFE) as an example. If I draw a circle in that color and then change those particular entries in the gamma table to whatever color I want, the circle will change to that color. (This is how the not-yet-released Turbo Precision Color works.)

This still limits you to 254 colors (assuming you don't animate full white and full black). However, you can do more by animating your circles to just red, and just green and just blue. If you do that, then you can use all the colors (N, 0, 0) & (0,N,0) and (0,0,N). If you don't mind your circles overlapping when you animate (for instance, you could have an indicator LED colored (N,N,0). You can then animate red to be indicating one variable, and green to be the other variable. The LED will then take on any shade from red to green to yellow. This lets you have more animating circles, but you can still only have 3x254=762 INDEPENDENT circles.

Of course, this probably won't help you any, but I had fun talking about what I was doing.

Pierce

P.S.: Since I mentioned it, its only fair to say that the new software is just a minor upgrade to support the 8..24 and 8..24GC. While I was at it I made it 3x faster, but that's beside the point.
--
My postings are my opinions, and my opinions are my own not that of my employer.

●●●

From: russotto@eng.umd.edu (Matthew T. Russotto)
Subject: Re: Setting up CLUTs - Advise???????

In article <24664.266d7b29@kuhub.cc.ukans.edu> brownrigg@kuhub.cc.ukans.edu writes:
>Can anyone suggest the general mechanism one needs to use to display a
>pixMap with PRECISELY CHOSEN colors?? I've read and re-read the relevent
>chapters of IMV5 and am still confused and frustrated as to what to do.

Yep. In fact, I can suggest TWO ways. One is to put a 'pltt' resource id 0 in your application resource file. The resource should contain valid pltt data with only two colors: black, and white. Then just use SetEntries from the color manager chapter to put YOUR colors in. Of course, be careful to SaveEntries and RestoreEntries at suspend and resume time under multifinder, so other applications don't get their colors messed up, and you don't have yours messed up. You also have to be very careful in a multi-screen environment using this method.

The second, and far superior method (IMO) is to use NewPalette to create a palette with the 254 colors you need, plus black and white in the first two entries (yes, I know some quantizers don't always put out a black and white. Tough.). Black and white should be marked as pmCourteous, all the other colors should be pmTolerant with a tolerance of 0 (pmIntolerant?). Then SetPalette(yourwindow, yourpalette, TRUE); ActivatePalette(yourwindow); to make the window automatically keep your color environment;

(the reason black and white should be set to pmCourteous is that the Palette manager used to reserve an additional index for them, in addition to the standard positions at the beginning and end of the color table. I don't know if this is still the case or if it is correct behavior, but setting them to courteous won't hurt)
--
Matthew T. Russotto russotto@eng.umd.edu

●●●

From: russotto@eng.umd.edu (Matthew T. Russotto)
Subject: Re: ? Bad pixmap data written by OpenPicture & CopyBits

In article <1990Jun13.181220.1@dev8j.mdcbbs.com> astor@dev8j.mdcbbs.com writes:

>My application has an offscreen pixmap, and I'm trying to copy the
>pixmap onto the scrap as a simple PICT.
>
>I use OpenPicture, CopyBits, ClosePicture, ZeroScrap, and PutScrap.

Try OpenPicture, CopyBits,ClosePicture,ZeroScrap,HLock,PutScrap.

--

Matthew T. Russotto russotto@eng.umd.edu russotto@wam.umd.edu

●●●

From: ccc_ldo@waikato.ac.nz (Lawrence D'Oliveiro, Waikato University)

Subject: Re: Fading ...(dissolve effect)

I had a play once with the idea of doing a "dissolve" effect using absolutely standard QuickDraw calls. It turns out that the CopyMask operation (Inside Mac page IV-24) is almost custom-designed for this sort of thing.

What I did was create a number of off-screen bitmaps, all the same size as the window in which the pictures will be displayed. The number of bitmaps corresponds to the number of "stages" in which the dissolve will take place: more stages makes the dissolve smoother, but takes longer.

