Technotes

Download Acrobat file (K)	Download PDD file (126K)	Download AppleWorks file (42K)

QuickDraw GX OffscreenLibrary.c in Detail: Description, Uses & Limitations

Technote 1038 **MARCH 1996**

This Technote discusses OffscreenLibrary.c from the QuickDraw GX Libraries.

This Note is intended for Macintosh QuickDraw GX developers who implement flickerless drawing or double buffering using OffscreenLibrary.c or who are considering using it for their QuickDraw GX graphics applications.

Contents

Important for all Apple Printing and Graphics Developers:

The information in this Technote is still relevant up to and including Mac OS 7.6 with QuickDraw GX 1.1.5. Beginning with the release of Mac OS 8.0, however, Apple plans to deliver a system which incorporates QuickDraw GX graphics and typography **only**. QuickDraw GX printer drivers and GX printing extensions will **not** be supported in Mac OS 8.0 or in future Mac OS releases. Apple's goal is to simplify the user experience of printing by unifying the Macintosh graphic and printing architectures and standardizing on the classic Printing Manager.

For details on Apple's official announcement, refer to </technotes/gxchange.html>

About the GX Libraries

For better or worse, the development of QuickDraw GX took seven years from conception to initial release. During that time, there were many requests for feature enhancements and interface improvements that, if implemented, might have taken seven more years to complete. As it turns out, some of these enhancements could readily be built on existing services, but there was no time to test or document these services with the rigor required to make them fully part of the released system.

The GX Libraries fill this gap by providing services built on top of the rest of GX in source form. This Technote and others document these services. Since GX libraries are provided as source, it is reasonable for developers to modify them to meet their specific needs. Care was taken for the libraries not to depend on the implementation details of GX, so that future versions of GX should not invalidate them, in original or modified form.

The libraries are likely to evolve to take advantage of improved algorithms, new Macintosh or GX services; if you modify one for your application's specific needs, it's worth occasionally reviewing the GX library provided by Apple to stay synchronized with any improvements.

What are Offscreens?

Slick graphics applications attempt to draw animations seamlessly to the screen, without flashing or flickering. QuickDraw GX provides a number of strategies that change these distractions into attractions. The most popular method to eliminate flickering is double buffering; the application draws into one bitmap while the computer displays a second bitmap. The bitmap receiving the drawing that will be displayed momentarily is called an offscreen.

What is in OffscreenLibrary.c?

The GX Library, OffscreenLibrary.c, provides utility functions to implement double buffering; it creates bitmaps that in turn are imaged by GX Graphics. It was written primarily by Oliver Steele, with contributions from the rest of the GX Graphics team.

OffscreenLibrary.c has two distinct groups of functions. The simpler revolves around the offscreen struct, and provides a single bitmap for back buffering. The other set uses a viewPortBuffer to support multiple offscreens that correspond to the portions of a window that spans multiple monitors of different depths.

Using OffscreenLibrary.c

The One Shot Solution: Struct offscreen

The meat of Offscreenlibrary.c is this struct:

```
struct offscreen {
        gxShape         draw;   /* a bitmap which, when drawn, transfers the
                                        offscreen to the display */
        gxTransform     xform;  /* this causes shapes owning it to draw
                                        offscreen. */
        gxViewDevice    device; /* the offscreen device whose colorSpace,
                                        etc. you may change */
        gxViewPort      port;   /* the offscreen port which may be put in
                                        any transform's viewPort list */
        gxViewGroup     group;  /* the global space in which the viewPort
                                        and viewDevice exist */
};
```

and these functions:

```
        void CreateOffscreen(offscreen *target, gxShape bitmapShape);
```

Use CreateOffscreen to fill in the offscreen struct, given the bitmap shape to back up.

```
        void DisposeOffscreen(offscreen *target);
```

When you're through with the offscreen, use DisposeOffscreen to get rid of the pieces.

```
        void CopyToBitmaps(gxShape target, gxShape source);
```

To copy one bitmap to another, use CopyToBitmaps.

GX makes it pretty darn easy to create an offscreen. For instance, you can create a bitmap that contains a diagonal line with these calls:

```
        gxLine aLine = {ff(20), ff(40), ff(60), ff(80)};
        gxShape lineBits = GXNewLine(&aLine);
        GXSetShapeType(lineBits, gxBitmapType);
```

Then, to create an offscreen from the line bitmap:

```
        offscreen offLine;
        CreateOffscreen(offLine, lineBits);
```

You can draw the line bitmap with:

```
        GXDrawShape(offLine.draw);
```

To add a rectangle to the line bitmap, first create a rectangle:

```
        gxRectangle aRect = {ff(50), ff(50), ff(60), ff(60)};
        gxShape rectToAdd = GXNewRectangle(&aRect);
```

Then change the rectangle to the transform in the offscreen:

```
        GXSetShapeTransform(rectToAdd, offLine.xform);
        GXDrawShape(rectToAdd);
```

Now, drawing the line bitmap will draw both the line and the rectangle:

```
                GXDrawShape(lineBits);
```

Once you're done, you can use DisposeOffscreen to get rid of it:

```
        DisposeOffscreen(&offLine);
```

