Technical Note TN2051
Mac OS X QuickDraw Performance

CONTENTS

The windowing system on Mac OS X is different from that of previous Mac OS versions in several fundamental ways. The primary difference where QuickDraw is concerned is that drawing in a window port ends up in the window's back buffer and not on the screen. This shift and the underlying changes in QuickDraw necessary to make it possible have affected the performance characteristics of QuickDraw drawing code in some new and interesting ways. This technote will discuss some of the trouble spots and explain how to avoid them.

[Feb 13 2003]

Avoid triple buffering

Because drawing occurs in the window back buffer and not directly on the screen, most of the motivation for doing your own double buffering is no longer present. In the past, it was sometimes necessary to composite your drawing into an offscreen `GWorld` and then copy it onto the screen in one `CopyBits` operation in order to avoid flickering and tearing. With Mac OS X, that extra buffer is not only unnecessary, it will actually degrade performance due to the extra copy required to move content from your offscreen to the window back buffer. Don't double buffer on Mac OS X.

Back to top

Simplify dirty region updates

In order for QuickDraw to know which parts of the window back buffer to flush to the screen, every call to QuickDraw needs to record the region it dirties. When a drawing sequence consists of a number of small items contained within a well-known larger area, it is usually best to dirty the whole region at once rather than having every QuickDraw routine incrementally add to the dirty region. This way, you short-circuit the dirty region updating code inside each QuickDraw routine.

Assuming all the small QuickDraw operations are enclosed in a bigger rectangular region (`bigRectRegion`), calling `QDSetDirtyRegion(port, bigRectRegion)` before making the first QuickDraw drawing operation dirties the whole rectangle at once, eliminating the need for the smaller QuickDraw calls to update the dirty region.

Setting the dirty region to a large rectangular region accomplishes two goals. First, it makes sure that the dirty region is rectangular, which greatly simplifies any subsequent region operations. Second, the large region encompasses all of the regions dirtied by the individual QuickDraw drawing calls, thus turning their dirty region updates into trivial intersection checks of the regions' bounding rectangles.

This technique is particularly handy for speeding up the drawing of complex vector graphics.

Back to top

Short-circuit QuickDraw's implicit LockPortBits

On Mac OS X, your window's back buffer is shared between your application and the window server. In order for your application to access the back buffer contents, the window server must be locked out temporarily. Because QuickDraw primitives are typically not bracketed by any sort of begin/end sequence, each QuickDraw routine needs to make sure the port bits are locked by implicitly calling `LockPortBits` before drawing. When you make a large number of consecutive QuickDraw calls, the cost of these implicit `LockPortBits` can add up significantly.

Luckily, calls to `LockPortBits` are nestable, thus allowing you to avoid having every QuickDraw routine lock and unlock the port bits by bracketing your entire routine sequence with a `LockPortBits/UnlockPortBits` pair:

`LockPortBits` calls can be nested and `LockPortBits` is smart enough to not redo all it's work if the port bits have already been locked, so create a nesting `LockPortBits` bits by bracketing your entire routine sequence with a `LockPortBits/UnlockPortBits` pair to speed up the intervening QuickDraw calls:

```
LockPortBits(GetWindowPort(window))
//  .. your QD drawing sequence ...
UnlockPortBits(GetWindowPort(window));
```

This ensures that the port bits are locked only once for the entire sequence of QuickDraw calls and the nested implicit `LockPortBits/UnlockPortBits` only increment/decrement a lock count. This way your app incurs the cost of locking the bits once for the entire sequence instead of once per QuickDraw call, eliminating a large performance bottleneck.

Note:
It is important to not keep the port bits locked for more than a second or two at the very most. The rest of the system doesn't appreciate having the lock maintained any longer than absolutely necessary. If your drawing sequence will take longer than a second or two, you will need to break up the sequence into separate segments, each surrounded by its own `LockPortBits/UnlockPortBits` pair.

WARNING:
You must not call any QuickTime APIs while using the port locking technique below. QuickTime's locking logic is much more complicated than QuickDraw's. All sorts of bad things can and will happen if you use `LockPortBits/UnlockPortBits` around calls to QuickTime.

Back to top

Don't flush to the screen more often than necessary

For most applications, there is no need to flush more than 30 frames per second to obtain smooth animation. Flushing more often only results in additional CPU overhead and may end up slowing down the desired animation.

To control when flushing occurs you want to use a time-based strategy aimed at providing 30 fps:

```
    /* for about 30 frames/sec */
const UInt32 kMinNanosecsBetweenFlushes = 1E9 / 30;

static AbsoluteTime sLastFlush = { 0, 0 };

....

AbsoluteTime curTime = UpTime();
Nanoseconds  delta = AbsoluteDeltaToNanoseconds( curTime, sLastFlush );
```

```
if ( U64Compare( UnsignedWideToUInt64(delta),
          U64SetU(kMinNanosecsBetweenFlushes)) > 0)
{
  sLastFlush = curTime;
  QDFlushPortBuffer(port, NULL);
}
```

Back to top

Summary

Mac OS X introduced some new performance characteristics for QuickDraw drawing, but the information presented here will help developers eliminate their graphics performance bottlenecks.

Back to top

Downloadables

 Acrobat version of this Note (36K) Download

Back to top