If you look at a particular pixel position in all of these bitmaps, you will find that the corresponding bit is 1 in exactly one of the bitmaps, and zero in all the others. Thus the way you use them is by making repeated calls to CopyMask, each time specifying the complete new picture (drawn into an offscreen buffer) as the SrcBits argument, the PortBits for the window in which the display will appear as the DstBits, and a different one of the "stage" bitmaps as the MaskBits argument. The 1 bits in MaskBits select which pixels get copied from SrcBits to DestBits; once you've gone through all the stage bitmaps, the window will be displaying the complete new picture.

Of course, the distribution of 1 bits within each stage bitmap has to be reasonably random. The way I ensured this was, for each pixel position, to generate a random integer in the range 1 to the number of stages, and use that number to select the bitmap which would get the 1 bit (all the rest would get 0 for that pixel).

This random-number generation turned out to take quite a long time; to speed things up, I changed it to operate on just a 32*32 portion of the bitmap, and then simply replicated that bit pattern "cell" by "tiling" across and down with CopyBits until I'd covered the entire bitmap. Another way would be to store the pregenerated 32*32 bit patterns in a resource and just read them into memory when the program runs.

Though I looked for it, I couldn't see any visible effect of repeating the same 32*32 pattern of bits--the distribution of the newly-appearing pixels still looked completely random. I didn't experiment to see if I could get away with a smaller cell size.

Another nice thing is that this technique also works with Color QuickDraw, and should also look good on a 24-bit-per-pixel display. For maximum speed, you should make SrcBits a PixMap with the same depth and colour table as DstBits, but MaskBits is still a plain, ordinary bitmap.

Speed? On 1-bit-per-pixel displays, with 6 dissolve stages, the result was quite presentable, even on a Mac Plus (though not as fast as HyperCard, which manages to use 12 stages). 8 bits per pixel on a vintage Mac II was actually slower than 1 bit per pixel on a Plus--but at least it worked!

Lawrence D'Oliveiro

●●●

From: h+@nada.kth.se
Subject: Re: Drawing all over the desktop

larsen@ginger.Princeton.EDU writes:

> ---
> I know that Apple doesn't want applications to draw all over the desktop.
> Since I'm doing it anyway, though, I need advice in order to do it as cleanly
> as possible. At the moment I'm checking screen depth and writing into video
> memory, but I would prefer a quickdraw solution.
> ---

The stuff to do is drawing into the WMgrPort. This is as compatible as it gets, though that isn't very. You have to be VERY careful to restore everything as it was before you call {Get/Wait}NextEvent. This means ony XORing, among other things. (Well, if it's a game and you don't want it to be MF compatible nor System 7 compatible, you don't have to restore the drawings, just the state of the WMgrPort)

I have written an INIT that, among other things, needs to draw in the WMgrPort, called SetWindow INIT. The source and the init is available via FTP from rascal.ics.utexas.edu, and is in the directory mac/hacking (if memory serves me right, might be programming or somewhere completely else...)

Generally, it is a VERY BAD idea to draw outside of windows. Only newly-converted MS-DOS programmers try and do this

without a good reason. (And I mean GOOD)

Happy hacking,

h+

●●●

From: hildreth@cg-atla.agfa.com (Lon Hildreth)
Subject: Re: PICT hacking question
Keywords: PICT

In article <1732@mountn.dec.com> minow@bolt.enet.dec.com (Martin Minow) writes:

> >The original image was scanned at 300 dpi. When I look at the output
> >image (even without processing), it is clearly at a coarser resolution
> >(probably 72 dpi).
> >
> >Could anyone tell me how I get the actual image at its intended resolution
> >(both into my program and out to a new PICT file)? Inside-Mac and the
> >TechNotes stack don't seem to have anything relevant.

You need to put yourself in the StdBits bottleneck during playback. The hRes and vRes fields of the srcBits passed in contain the horizontal and vertical resolution of the image in dots per inch. They're Fixed point numbers, so you can call Fix2Long on them to get usable numbers. First, make sure that srcBits is a PixMap by checking the high bit of rowBytes.

