New Technical Notes

Macintosh

Developer Support

Off-Screen Bitmaps

Imaging

Revised by: Jon Zap & Forrest Tanaka June 1990
Written by: Jim Friedlander & Ginger Jernigan July 1985

This Technical Note provides an example of creating an off-screen bitmap, drawing to it, and then copying from it to the screen.

Changes since April 1990: Clarified the section on window updates with off-screen bitmaps to explicitly limit these updates to your own windows.

The following is an example of creating and drawing to an off-screen bitmap, then copying from it to an on-screen window. We supply this example in both MPW Pascal and C.

MPW Pascal

First, let's look at a general purpose function to create an off-screen bitmap. This function creates the `GrafPort` on the heap. You could also create it on the stack and pass the uninitialized structure to a function similar to this one.

```
FUNCTION CreateOffscreenBitMap(VAR newOffscreen:GrafPtr; inBounds:Rect) : BOOLEAN;

VAR
 savePort  : GrafPtr;
 newPort   : GrafPtr;

BEGIN
 GetPort(savePort);        {need this to restore thePort after OpenPort changes it}

 newPort := GrafPtr(NewPtr(sizeof(GrafPort)));    {allocate the GrafPort}
 IF MemError <> noErr THEN BEGIN
   CreateOffscreenBitMap := false;                {failed to allocate it}
   EXIT(CreateOffscreenBitMap);
 END;
 {
 the OpenPort call does the following . . .
   allocates space for visRgn (set to screenBits.bounds) and clipRgn (set wide
   open)
   sets portBits to screenBits
   sets portRect to screenBits.bounds
   etc. (see IM I-163,164)
   side effect: does a SetPort(offScreen)
 }
 OpenPort(newPort);
 {make bitmap exactly the size of the bounds that caller supplied}
 WITH newPort^ DO BEGIN {portRect, clipRgn, and visRgn are in newPort}
   portRect := inBounds;
   RectRgn(clipRgn, inBounds);        {avoid wide-open clipRgn, to be safe}
```

```
    RectRgn(visRgn, inBounds);          {in case inBounds is > screen bounds}
  END;

  WITH newPort^.portBits DO BEGIN       {baseAddr, rowBytes and bounds are in
  newPort}
    bounds := inBounds;
    {rowBytes is size of row  It must be rounded up to even number of bytes}
    rowBytes := ((inBounds.right - inBounds.left + 15) DIV 16) * 2;

    {number of bytes in BitMap is rowBytes * number of rows}
    {see note at end of Technical Note about using _NewHandle rather than _NewPtr}
    baseAddr := NewPtr(rowBytes * LONGINT(inBounds.bottom - inBounds.top));
  END;
  IF MemError <> noErr THEN BEGIN    {see if we had enough room for the bits}
    SetPort(savePort);
    ClosePort(newPort);              { dump the visRgn and clipRgn }
    DisposPtr(Ptr(newPort));         { dump the GrafPort}
    CreateOffscreenBitMap := false;
  END
  ELSE BEGIN
    { since the bits are just memory, let's erase them before we start }
    EraseRect(inBounds);            {OpenPort did a SetPort(newPort)}
    newOffscreen := newPort;
    SetPort(savePort);
    CreateOffscreenBitMap := true;
  END;
END;
```

Here is the procedure to get rid of an off-screen bitmap created by the previous function:

```
PROCEDURE DestroyOffscreenBitMap(oldOffscreen : GrafPtr);
BEGIN
  ClosePort(oldOffscreen);                       { dump the visRgn and clipRgn }
  DisposPtr(oldOffscreen^.portBits.baseAddr);    { dump the bits }
  DisposPtr(Ptr(oldOffscreen));                  { dump the port }
END;
```

Now that you know how to create and destroy an off-screen bitmap, let's go through the motions of using one. First, let's define a few things to make the `_NewWindow` call a little clearer.

```
CONST
  kIsVisible   = true;
  kNoGoAway    = false;
  kMakeFrontWindow = -1;
  myString     = 'The EYE';  {string to display}
```

