ASSEMBLING THE CAST

Sprite creation is one of the most exciting features of the Commodore 64's graphics capabilities — both in terms of the enjoyment gained in designing them and because they allow fast-moving games to be written in BASIC. In this part of our project to create a Subhunter game, we take you through the full procedure of sprite design.

A sprite is a large movable graphics shape. It is designed in much the same way as the eight-by-eight user-defined characters discussed earlier in the course (see page 233) but is constructed on a much larger grid. Once a sprite is defined, attributes such as colour and screen position are controlled by a set of special registers in the Commodore 64's video control (or VIC) chip.

A sprite is made up of 21 rows of 24 pixels. Each row consists of three eight-pixel segments and is represented by three bytes of memory, so 63 bytes in all are required to store the data for one sprite. As with user-defined characters, each pixel on the sprite grid that will be illuminated in the final sprite shape is represented by binary one (and the non-illuminated pixels by binary zero). Thus, for each row of the sprite we can calculate the decimal equivalents of each group of eight binary digits. The diagrams we give here show the four sprites that will be used in the Subhunter game. The numbers down the side of each drawing are the decimal equivalents that will form the data statements for each sprite (as given in lines 6000 to 6370 of the program).

Moving Target
A sprite is made up from 21 rows of three bytes; these bytes are actually binary bit patterns, but are stored in the BASIC program data lines as their decimal number equivalents. These values can be seen beside the sprite diagrams and in the program listing. The program POKEs the values into a dedicated area of RAM, where the video controller chip accesses them as sprite data, displaying and moving them on the screen with a minimum of programming effort

LOWERING THE TOP OF MEMORY

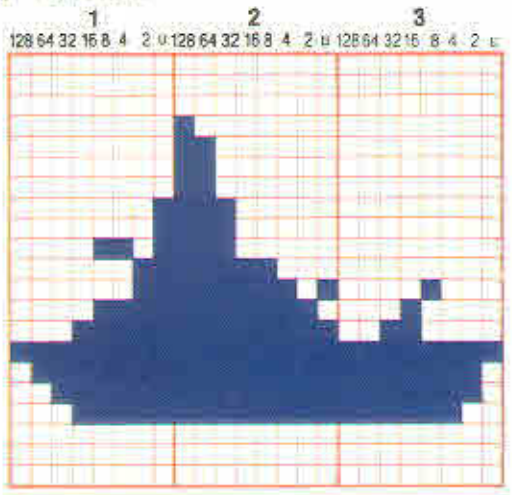
Once a sprite has been defined and converted into a series of DATA statements, the data must be READ and POKEd into memory. Sprite data can be positioned in several memory locations. For example, using the locations starting at 12288 will place it in the BASIC program's area, which runs from 2048 to 40960. As a BASIC program is typed into the Commodore 64, it fills memory space from location 2048 onwards. A program would need to be 10 Kbytes long before it would reach location 12288 and thus overwrite the sprite data. However, when a program is running, any variables used are stored in the area above that used to store the program — string variables, in particular, build down from the top of the BASIC program area. As the Subhunter game uses the timer variable, TI$, regular updating of its value and its subsequent storage will eventually overwrite the area in which we wish to store the sprite data.

A solution to this problem is to lower the top of the BASIC program area to below the area in which the sprite data is kept. The pointer to the address of the top of memory is held at locations 55 (lo-byte) and 56 (hi-byte). Normally these two locations contain the values 0 and 160 respectively, which represent the address 40960. In lo-hi form, the location 12288 is given as 0 and 48. We can lower the top of memory to this location by simply POKEing these values into locations 55 and 56 at the start of the program (see line 90).

SPRITE POINTERS

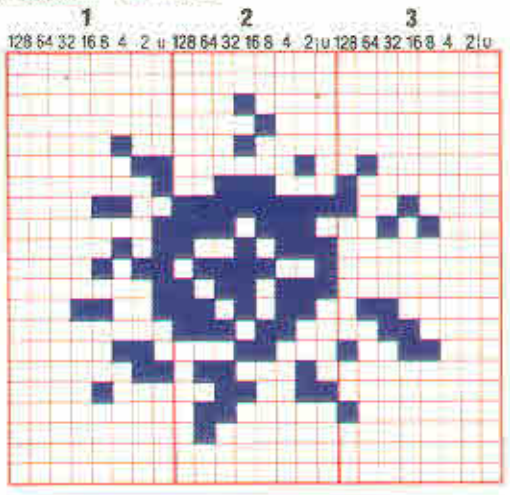
As sprite data can be positioned in various parts of memory, a pointer is needed to indicate where that data begins. There are eight sprite pointers, held in locations 2040 (for sprite 0) to 2047 (for sprite 7). The value held in each sprite pointer is related to the area that holds the sprite data by this formula: start of 63 bytes of data = (sprite pointer)×64. The data for the ship in our program starts at 12288 and the ship is to be designated sprite 0, so the pointer in location 2040 is 192

SHIP – SPRITE 0



1	2	3
0	0	0
0	0	0
0	0	0
0	128	0
0	192	0
0	192	0
0	192	0
1	224	0
1	224	0
13	224	0
3	248	128
3	253	8
15	254	16
31	255	48
255	255	255
127	255	254
63	255	254
31	255	252
0	0	0
0	0	0
0	0	0

EXPLOSION – SPRITE 1



1	2	3
0	0	0
0	0	0
0	16	0
0	8	0
4	16	0
3	2	64
1	56	128
12	255	144
1	238	40
5	151	0
11	121	0
1	183	0
25	214	96
0	236	48
6	24	152
3	98	0
8	51	0
0	96	128
0	64	0
0	0	0
0	0	0