For creating a new PICT, make sure that the hRes and vRes fields of your source pixmap is set properly. Also, the dest rect of the CopyBits call should be at 72 dpi.

--
Lon Hildreth ...!{decvax,uunet,samsung}!cg-atla!hildreth

●●●

From: keith@Apple.COM (Keith Rollin)
Subject: Re: Wanted- a window that comes up in black

In article <23026@dartvax.Dartmouth.EDU> anarch@eleazar.dartmouth.edu (The Anarch) writes:

> > Apologies if this is an elementary question; I'm still pretty new to Mac
> >programming. Anyway, I'm trying to get a window to come up onscreen with a
> >black content region. Now I know that I can easily do NewWindow and then
> >PaintRect the window's body, but that causes a brief white flash, which I'd
> >like to avoid but have been unable to get rid of. What I *have* tried:
> >changing the bkColor and bkPat for the window in various ways before a
> >ShowWindow; I haven't gotten any positive results.

Here's one for the oneliners file:

The window Mgr doesn't use your window's bkPat when erasing the content region.

No. That would be too simple, too easy. Instead, since the content region is drawn in the WMgrPort, the Window Manager uses the WMgrPort's bkPat. Since this is always white, your window's content region is always drawn in white when it is updated.

There are several not-so-good ways around this. As I recall, if you have a color window, the window's rgbBkColor IS respected, and you can have your window updated in any color you want (but not pattern). You can also try munging around with the PaintWhite low-memory global, which can be used to prevent the window manager from clearing your content region to any color at all, allowing you to do it. However, there are problems with that, as Inside Mac I-297 points out. Finally, you could write a WDEF wherein the entire window is a structure region, and where there is no content region. Since the WDEF is solely responsible for drawing the strucRgn, you will have total control over how it gets redrawn.

--

USENET Macintosh Programmer's Guide

Keith Rollin --- Apple Computer, Inc. --- Developer Technical Support

●●●

From: jwwalker@usceast.UUCP (Jim Walker)
Subject: Re: Wanted- a window that comes up in black

I thought of another "not so good way" that Keith Rollin didn't mention: Before showing your window, temporarily change the bkPat of the Window Manager port to black. Be sure to change it back to white once the window is drawn! You may also need to set the window's bkPat to black to prevent a flash. And you would probably need a custom WDEF that can draw its frame properly when the bkPat is black. (I did enough experimentation with this to convince myself that it could be made to work.)
--

Jim Walker jwwalker@cs.scarolina.edu 76367.2271@compuserve.com

●●●

From: jackiw@cs.swarthmore.edu (Nick Jackiw)
Subject: Re: Dragging Bitmaps?

siegel@endor.harvard.edu (Rich Siegel) writes:

> Does anyone have any suggestions on how to drag small bitmaps around, in the
> same fashion as MacPaint? Currently I drag outlines around, but it would be
> much nicer to move the object as a whole.
>
> R.

Sure: you'll need to use some offscreen bitmaps though. The major design decisions that affect the process are: (1) do you want it to be really smooth? (2) must your bitmaps be of eccentric shape or can they be limited to rects?

(1) For rectangles

The Cheap Way:

Get two offscreen bitmaps the size of the one to drag, and fill them both with the bitimages. Call these SOURCE and BUFFER. Now watch the mouse. Every time it moves, calculate the new location (presumably you measure the offset into the bitmap at which the mouse went down, and then calculate the new location based on this offset and the vector of mouse movement, so the mouse "sticks" to the same part of the rectangle). Copy BUFFER to the old location, copy the rect of "underlaid" bits at the new location to BUFFER, and then copy SOURCE to the new location. Repeat until mouseup.

The problem with this is that you'll have flicker--you'll be overwriting background bits with source bits and source bits with background bits.

