Creating Offscreen Bitmaps How to create and draw into off-screen bitmaps

A typical use for an off-screen bitmap is to preserve a part of a window. You would do this, for example, when a dialog box obscures part of your windows and is then dismissed. Having saved an off-screen bitmap, 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 those belonging to other applications or to the desktop. 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.

Another 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 (dialog box, etc.), 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.

In some cases, it can be just as fast and convenient to define a picture ('PICT') and then draw it into your window when necessary. In some cases (such as text rotation), it is advantageous to do the drawing off-screen, manipulate the bit image and then copy the result to the visible window. This technique reduces flicker and avoids writing directly to the screen, which is inherently dangerous.

It is also important to realize that, if you are planning on using the pre-Color Quickdraw eight-color model, an off-screen bitmap loses all color information and you won't see your colors -- even 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. When it is present, use a PixMap instead of a BitMap and the color toolbox calls instead of the standard Quickdraw calls.

This example code shows how to create an off-screen bitmap and display it on the screen.

Example

```
#include <Events.h>
#include <Quickdraw.h>
#include <Windows.h>

#define kIsVisible TRUE
#define kNoGoAway FALSE
#define kNoWindowStorage 0L
#define kFrontWindow ((WindowPtr) -1L)

/* prototypes */
Boolean CreateOffscreenBitMap(GrafPtr *, Rect *);
```

```
void DestroyOffcreenBitMap(GrafPtr);

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;
/* failure to allocate 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 Inside Macintosh Volume 1 pages 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);
    RectRgn(newPort->visRgn, inBounds);
/* rowBytes is size of row, 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 example about using NewPtr instead of NewHandle  */
    newPort->portBits.baseAddr =
        NewPtr(newPort->portBits.rowBytes * (long)(inBounds->bottom -          e
inBounds->top));

    if(MemError() != noErr)
        {
        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;                                /* success */
}
```

```
/*
    DestroyOffscreenBitMap - get rid of an off-screen bitmap created
    by CreateOffscreenBitMap
*/
void DestroyOffcreenBitMap(GrafPtr oldOffscreen)
{
    ClosePort(oldOffscreen);                /* dump the visRgn and clipRgn  */
    DisposPtr(oldOffscreen->portBits.baseAddr);/* dump the bits */
    DisposPtr((Ptr)oldOffscreen);                   /* dump the port */
}

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;

    InitGraf(&thePort);
    InitFonts();
    FlushEvents( everyEvent, 0 );
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs(0L);
    InitCursor();
    MaxApplZone();

    myWBounds = 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, black);
    InsetRect(&ovalRect, 1, 20);
    FillOval(&ovalRect, white);
    InsetRect(&ovalRect, 40, 1);
    FillOval(&ovalRect, 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);

    DestroyOffcreenBitMap(offscreen);      /* dump the off-screen bitmap */
    while(!Button())                       /* give user chance to see output */
        ;
}
```

In the example given above, the bits of the BitMap structure, which are pointed to by the baseAddr field are allocated by a call to **NewPtr**. If your off-screen bitmap is close to the size of the screen, then the amount of memory 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 period of time. One alternative that lessens this problem is to get the bits via **NewHandle** so that 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, unlocking the handle when it is not is use.