Technical Note QD15
RowBytes Revealed II

CONTENTS

This Technical Note discusses the maximum rowBytes value for a `pixMap`.

[May 01 1993]

Introduction

This technical note concerns applications which create and read very large `pixMaps`. The `rowBytes` field defines the limits of the largest `pixMap`. For applications which use 32-Bit QuickDraw, this limit is well-defined: `0x3FFE`. However, a few applications want to go beyond this limit, and under some extremely controlled situations, this limit can be extended to `0x7FFE`. This note explains why the limit exists, and proposes a few difficult methods to exceed this limit. Along the way, proper parameter passing for `CopyBits` is discussed and a few debugging tips are thrown in.

Back to top

Largest size of rowBytes of a pixMap

`rowBytes` is important because it defines the size of a `pixMap`. `rowBytes` had a largest size of `0x1FFE` up through Color QuickDraw. The top three bits were reserved.

Version 1.0 of 32-Bit QuickDraw extended that limit to `0x3FFE`: only the top two bits are reserved. QuickDraw uses the two bits for identification purposes. The top bit distinguishes between a pixMap and a `bitMap`. If the top bit of `rowBytes` is set, then QuickDraw knows the structure is a pixMap. The second highest bit is used by `CopyBits, CopyMask, CopyDeepMask, SeedCFill, and CalcCMask` to distinguish between their `bitMap` parameters. The implications of this identification scheme are fully discussed below.

Back to top

3 Types of BitMap Parameters

CopyBits, CopyMask, CopyDeepMask, SeedCFill, and CalcCMask are routines which take bitMaps as their first two parameters. For historical and compatibility reasons, three types of parameters can be passed in: bitMap, pixMaps and the portBits field of a cGrafPort.

For the purposes of this note, I will focus on the CopyBits call, though my discussion will apply to the other four routines. The behavior of these five calls is identical for the bitMap parameters.

The BitMap

The bitMap parameter is actually a pointer to a bitMap. In Pascal, the pointer is implicit, since a bitMap structure is greater than 4 bytes in size and the Pascal compiler creates a pointer for any data structure greater than 4 bytes.

```
In C:
pascal void CopyBits(const BitMap *srcBits,const BitMap *dstBits,const Rect
    *srcRect, const Rect *dstRect,short mode,RgnHandle maskRgn) = 0xA8EC;

In Pascal:
PROCEDURE CopyBits(srcBits: BitMap;dstBits: BitMap;srcRect: Rect;dstRect: Rect;
    mode: INTEGER;maskRgn: RgnHandle);
    INLINE $A8EC;
```

The PixMap

With Color QuickDraw, CopyBits accepts pixMaps as BitMap parameters. This support, however, does not come without a cost. CopyBits needs a method to distinguish between the two parameters. As mentioned above, the top bit of rowBytes distinguishes between pixMaps and bitMaps. While this bit allows pixMaps to be passed in to CopyBits, the size of rowBytes is diminished. All design choices have trade-offs.

The portBits of a CGrafPort

In designing Color QuickDraw, certain rules were set up to provide compatibility with the new features. Here was one rule: you should be able to pass a portPixMap field to CopyBits just as you would pass a portBits field. Unfortunately, portPixMap is a handle whereas portBits is a BitMap. Tricky type-casting, however, allows one to pass in a cGrafPort's portPixMap field, as the following examples show. You must coerce the cGrafPtr in to a grafPtr in order to use the portBits field.