The more expensive way:

Get an offscreen copy of the area in which you wish to permit dragging, without the dragable image in it. Call this BACKBITS. Now get an offscreen copy of the draggable image, call this SOURCE. Now get a third bitmap, the size of BACKBITS, left blank; call this SCRATCH. For each new mouse move, copy the UnionRect of the oldLocation and the newLocation from BackBits to Scratch, and then throw in (to Scratch) a copy of the SOURCE at new location. Copy this unionRect to the screen.

(2) For irregularly-shaped bitmaps

You know about BitMap2Rgn, I presume. It's on the DTS cds in object code, or you can count on it being available if you'll only run on 32Bit QD machines. (There may be licensing hassles concerning the object code; read the docs or write your own.) With BitMap2Rgn, get a region that includes/excludes the appropriate parts of your SOURCE. On a blank bitmap the size of SOURCE, do a FillRgn(bitsRgn,black). This gives you a mask, which you should use in either of the above schemes: wherever you formerly did CopyBits(source,somePlace...) now do a CopyMask(source, somePlace,maskBits,...).

If you want source code, I can whip something up. There are slightly more efficient ways to do it than the above, but that's the basic scheme as I understand it.

Call this BACKBITS.

--
nicholas jackiw | jackiw%campus.swarthmore.edu@swarthmr.bitnet

●●●

From: dowdy@apple.com (Tom Dowdy)
Subject: Re: Wanted- a window that comes up in black

Here's YET ANOTHER way to do this. You can judge if it is less or more terrible than those already suggested. I call it about even, but the one effect that it relies on IS documented.

I place it behind the WDEF approach, and ahead of hitting fields in the WMgrPort in the cleaniness rankings.

a) Open a port (PORT, not window) in the location where the window will be.
b) Fill port with black
c) Dispose of port.
d) Slam low memory PaintWhite ($9DC) to be zero (this keeps the window manager from filling the window with white, but leaves what is behind it, ie black), save the old copy.
e) Open your window, which will get the black area, and will NOT be filled with white.
f) Restore PaintWhite.

Tom Dowdy Internet: dowdy@apple.COM

●●●

From: lsr@Apple.COM (Larry Rosenstein)
Subject: Re: Dragging Bitmaps?

In article <3398@husc6.harvard.edu> siegel@endor.harvard.edu (Rich Siegel) writes:
>
>Does anyone have any suggestions on how to drag small bitmaps around, in the
>same fashion as MacPaint? Currently I drag outlines around, but it would be
>much nicer to move the object as a whole.

The basic approach is to cache the image behind the moving bitmap, and use that cache to erase the bitmap before it moves. So the basic code would be to use 1 CopyBits to erase the old bitmap, and 1 CopyBits to draw the bitmap in its new position. (This assumes that you have cached the entire view beforehand. This is always better than copying the bits from the screen.)

You will get better results if you do both steps with 1 CopyBits. To do this you need another offscreen bitmap the size of the union of the old and new rectangles. You initialize this bitmap from the larger cache, and draw the moving bitmap in its desired position. Then you use 1 CopyBits to copy everything onto the screen at once.

--
Larry Rosenstein, Object Specialist

●●●

From: beard@ux1.lbl.gov (Patrick C Beard)
Subject: Re: 2-16-256-million color mode
Keywords: standards, color, monitors

In article <1037@swan.ulowell.edu> jkeegan@hawk.ulowell.edu (Jeff Keegan) writes:
#Is there ANY "acceptible" way to change the color mode other than telling

#someone to go change it?

There is a new call that is documented in the latest batch of Tech Notes called "SetDepth()". This lets you do precisely this from software.

On the other hand, the user interface guidelines for this aren't really spelled out. I think that an application should notify the user that it is about to change the depth of the monitor, and allow the user to quit if that is not what he wants to do.

- Patrick Beard, Macintosh Programmer (beard@lbl.gov) -

●●●

From: minow@mountn.dec.com (Martin Minow)
Subject: Reading PICT's revisited.

