Bit-Mapped Graphics

When an object or design is too complicated to draw with rectangles, lines, and pattern filled objects, the best way to store it is to just store the bit-map. The problem with bit-maps is that they take up a lot of memory space. Storing bit-maps, therefore, usually goes hand in hand with some type of compaction scheme. In general, bit-mapped data can benefit greatly by some method of run-length encoding. Run-length encoding treats the bit-map data as one long linear string of bytes. Where several identical data bytes appear in series, the same data can be represented more compactly by a byte which contains a count followed by the data byte to be repeated. Compacting data has been the subject of many a scholarly paper, and there are many approaches which can be taken. GEOS bit-maps support three formats for storing the data. GEOS bit-maps are called Bit-Mapped Objects. The formats can be switched at will within the bit-mapped object. This allows the compaction program to tailor the technique being used to compact the data to the character of the data being compacted. Two of the formats employ methods of compaction, while the third indicates a number of unique data bytes to follow with each one appearing once, a strict bit-map.

The first byte in a bit-mapped object is referred to as the COUNT byte. Encoded into this byte is the format and number of data bytes in that format which follow. Together the COUNT byte and the following bit-mapped data make up a COUNT/Bit-map pair. Several COUNT/Bit-map pairs make up a bit-mapped object.

The Compaction Formats

> The compaction schemes compact and uncompact data bytes in SCANLINES — horizontally accross the screen — not as byte arranged in cards as they appear in c64 memory.

When uncompacted, any arbitrary byte N appears on the same scanline and immediately to the left of byte N+1 (unless N is the last byte on the line and N+1 the first byte on the next line). This, of course, is totally different from the way graphics data is normally stored in the c64 memory. The reason for this reorganization is that bit-mapped data compacts much better horizontally. After the data in each COUNT/Bit-map pair is uncompacted, it is reordered to be placed correctly within screen RAM: the second byte to be uncompacted, for example, is placed eight memory locations after the first so that it appears

on the same scanline. (This makes compaction slower, but since the disk is so much slower than anything else, saving disk space and seek time by decreasing the number of blocks to write turns out to be a worthwhile tradeoff.) The methods of compaction are discussed below.

Each COUNT/Bit-map pair begins with a COUNT byte. The value of this byte determines the format of the following data. If COUNT is within 0 to 127 then the first format is indicated. 128 to 220 indicates the second format, 221 to 255 is the third. The byte following COUNT in the first format is repeated COUNT times. Thus if COUNT were 100 and the following byte were 0, then this would uncompact to 100 consecutive 0's in the bitmap. If COUNT takes a value within 128 to 220, then the following (COUNT-128) bytes are straight bitmap, each used once. Thus if COUNT were equal to 128 + 35 = 163, then this would indicate that COUNT is followed by 35 unique bytes. If COUNT takes on a value within 221 to 255, then the following data is in BIGCOUNT format. BIGCOUNT is an interesting animal and deserves greater explanation.

BIGCOUNT was invented as a way of repeating a pattern which takes up several bytes. That is, suppose you had a 4 byte repeating pattern:

xxxy xxxy xxxy xxxy.

To display this in BIGCOUNT format you must first discribe the 4 byte pattern in one of the first two modes. xxxy can be described in the second format as (132 -128) xxxy, which says use the next 4 bytes once each. You can also describe this pattern as 3x 1y, which is two entries in the first format telling GEOS to use a three times and b once.

Now that the pattern to repeat is described you need to tell GEOS how long the pattern description is and how many times to repeat it. The first byte is the number of bytes in the pattern, and is presented as (COUNT - 220). This is the COUNT byte. The byte immediatly following the COUNT byte is referred to as the BIGCOUNT byte. It contains the number of times to repeat the multibyte pattern. The above pattern can thus be represented as the following string:

224 4 3x 1y

This saves ten bytes out of a 16 byte pattern. Below is a table that summarizes the three formats.

COUNT	FORMAT	DESCRIPTION
00 - 127	**COUNT DATABYTE**	Use next byte COUNT times
128 - 220	**COUNT DATABYTE, ...**	Use next (COUNT-128) bytes once each.
221 - 255	**COUNT BIGCOUNT PATTERN**	COUNT - 220 = number of bytes in pattern to follow. Doesn't count BIGCOUNT. BIGCOUNT = number of times to repeat the PATTERN. PATTERN describes a pattern using the first two formats.

To summarize, the bit-mapped object is a collection of COUNT/Bit-map pairs in different compaction formats. A COUNT/Bit-map pair consists of a format byte followed by a series of data bytes in the indicated compaction format. After being uncompacted, the data is reordered from scanlines to cards.