Here's the body of the test code:

```
VAR
  offscreen : GrafPtr;    {our off-screen bitmap}
  ovalRect  : Rect;       {used for example drawing}
  myWBounds : Rect;       {for creating window}
  OSRect    : Rect;       {portRect and bounds for off-screen bitmap}
  myWindow  : WindowPtr;

BEGIN
  InitToolbox;                        {exercise left to the reader}

  myWBounds := screenBits.bounds;     { size of main screen }
  InsetRect(myWBounds, 50,50);        { make it fit better }
```

```
 myWindow := NewWindow(NIL, myWBounds, 'Test Window', kIsVisible,
                       noGrowDocProc, WindowPtr(kMakeFrontWindow), kNoGoAway, 0);

 IF NOT CreateOffscreenBitMap(offscreen,myWindow^.portRect) THEN BEGIN
   SysBeep(1);
   ExitToShell;
 END;

 { Example drawing to our off-screen bitmap }
 SetPort(offscreen);
 OSRect := offscreen^.portRect;    { offscreen bitmap's local coordinate rect }
 ovalRect := OSRect;
 FillOval(ovalRect, black);
 InsetRect(ovalRect, 1, 20);
 FillOval(ovalRect, white);
 InsetRect(ovalRect, 40, 1);
 FillOval(ovalRect, black);
 WITH ovalRect DO
   MoveTo((left+right-StringWidth(myString)) DIV 2, (top+bottom-12) DIV 2);
 TextMode(srcXor);
 DrawString(myString);

 { copy from the off-screen bitmap to the on-screen window.  Note that in this
 case the source and destination rects are the same size and both cover the
 entire area.  These rects are allowed to be portions of the source and/or
 destination and do not have to be the same size.  If they are not the same size
 then _CopyBits scales the image accordingly
 }
 SetPort(myWindow);
 CopyBits(offscreen^.portBits, myWindow^.portBits,
          offscreen^.portRect, myWindow^.portRect, srcCopy, NIL);

 DestroyOffscreenBitMap(offscreen);    {remove the evidence}

 WHILE NOT Button DO;                  {give user a chance to see the results}
END.
```

MPW C

First, let's look at a general purpose function to create an off-screen bitmap. This function creates the `GrafPort` on the heap. You could also create it on the stack and pass the uninitialized structure to a function similar to this one.

```
Boolean CreateOffscreenBitMap(GrafPtr *newOffscreen, Rect *inBounds)
{
 GrafPtr savePort;
 GrafPtr newPort;

 GetPort(&savePort);    /* need this to restore thePort after OpenPort */

 newPort = (GrafPtr) NewPtr(sizeof(GrafPort));    /* allocate the grafPort */
 if (MemError() != noErr)
   return false;                /* failed to allocate the off-screen port */
 /*
 the call to OpenPort does the following . . .
   allocates space for visRgn (set to screenBits.bounds) and clipRgn (set wide
   open)
   sets portBits to screenBits
   sets portRect to screenBits.bounds
   etc. (see IM I-163,164)
   side effect: does a SetPort(&offScreen)
```

```
 */
 OpenPort(newPort);
 /* make bitmap the size of the bounds that caller supplied */
 newPort->portRect = *inBounds;
 newPort->portBits.bounds = *inBounds;
 RectRgn(newPort->clipRgn, inBounds);    /* avoid wide-open clipRgn, to be safe  */
 RectRgn(newPort->visRgn, inBounds);     /* in case newBounds is > screen bounds */

 /* rowBytes is size of row, it must be rounded up to an even number of bytes */
 newPort->portBits.rowBytes = ((inBounds->right - inBounds->left + 15) >> 4) << 1;

 /* number of bytes in BitMap is rowBytes * number of rows */
 /* see notes at end of Technical Note about using _NewHandle rather than _NewPtr
    */
 newPort->portBits.baseAddr =
          NewPtr(newPort->portBits.rowBytes * (long) (inBounds->bottom - inBounds-
          >top));
 if (MemError()!=noErr) {   /* check to see if we had enough room for the bits */
   SetPort(savePort);
   ClosePort(newPort);      /* dump the visRgn and clipRgn */
   DisposPtr((Ptr)newPort); /* dump the GrafPort */
   return false;            /* tell caller we failed */
   }
 /* since the bits are just memory, let's clear them before we start */
 EraseRect(inBounds);     /* OpenPort did a SetPort(newPort) so we are ok */
 *newOffscreen = newPort;
 SetPort(savePort);
 return true;             /* tell caller we succeeded! */
}
```

Here is the function to get rid of an off-screen bitmap created by the previous function:

```
void DestroyOffscreenBitMap(GrafPtr oldOffscreen)
{
 ClosePort(oldOffscreen);                       /* dump the visRgn and clipRgn */
 DisposPtr(oldOffscreen->portBits.baseAddr);    /* dump the bits */
 DisposPtr((Ptr)oldOffscreen);                  /* dump the port */
}
```

Now that you know how to create and destroy an off-screen bitmap, let's go through the motions of using one. First, let's define a few things to make the `_NewWindow` call a little clearer.

```
#define kIsVisible true
#define kNoGoAway false
#define kNoWindowStorage 0L
#define kFrontWindow ((WindowPtr) -1L)
```