Thanks for all the good replies to my question on reading a 300 dpi PICT into a (72 dpi) window. Here's the code I use to read and write PICTs into an offscreen GrafPort.

Martin Minow
minow@bolt.enet.dec.com

```
/*                    ReadPicture.c                    */
/*
* Picture I/O
*
* For some reason, the PICT dimensions do not correctly
* describe the picture: it loses its 300 dpi resolution
* when read in. MAGIC scales the PICT by a factor of 4.
* We should get the "real" scale factor by inserting
* our function into the StdBits bottleneck.
*/
#include "CleanPICT.h"
#include <SysErr.h>
#define MAGIC(size)  ((((long) size) * (4 * 72)) / 72)

/*
* See TechNote 27 (original format), and the Essential
* MacTutor, vol 3, p 417.  (This is now unused.)
*/
typedef struct {
 OSType  fType;          /* 'DRWG' for MacPaint files  */
 short    hdrID;          /* 'MD' for MacPaint format    */
 short    version;        /* File version                */
 short    prRec[60];      /* 120 byte print record      */
 Fixed    xOrigin;        /* Drawing origin              */
 Fixed    yOrigin;        /* Drawing origin              */
 Fixed    xScale;          /* Screen resolution          */
 Fixed    yScale;          /* Screen resolution          */
 short    atrState[31];    /* Drawing attribute state    */
 short    lCnt;            /* Top-level objects          */
 short    lTot;            /* Total number of objects    */
 long    lSiz;            /* Total size of list         */
 Long    top;            /* Box enclosing all objects  */
 Long    left;
 Long    bottom;
 Long    right;
 short    filler1[141];    /* 282 bytes unused           */
} MacDrawHdrRec;

static int      PICTfile;  /* The file ioRefNum         */

/*
* Use this to peek into the bitmap as it is read in.
* The peeking is done by jumping into the Debugger.
*/
pascal void      myStdBits(
```

```
   BitMap *, Rect *, Rect *, int, RgnHandle);
pascal void      read_picture_data(Ptr, int);
pascal void      write_picture_data(Ptr, int);
```

```
/*
* read_picture() reads the picture from the PICT file,
* constructing the window at the proper size.
* "window" is really a document structure:
*    typedef struct {
*   WindowRecord window;
*   GrafPtr      pictPort; -- offscreen grafport
*   Rect      pictSize; -- true PICT size
*    } DocumentRecord;
* The DOC macro handles type casting and dereferencing.
*/
OSErr
read_picture(window, theFile)
WindowPtr      window;    /* Read into this document  */
int            theFile;   /* From this open PICT file */
{
   PicHandle        handle;
   QDProcs          procedures;
   OSErr            status;
   long            size;
   long            place;
   Rect            box;
   GrafPtr          oldPort;
   MacDrawHdrRec  header;

   PICTfile = theFile;
   handle = (PicHandle) NewHandle(sizeof (Picture));
   if (handle == NIL) {
     DebugStr("\pCan't get memory for picture");
     return (MemError());
   }
   /*
    * Read the MacDraw header record -- that was a
    * good idea, but it didn't work as the headers
    * are garbage in the PICT I'm using.
    */
   if (sizeof header != 512)
     DebugStr("\pMacDrawHdrRec wrong size!");
   read_picture_data((Ptr) &header, sizeof header);
   HLock(handle);
   read_picture_data((Ptr) *handle, sizeof (Picture));
   HUnlock(handle);
   box = (**handle).picFrame;
   DOC.pictSize = box;
   box.right = MAGIC(box.right);
   box.bottom = MAGIC(box.bottom);
   GetPort(&oldPort);
   /*
    * Create an offscreen GrafPort.  See TechNote 41.
    */
   DOC.pictPort = CreateOSGrafPort(box);
   if (DOC.pictPort == NIL) {
     DebugStr("\pNo memory for picture");
     SetPort(oldPort);
     return (MemError());
   }
   SetStdProcs(&procedures);
   DOC.pictPort->grafProcs = &procedures;
   procedures.getPicProc = (Ptr) read_picture_data;
/*  procedures.bitsProc = (Ptr) myStdBits;    -- unused  */
   DrawPicture(handle, &box);
   DOC.pictPort->grafProcs = NIL;
```

```
    DisposHandle((Handle) handle);
