WHAT IS RASTEROP?

One of the most useful operations in image composition, image transformation and image analysis is the **rasterop**. This is the low-level operation that your window system uses to paint bitmap characters to your frame buffer, and to paint and repaint windows when they are generated, resized, or moved.

This operation has a long and distinguished history, with the driving force being the desire to use bitmapped graphics terminals for realtime I/O. According to Eric Raymond's *New Hacker's Dictionary* (1991, MIT Press), the PDP-10 had a BLT (BLock Transfer) machine instruction to copy or move a contiguous block of data within memory. (This is not to be confused with the common BLT (Branch if Less Than zero) assembly instruction, or the BLT (Bacon Lettuce and Tomato) sandwich.) Suppose you have an image that is represented by a contiguous block of memory in line raster order, but you want to copy or move data between rectangular sub-blocks. In general, this data will be composed of disconnected fragments of the original block of data. One version of rasterop that worked on rectangular two-dimensional subimages of the display buffer was implemented at Xerox PARC in the early 1970s by Dan Ingalls.

This utility was used to display character bitmaps on the Alto, a 16-bit computer derived from the Data General Nova minicomputer. At Xerox, the operation was called *Bitblt*, pronounced "bit blit", which is short for "Bit Block Transfer." (The Alto displays were 1 bit/pixel, monochrome.) Many implementations were subsequently written for other computers with bitmapped graphics displays, such as Sun's Unix workstations dating from the early 1980s that used Sun's proprietary *Sunview* window system.

For efficiency, some of the display bitblt operations need to work *in place*, so that the display buffer is both the source and the destination of the pixels. In such situations, bits have to be copied in a particular order so as not to overwrite bits that need to be moved later. Some bitblt (rasterop) operations do not use the display buffer as either the source or destination of the bits. Such memory-to-memory operations are used, for example, by *imagers* that compose page images for display or printing, or, as we shall see, for many image processing operations in general.

There are two well-tested versions of rasterop in open source that I know of: one with X windows and one with ghostscript. However, these are both complicated by internal reference to graphical display imaging, and they have a very large set of operators.

They do not have a clean interface to the low-level bit manipulation routines, and they are encumbered by Gnu's GPL ``copyleft'' (for X) and by Aladdin's own copyleft (for ghostscript). For these reasons, I have made a set of efficient low-level routines for rasterop, with a clean interface between the user's image data structure and low-level data. If you want to use a different image data structure, it is a simple matter to rewrite the interface shim to the low-level operations.

The general binary rasterop function replaces a rectangular part of a destination (*dest*) image with a bitwise combination of the pixels in the *dest* and those in an arbitrary rectangle of the same size in a source (*src*) image. The bitwise combination can be one of the following:

```
    PIX_SRC                              s
(replacement)
    PIX_NOT(PIX_SRC)                    ~s
(replacement with bit inversion)
    PIX_SRC | PIX_DST                    s | d
    PIX_SRC & PIX_DST                    s & d
    PIX_SRC ^ PIX_DST                    s ^ d
    PIX_NOT(PIX_SRC) | PIX_DST          ~s | d
    PIX_NOT(PIX_SRC) & PIX_DST          ~s & d
```

```
PIX_NOT(PIX_SRC) ^ PIX_DST          ~s ^ d

PIX_SRC | PIX_NOT(PIX_DST)           s | ~d

PIX_SRC & PIX_NOT(PIX_DST)           s & ~d

PIX_SRC ^ PIX_NOT(PIX_DST)           s ^ ~d

PIX_NOT(PIX_SRC | PIX_DST)          ~(s | d)

PIX_NOT(PIX_SRC & PIX_DST)          ~(s & d)

PIX_NOT(PIX_SRC ^ PIX_DST)          ~(s ^ d)
```

where the version on the left uses the macros that Sun originally defined around 1981 for their Pixrect library, and the version on the right is an obvious shorthand. The first two are independent of the data in the *dest*, but they are "binary" in the sense that the low-level functions need to index into both the *src* and *dest* arrays. Of these 14 operations, only 12 are unique. The following three operations are identical:

```
PIX_NOT(PIX_SRC) ^ PIX_DST          ~s ^ d

PIX_SRC ^ PIX_NOT(PIX_DST)           s ^ ~d

PIX_NOT(PIX_SRC ^ PIX_DST)          ~(s ^ d)
```

In addition to the 12 unique binary operations, there are two unary rasterop functions that operate on the *dest* and only depend on the *dest*:

```
    PIX_DST                                 d    (this is a
no-op)
    PIX_NOT(PIX_DST)                       ~d    (bit
inversion)
```

And finally there are two unary operations that operate on the *dest* and are independent of both *src* and *dest*:

```
    PIX_CLR                                 (set all bits
to 0)
    PIX_SET                                 (set all bits
to 1)
```

In all, there are a total of 16 unique boolean operations. The Sun definition of these macros, along with an analysis of how they work properly in composition, is given in `rop.c`.

Before we go into any details, let me explain some of the things besides image composition that rasterop can do.

Source: http://www.leptonica.com/rasterops.html