Here's the body of the test code:

```
main()
{
 char* myString = "\pThe EYE";  /* string to display */

 GrafPtr   offscreen;           /* our off-screen bitmap */
 Rect      ovalRect;            /* used for example drawing */
 Rect      myWBounds;           /* for creating window */
 Rect      OSRect;              /* portRect and bounds for off-screen bitmap*/
 WindowPtr myWindow;

 InitToolbox();                 /* exercise for the reader */
```

```
 myWBounds = qd.screenBits.bounds;  /* size of main screen */
 InsetRect(&myWBounds, 50,50);  /* make it fit better */
 myWindow = NewWindow(kNoWindowStorage, &myWBounds, "\pTest Window", kIsVisible,
                      noGrowDocProc, kFrontWindow, kNoGoAway, 0);
 if (!CreateOffscreenBitMap(&offscreen, &myWindow->portRect)) {
   SysBeep(1);
   ExitToShell();
   }
 /* Example drawing to our off-screen bitmap*/
 SetPort(offscreen);
 OSRect = offscreen->portRect;  /* offscreen bitmap's local coordinate rect */
 ovalRect = OSRect;
 FillOval(&ovalRect, qd.black);
 InsetRect(&ovalRect, 1, 20);
 FillOval(&ovalRect, qd.white);
 InsetRect(&ovalRect, 40, 1);
 FillOval(&ovalRect, qd.black);
 MoveTo((ovalRect.left + ovalRect.right - StringWidth(myString)) >> 1,
        (ovalRect.top + ovalRect.bottom - 12) >> 1);
 TextMode(srcXor);
 DrawString(myString);

 /* copy from the off-screen bitmap to the on-screen window.  Note that in this
 case the source and destination rects are the same size and both cover the
 entire area.  These rects are allowed to be portions of the source and/or
 destination and do not have to be the same size.  If they are not the same size
 then _CopyBits scales the image accordingly.
 */
 SetPort(myWindow);
 CopyBits(&offscreen->portBits, &(*myWindow).portBits,
          &offscreen->portRect, &(*myWindow).portRect, srcCopy, 0L);

 DestroyOffscreenBitMap(offscreen);    /* dump the off-screen bitmap */
 while (!Button());     /* give user a chance to see our work of art */
}
```

Comments

In the example code, the bits of the `BitMap` structure, which are pointed to by the `baseAddr` field, are allocated by a `_NewPtr` call. If your off-screen bitmap is close to the size of the screen, then the amount of memory needed for the bits can be quite large (on the order of 20K for the Macintosh SE or 128K for a large screen). This is quite a lot of memory to lock down in your heap and it can easily lead to fragmentation if you intend to keep the off-screen bitmap around for any length of time. One alternative that lessens this problem is to get the bits via `_NewHandle` so the Memory Manager can move them when necessary. To implement this approach, you need to keep the handle separate from the `GrafPort` (for example, in a structure that combines a `GrafPort` and a `Handle`). When you want to use the off-screen bitmap you would then lock the handle and put the dereferenced handle into the `baseAddr` field. When you are not using the off-screen bitmap you can then unlock it.

This example does not demonstrate one of the more typical uses of off-screen bitmaps, which is to preserve the contents of windows so that after a temporary window or dialog box obscures part of your windows and is then dismissed, you can quickly handle the resulting update events without recreating all of the intermediate drawing commands.

Make sure you only restore the pixels within the content regions of your own windows in case the temporary window partly obscures windows belonging to other applications or to the desktop. Another application could change the contents of its windows while they are behind your temporary window, so you cannot simply restore all the pixels that were behind the temporary window because that would restore the old contents of the other application's windows. Instead, you could keep keep an off-screen bitmap for each of your windows and then restore them by copying each bit map into the corresponding window's ports when they get their update events.

An alternate method is to make a single off-screen bitmap that is as large as the temporary window and a region that is the union of the content regions of your windows. Before you display the temporary window, copy the screen into the off-screen bit map using the region as a mask. After the temporary window is dismissed, restore the obscured area by copying from the off-screen bit map into a copy of the Window Manager port, and use the region as a mask. If the region has the proper shape and location, it prevents `_CopyBits` from drawing outside of the content regions of your windows. See Technical Note #194, WMgrPortability for details about drawing across windows.

In some cases it can be just as fast and convenient to simply define a picture (PICT) and then draw it into your window when necessary. There are cases, however, such as text rotation, where it is advantageous to do the drawing off the screen, manipulate the bit image, and then copy the result to the visible window (thus avoiding the dangers inherent in writing directly to the screen). In addition, this technique reduces flicker, because all of the drawing done off the screen appears on the screen at once.

It is also important to realize that, if you plan on using the pre-Color QuickDraw eight-color model, an off-screen bitmap loses any color information and you do not see your colors on a system that is capable of displaying them. In this case you should either use a PICT to save the drawing information or check for the presence of Color QuickDraw and, when it is present, use a `PixMap` instead of a `BitMap` and the color toolbox calls (*Inside Macintosh*, Volume V) instead of the standard QuickDraw calls (*Inside Macintosh*, Volume I).

You may also want to refer to the OffScreen library (DTS Sample Code #15) which provides both high- and low-level off-screen bitmap support for the 128K and later ROMs. The OffSample application (DTS Sample Code #16) demonstrates the use of this library.

Further Reference:

- *Inside Macintosh*, Volumes I & IV, QuickDraw
- *Inside Macintosh*, Volume V, Color QuickDraw
- Technical Note M.IM.PrincipiaOffscreen — Principia Off-Screen Graphics Environments
- Technical Note M.TB.WMgrPort — WMgrPortability

- DTS Macintosh Sample Code #15, OffScreen & #16, OffSample