    SetPort(oldPort);
    /*
```

```
    * Check for errors by getting the file position and
    * checking that it is at the end of file.
    */
   if ((status = GetEOF(PICTfile, &size)) != noErr
    || (status = GetFPos(PICTfile, &place)) != noErr) {
      DebugStr("\pCan't get EOF or file position");
      return (status);
   }
   if (size != place) {
     DebugStr("\pDidn't read entire picture");
     return (dsSysErr);
   }
   /*
    * Ok so far.  Now, change the window size so the
    * picture fills the window -- but keep the proportions
    * as close to the original as possible.
    */
   SetRect(
     &box,
     2,
     GetMBarHeight() * 2,
     width(DOC.pictSize) + 2,
     GetMBarHeight() * 2 + height(DOC.pictSize)
   );
   if (box.bottom > (screenBits.bounds.bottom - 2)) {
     box.bottom = screenBits.bounds.bottom - 2;
     size = height(box);
     box.right = box.left
       + ((long) width(box) * size) / height(DOC.pictSize);
   }
   if (box.right > (screenBits.bounds.right - 2)) {
     box.right = screenBits.bounds.right - 2;
     size = width(box);
     box.bottom = box.top
       + ((long) height(box) * size) / width(DOC.pictSize);
   }
   SizeWindow(window, width(box), height(box), TRUE);
   InvalRect(&window->portRect);
   ShowWindow(window);
   return (MemError());
}

/*
* This should implement a "vanilla" StdBits -- for some
* reason, though, it "bombs" when it runs to completion:
* probably because its called with a NULL argument at
* eof.  It was only used to check that the created
* "MAGIC" bitmap is the same size as the bitmap inside
* of the PICT -- by running the function under the
* debugger.
*/
pascal void
myStdBits(srcBits, srcRect, dstRect, mode, maskRgn)
BitMap    *srcBits;
Rect      *srcRect;
Rect      *dstRect;
short      mode;
RgnHandle  maskRgn;
{
   CopyBits(
     srcBits,
     &thePort->portBits,
     srcRect, dstRect,
```

```
    mode,
    maskRgn
);
```

```
}

/*
* Called indirectly to read a chunk of picture data.
*/
pascal void
read_picture_data(data_ptr, byte_count)
Ptr        data_ptr;
int        byte_count;
{
   OSErr        status;
   long        count;

   count = byte_count;
   status = FSRead(PICTfile, &count, data_ptr);
   if (status != noErr)
     DebugStr("\pReading picture");
}

/*
* write_picture() writes the current picture to a
* specified (open) file.  It should be redone to
* add error handling.
*/
void
write_picture(window, theFile)
WindowPtr      window;
int            theFile;
{
   PicHandle    picHandle;
   QDProcs      procedures;
   int          i;
   long        temp;
   Picture      header;
   GrafPtr      tempPort;
   GrafPtr      oldPort;

   GetPort(&oldPort);
   PICTfile = theFile;
   /*
    * Write a dummy MacPaint header
    */
   temp = 0L;
   for (i = 0; i < 512; i += sizeof temp)
     write_picture_data((Ptr) &temp, (int) sizeof temp);
   header.picSize = 0;
   header.picFrame = DOC.pictSize;
   write_picture_data((Ptr) &header, (int) sizeof header);
   /*
    * Write the picture by creating a GrafPort with the
    * same dimensions as the original, then drawing the
    * cleaned up picture.  Can this be done without
    * creating a new GrafPort?
    */
   tempPort = CreateOSGrafPort(DOC.pictSize);
   if (tempPort == NIL) {
     DebugStr("\pNo space for temp port");
     return;
   }
   SetStdProcs(&procedures);
   tempPort->grafProcs = &procedures;
   procedures.putPicProc = (Ptr) write_picture_data;
   picHandle = OpenPicture(&tempPort->portRect);
```

```
CopyOSGrafPort(DOC.pictPort, tempPort);
ClosePicture();
KillPicture(picHandle);
```

```
   DeleteOSGrafPort(tempPort);
   DOC.pictPort->grafProcs = NIL;
   SetPort(oldPort);
}

/*
* Called indirectly to write a chunk of picture data.
*/
pascal void
write_picture_data(data_ptr, byte_count)
Ptr        data_ptr;
int        byte_count;
{
   OSErr        status;
   long        count;

   count = byte_count;
   status = FSWrite(PICTfile, &count, data_ptr);
   if (status != noErr)
     DebugStr("\pWriting picture");
}
```