In C:

```
    CGrafPtr   colorPort;   /* Graphics environment for color off screen */
    GrafPtr    savedPort;   /* Pointer to the saved graphics environment */

    CopyBits(     &((GrafPtr)colorPort)->portBits, &savedPort->portBits,
                        &colorPort->portRect, &savedPort->portRect,
                        srcCopy, nil     );
```

In Pascal:

```
    colorPort:   CGrafPtr;   {Graphics environment for color off screen}
    savedPort:   GrafPtr;    {Pointer to the saved graphics environment}

    CopyBits(    GrafPtr(colorPort)^.portBits, savedPort^.portBits,
            colorPort^.portRect, savedPort^.portRect,
            srcCopy, NIL);
```

Typecasting is not the only trickery involved here. CopyBits is faced with a major headache if you think about it. PortPixMap is a handle, and the parameters of CopyBits expect a pointer to a pixMap. To solve this, CopyBits has a little algorithm in it which dereferences the pointer twice if you pass in a portBits. To identify the type of pointer it is getting, CopyBits looks at the top two bits of the rowBytes field. As mentioned above, the top bit distinguishes between a pixMap and a bitMap. The next bit tells CopyBits if the pixMap is part of a cGrafPort or not (see Fig. 1). If the two bits are set, then CopyBits knows it is being passed a pixMapHandle, and it will perform the dereference.

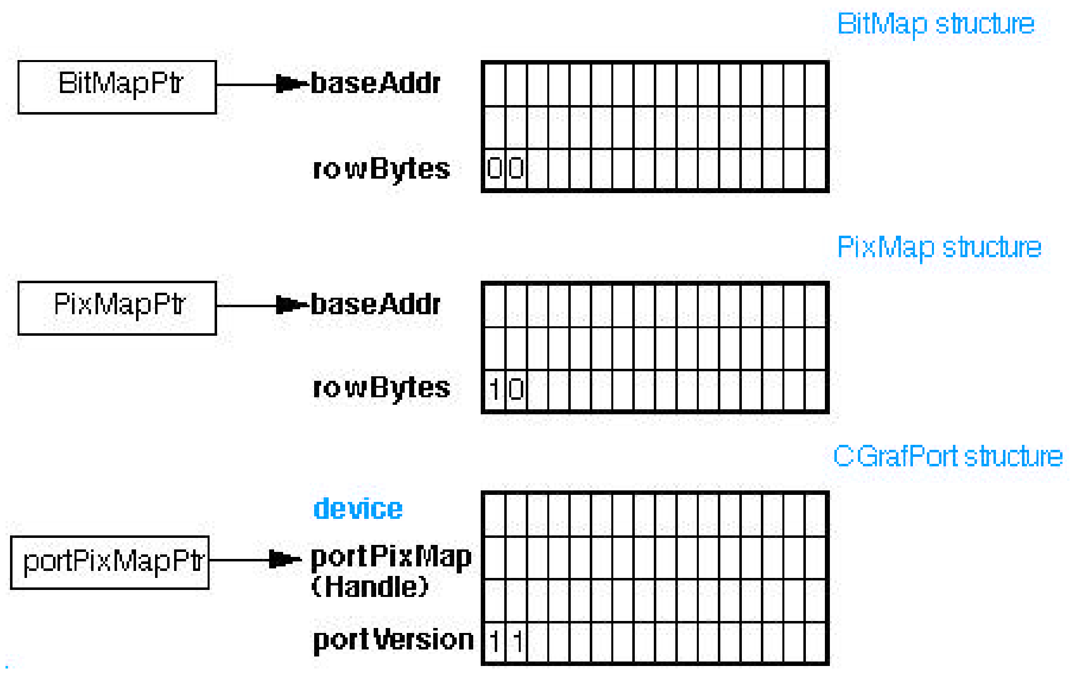

Figure 1. CopyBit's BitMap parameters

Figure 1. displays three types of CopyBit's BitMap parameters. Each points to a either a bitMap, pixMap, or the portPixMap field in a cGrafPort. After 4 bytes, the top two bits of the fifth byte identifies the pointer.

Back to top

Why Large PixMaps Crash Apps

As you can see, the top two bits of rowBytes have specific functions. The highest bit distinguishes between bitMaps and pixMaps. The second highest bit identifies the type of BitMap parameter. If it is set, a dereference is applied; if not, nothing happens to the pixMap. If your pixMap uses the second highest bit of rowBytes and you pass it into a QuickDraw application, it will think it is part of a cGrafPort, and will perform a handle dereference on your baseAddr. The first dereference will get to video memory, but the second dereference will be on whatever random video data happens to be there. Your application will land up in Never-Never land.