The function CopyToBitmaps uses the offscreen structure internally to copy one bitmap onto another. The name is somewhat misleading, since the shape to be copied can be any shape type, not necessarily a bitmap. For instance, you can use it to create a bitmap that has a specific bit depth from a picture:

```
        static gxShape Create8BitPicture(gxShape myPicture)
        {
                gxRectangle bounds;
        // get the bounding box of the picture
                GXGetShapeBounds(myPicture, 0, &bounds);
        // move the picture so that itÕs upper left corner is at (0, 0)
                GXMoveShape(myPicture, -bounds.left, -bounds.top);
        // create a bitmap big enough to hold the picture
                gxShape bitmap = {nil, FixRound(bounds.right - bounds.left),
                        FixRound(bounds.bottom - bounds.top), 0, 8, nil, nil, nil};
        // copy the picture to the bitmap
                CopyToBItmaps(bitmap, myPicture);
        // move the bitmap to the pictureÕs original position
                GXMoveShape(myPicture, bounds.left, bounds.top);
        // restore the pictureÕs original position
                GXMoveShape(bitmap, bounds.left, bounds.top);
                return bitmap;
        }
```

The ViewPortBuffer Multiple Offscreen Scheme

The Macintosh is relatively unique among computers in that it allows windows to straddle two or more monitors at the same time. QuickDraw GX fully embraces this capability, and takes it to the logical extreme; not only can viewPorts cross multiple viewDevices, but the viewDevices themselves can overlap each other.

This makes allocating an offscreen bitmap a challenge, since there may be no single best depth that allows drawing to contain the correct amount of color and draw as quickly as possible. The solution provided by a viewPortBuffer creates a picture containing an array of offscreens that match the desired multiple viewDevices.

Here's the interface to viewPortBuffer.

```
        typedef struct viewPortBufferRecord **viewPortBuffer;
```

The viewPortBuffer is a blind handle that points to the internals kept by these routines. It is never necessary to directly access the fields pointed to by this handle.

```
        viewPortBuffer NewViewPortBuffer(gxViewPort originalPort);
```

To create an offscreen for a window that may cross multiple monitors, call NewViewPortBuffer. It takes the window's viewPort, returns a reference to the internal structure. The window's viewPort can be retrieved from GXGetWindowViewPort.

```
        void DisposeViewPortBuffer(viewPortBuffer target);
```

When the window is closed, call DisposeViewPortBuffer to get deallocate the internal objects allocated by the viewPortBuffer.

```
        gxViewPort GetViewPortBufferViewPort(viewPortBuffer source);
```

To draw shapes into the offscreen, first call GetViewPortBufferViewPort. GetViewPortBufferViewPort returns a viewPort that references the multiple offscreens. Drawing into this viewPort draws into as many offscreen bitmaps as is appropriate. To attach this viewPort to a single shape, use the library routine SetShapeViewPort. To change all shapes of a given type, try `SetTransformViewPort( GXGetDefaultTransform( theType ));`

```
        gxShape GetViewPortBufferShape(viewPortBuffer source);
```

To draw the offscreens, call GetViewPortBufferShape to get the shape to draw. Drawing the returned shape transfers the offscreen bitmaps to the viewDevices pointed to by the original viewPort, typically the window's viewPort.

```
        Boolean ValidViewPortBuffer(viewPortBuffer target);
```

The user may foul things up by changing the monitors depth or the window's position. After a window-altering event, call ValidViewPortBuffer to see if the viewPortBuffer needs to be recomputed.

```
        Boolean UpdateViewPortBuffer(viewPortBuffer target);
```

If the viewPortBuffer is out of date, UpdateViewPortBuffer will put things right again. It returns true if the viewPortBuffer was already valid.

Here's a convoluted example that builds the offscreens and draws a shape.

```
static void BufferDraw(gxShape shape, WindowPtr window)
{
// create the viewPortBuffer from the viewPort associated with the window
        viewPortBuffer buffer =
                NewViewPortBuffer(GXGetWindowViewPort(window));
// retrieve the viewPort created that allows drawing into the offscreen
        viewPort offscreenPort = GetViewPortBufferViewPort(buffer);
// point the shape to that offscreen
        SetShapeViewPort(shape, offscreenPort);
// draw the shape into the offscreen
        GXDrawShape(shape);
// draw the offscreen into the window
        GXDrawShape(GetViewPortBufferShape(buffer));
// throw the offscreen away
        DisposeViewPortBuffer(buffer);
}
```

How the ViewPortBuffer Works

Since the viewPortBuffer is implemented as a library, you can read the code yourself; you'll find it is pretty straightforward.

The implementation is split into a few steps:

1. Figure out which devices the window/viewPort crosses.

2. For each device, figure out the coordinates for the viewPort on that device.

3. Create a bitmap that has the same pixel depth, color set, color profile and color space as the corresponding device.

4. Keep track of the allocations and object references so that closing the window (or disposing the

device) doesn't leave any dangling references or pointers.

Summary

GX Libraries contain a wealth of information and show how to use QuickDraw GX to solve real problems. OffscreenLibrary.c shows how to use GX to construct flickerless drawing by implementing double buffering on a single device or on multiple devices.

- MacOS SDK CD, Development Kits (Disc 1): QuickDraw GX: Programming Stuff: GX Libraries:
- *Inside Macintosh: QuickDraw GX Objects*
- *Inside Macintosh: QuickDraw GX Environment and Utilities*