●●●

From: minow@mountn.dec.com (Martin Minow)
Subject: How to read a Pict into a scrollable window

In article <1990Jun29.215003.26596@Neon.Stanford.EDU> jhsu@Neon.Stanford.EDU (Jeffrey H. Hsu) writes:

> >Can somebody tell me how to read a PICT document into a scrollable window? I
> >can deal with the PICT as a resource but not as a file.

I can't help with the scrollable part, but here's some code I use to read a PICT file into an offscreen bitmap. (The code is incomplete, but it should give you the basic structure.) There's a Tech Note that explains what's going on.

```
extern int   picture_file;       /* PICT file, already open */

void      read_background_picture(void);
pascal void  read_picture_data(Ptr, int);


/*
* Called indirectly to read a chunk of picture data.
*/
pascal void
read_picture_data(data_ptr, byte_count)
Ptr    data_ptr;
int    byte_count;
{
   OSErr     status;
   long      count;

   count = byte_count;
   status = FSRead(picture_file, &count, data_ptr);
   if (status != noErr)
       status_error("\pReading background picture", status);
}

void
read_background_picture()
{
   PicHandle handle;
```

```
QDProcs      procedures;
OSErr     status;
long      size;
long      place;
```

```
    int    picture_width, picture_height;
    int    screen_width, screen_height;
    Rect       box;

    SetStdProcs(&procedures);
    ((GrafPtr) overhead_window)->grafProcs = &procedures;
    /*
     * Use our function to read chunks of picture data
     */
    procedures.getPicProc = (Ptr) read_picture_data;
    handle = (PicHandle) NewHandle(sizeof (Picture));
    /*
     * Rewind the file and skip over the MacDraw header block.
     */
    if ((status = SetFPos(picture_file, fsFromStart, 512L)) != noErr) {
        status_error("\pCan't rewind picture file", status);
        return;
    }
    HLock(handle);
    /*
     * Start by reading the header information.
     */
    read_picture_data((Ptr) *handle, sizeof (Picture));
    HUnlock(handle);
    /*
     * Center the picture in the window and read it in.
     */
    box = (*handle)->picFrame;
    OffsetRect(         /* Normalize topleft to 0,0   */
        &box,
        -box.left,
        -box.top
    );
    screen_width = thePort->portRect.right - thePort->portRect.left;
    screen_height = thePort->portRect.bottom - thePort->portRect.top;
    picture_width = box.right - box.left;
    picture_height = box.bottom - box.top;
    OffsetRect(         /* Now, center it nicely      */
        &box,
        (screen_width - picture_width) / 2,
        (screen_height - picture_height) / 2
    );
    DrawPicture(handle, &box);
    ((GrafPtr) overhead_window)->grafProcs = NIL;
    DisposHandle((Handle) handle);
    /*
     * Check for errors by getting the file position and
     * checking that it is at the end of file.
     */
    if ((status = GetEOF(picture_file, &size)) != noErr
     || (status = GetFPos(picture_file, &place)) != noErr)
         status_error("\pCan't get EOF or file position", status);
    if (size != place)
        status_error("\pDidn't read entire picture, lost", size - place);
}
```

Martin Minow

●●●