If you create pictures which can be redistributed, leave the top two bits of rowBytes alone. Below, I will identify one method of going around this limit, but it will only work in specific situations. If you read pictures, look below for reading pictures which go around this limit.

Back to top

Identifying CGrafPorts and GWorlds in a Debugger

You'll notice in Figure 1 that the portVersion field of a cGrafPort coincides with the location of the rowBytes field of a grafPort. Remember, a cGrafPort has the same size as a grafPort. During debugging, you can use the same information which CopyBits uses to identify cGrafPorts.

If you use a grafPort template to display memory for an unknown grafPort, you can tell if it is a cGrafPort because the rowBytes will be equal to 0xC000. The 0xC corresponds to the two high bits being set in the portVersion field of a cGrafPort. Since these bits can not be set in a grafPort, you know you have a cGrafPort. In addition, if the bottom bit of the portVersion field is set, then it is a gWorld. Thus, if your rowBytes field has a value of 0xC001, then you know you have a gWorld.

Even or odd rowBytes?

Since the dawn of Macintosh, it has been said that rowBytes should be even because each row of a pixMap must contain an integral number of words. Actually, rowBytes has to be even because QuickDraw accesses bitmap data using word or long operands, and these generate address errors when it references an odd address on the 68000, which would happen if rowBytes is odd. The 68020 and later handle odd addresses fine, and so rowBytes can be odd. But, it is still recommended that rowBytes be even, because misaligned accesses incur a performance penalty.

Back to top

Going Around The Limit Today

In the world today applications have a difficulty because they can either refuse to create pixMaps as big as users want or cause crashes by confusing CopyBits into dereferencing the base address of the pixMap if rowBytes exceeds the established limit of less than 0x4000. Some well-known image applications create such pixMaps with bad results. Reading large pixMaps causes the crash, but applications which create them are the ultimate culprit.

As far as a solution for the present, a possibility is to bypass the CopyBits dereferencing algorithm. You can call StdBits directly since it does not mind dealing with larger than legal rowBytes. The problem here is that the destination is implied and the application has to make sure that everything is alright. Also, if the destination spans multiple devices, the application has to divide the task, targeting each device at the time. See the DeviceLoop procedure in IM VI for ideas on this.

A second possibility is to patch CopyBits in situations where you know it can only be fed pixMaps. DrawPicture time is one example. You need to patch right before DrawPicture because you know a picture will contain only pixMaps. That is, you know CopyBits will not be passed portBits. If rowBytes is too big, then the application could split the job, banding the image vertically until the resulting rowBytes values fall within range. After the DrawPicture call, you will need to unpatch.

Back to top

Going Around The Limit Tomorrow

All this is known by engineering and some future directions are already being studied, for example it is possible that a next release of QuickDraw will support pixMap with a rowBytes constant value indicating that the real rowBytes is contained in the planeBytes field instead; I am sure you can immediately think of cases where this is also going to cause problems but we think that the problems are less important than the limitation being overcome.

Back to top

Conclusion

The limitations of rowBytes is becoming an increasingly painful thing, applications can easily create pixMaps (and PICTs) that exceed the limit of 0x4000. It is possible for an application to patch CopyBits in order to work around this limitation but the application writer has to decide what is appropriate for each set of conditions. Thus, rowBytes has traditionally been said to have a maximum value of 0x3FFE. But, if your application avoids the use of CopyBits, CopyMask, and CopyDeepMask, then you can use a rowBytes value of 0x7FFF without harm. However, those situations are rare, and, for all practical matters, the limit of rowBytes is 0x3FFE.

Back to top

References

Inside Macintosh , Volume I, QuickDraw

Inside Macintosh , Volume V & VI, Color QuickDraw

develop 1, "Realistic Color For Real-World Applications"

Back to top

Downloadables

📄	Acrobat version of this Note (K).	Download

Technical Notes by Date | Number | Technology | Title
Developer Documentation | Technical Q&As | Development Kits | Sample Code