SeedFill Calculate a mask for use in CopyMask

#include <Quickdraw.h> **Quickdraw**

void **SeedFill(***srcPtr*, *destPtr*, *srcRowBytes*,*destRowBytes*, *height*, *wrdsWide*, *seedH*, *seedV* **)**;

Ptr	*srcPtr* ;	address within a BitMap of place to start calculating
Ptr	*destPtr* ;	address within a BitMap of where to store 1s and 0s
short	*srcRowBytes* ;	source bitmap rowBytes
short	*destRowBytes* ;	destination bitmap rowBytes
short	*height* ;	height of src and dest rectangle, in pixels
short	*wrdsWide* ;	width of src and dest rectangle, in 16-bit words
short	*seedH* ;	horizontal offset, in pixels, to start pouring
short	*seedV* ;	vertical offset, in pixels, to start pouring

SeedFill examines a portion of a source bitmap and fills a portion of a destination bitMap with 1s. It finds an enclosed area surrounding a specified point in the source, and floods that area in the destination (as in the lasso tool of many paint programs) with 1s (black paint). Use this function as one step in implementing a "paint bucket" tool.

srcPtr is the address of a 16-bit word *inside* a bitMap data area. **SeedFill** will use this as if it were the upper left corner of a rectangle, as defined by *height* and *wrdsWide*. It will examine this implied rectangle as it floods portions of the destination bitMap.

destPtr is the address of a 16-bit word *inside* a bitMap data area. **SeedFill** will use this as if it were the upper left corner of a rectangle, as defined by *height* and *wrdsWide*. It will fill all or part of this "rectangle" with 1s.

Note: Both *srcPtr* and *destPtr* must point to even (word) addresses.

srcRowBytes and . . .
destRowBytes are the widths of the BitMap into which *srcPtr* and *destPtr*, respectively point; i.e., the function will add this value to its current address pointer to move "down one line" in the bitMap.

height is the height, in pixels, of both the source data area and the destination area.

wrdsWide is the width, **in 16-bit words**, of both the source and destination data area.

seedH and...
seedV identify the point to start flooding. These are offsets, in pixels, from the boundary of the implied rectangle defined by *srcPtr* , *height*, and *wrdsWide*.

Returns: none

Notes: Use **SeedFill** to flood an area of a destination bitMap with black paint (i.e., 1s). The flooded area will match the inside and boundary of a section of the source that is enclosed by black pixels. Note that *destPtr* and *srcPtr*

must not be the same (nor may their data areas overlap): image flooding always requires some temporary storage since one of **SeedFill's** first step is to clear the data starting at *destPtr*.

Flooding an area with some other pattern requires that you take the intermediate steps of using **SeedFill** to create a mask, creating an bitmap filled with the desired pattern, and using **CopyMask** and **CopyBits** to move the data around. See the example, below.

Calls to **SeedFill** are not recorded in a Picture definition. See **OpenPicture**.

SeedFill's parameters of this call are rather obscure but you can find some related information in **CalcMask**. In particular, note that *seedH* and *seedV* are not necessarily in either global or local coordinates. They are a count of pixels from the start of the data area defined by the *srcPtr* as offset from the bounds of a "rectangle" implied by *height* and *wrdsWide*.

> **Note**: When running on 256K ROMs, you may wish to use the more flexible **SeedCFill** function.

The following example is very extravagant memory-wise since it allocates three data areas, each the size of the current window (a 400x400 window requires about 20K of storage). However, by doing so, it avoids a series of complex address calculations and coordinate conversions.

Example

```
#include <Quickdraw.h>

void SeedFillTest()
{
    Rect        r, winRect;
    Point       seedPt;
    BitMap      realBits;
    BitMap      srcBits, destBits, patBits;

    SetPort( theWindow );

    winRect = theWindow->portRect;

    realBits = theWindow->portBits;        /* the visible screen */

    /* --- this does not check for not-enough-memory errors
    */
    NewBitMap( &srcBits, &winRect );       /* to hold copy of source */
    NewBitMap( &destBits, &winRect );      /* receives the flooded mask */
    NewBitMap( &patBits, &winRect );       /* pattern to filter via CopyMask */
    SetPortBits( &patBits );               /* create that pattern */
    FillRect( &winRect, gray );
    SetPortBits( &realBits );

    /* -------- draw two concentric rectangles ------
    */
```

```
    EraseRect( &winRect );
    SetRect( &r, 20,20,100,100 ); FrameRect( &r );
    SetRect( &r, 50,30,80,80 ); FrameRect( &r );
                                        /* seed is inside of outer, outside of inner */
    seedPt.h = 25; seedPt.v = 25;       /* we can use local coords in this case */

    /* -------- duplicate visible window into srcBits ------
    */
    CopyBits( &realBits, &srcBits, &winRect, &winRect, srcCopy, 0 );

    seedPt.h = 25; seedPt.v = 25;       /* can use local coords in this case */

    /* -------- create mask with 1s wherever the paint floods ------
    */
    SeedFill( srcBits.baseAddr, destBits.baseAddr,      /* srcPtr , destPtr */
            srcBits.rowBytes, destBits.rowBytes,        /* src/destRowBytes */
            destBits.bounds.bottom - destBits.bounds.top, /* height */
            (destBits.rowBytes + 1) /2,                 /* wrdsWide */
            seedPt.h, seedPt.v );                       /* seedH, seedV */
    /* -------- copy pattern, through mask, into srcBits ------
    */
    CopyMask(  &patBits, &destBits, &srcBits,
                &winRect, &winRect, &srcBits.bounds );

    /* --------- copy srcBits into visible video memory ------
    */
    CopyBits( &srcBits, &realBits, &winRect, &winRect, srcCopy, 0 );

    DisposeBitMap( &srcBits );
    DisposeBitMap( &destBits );
    DisposeBitMap( &patBits );
}

/* ------------- allocate storage for and setup  a bit map -------------
  * returns 0 on allocation error
 */
Ptr NewBitMap(BitMap theBits, Rect *rp )
{
    theBits->rowBytes = ((rp->right - rp->left + 15) / 16) * 2;
    theBits->baseAddr =NewPtr(theBits->rowBytes * (rp->bottom-rp->top));
    theBits->bounds = *rp;

    return( theBits->baseAddr );
}
/* ------------- allocate storage for and setup  a bit map -------------
  * returns 0 on allocation error
  */
DisposeBitMap( BitMap theBits )
{
    return ( DisposPtr( theBits->baseAddr ) );
}
```