AN ENCODING SCHEME FOR PRESENTATION GRAPHICS WITH ANIMATION

Howard J. Ferch
Dept. of Computer Science, University of Manitoba
Winnipeg, Manitoba, R3T 2N2

ABSTRACT

Bit-mapped raster graphics systems are used in the majority of personal computer systems. As the resolution of these systems increases, and the number of levels of grey-scale, or the number of colours, is increased, encoding of images becomes of greater concern, particularly for interactive systems, or those using animation.

This paper presents a variation of run-length encoding which is faster, but is similar in its degree of space encoding, to run-length encoding. Its application to a menu-driven presentation graphics system is discussed.

One particular advantage of this scheme is its adaptability to the types of animation possible on personal computer raster systems.

KEYWORDS: raster graphics, encoding, animation

INTRODUCTION

Bit-mapped raster graphics systems are very commonly used, particularly on personal computers. These systems typically have a dual-ported frame buffer, which is accessible both to the display hardware, and to the central processor (by mapping the frame buffer into the processor's address space) [FOLEY82]. As memory capacity and speeds go up, and costs go down, it is possible to have higher resolution screen displays, and to display more grey-levels, or colours.

The increased size of the frame buffers means that images are larger, and hence occupy more disk space, take longer to read, and take longer to copy into the frame buffer. This has particular implications to applications in which speed is of major importance, such as those which operate interactively, or which contain animation.

One system making use of such technology is a menu-driven graphics presentation system being developed for Parks Canada. In this system, an IBM PC-XT equivalent is being used to provide interpretive information to park visitors. Each menu page is displayed in a resolution of 640 by 400 pixels, with 16 colours possible for each pixel. Simple animation sequences are used to present the information. For example, to demonstrate the creation of sinkholes, an animation sequence shows rain falling, the water soaking into the ground and dissolving the gypsum, the ground being undermined, and then the ground collapsing. Since each image takes 128,000 bytes of storage, an encoding scheme is required to reduce the disk space, the main memory space, the disk transfer time, and the display time.

EXISTING ENCODING SCHEMES

A large amount of attention has been paid to the problem of encoding graphical images. (See [HALL79] for an overview of such techniques). However, the majority of such schemes are not adequate for this particular application. Techniques such as Huffman encoding, which use variable length bit strings, are simply too slow for the computing power of a personal computer. Most image creation packages for personal computers (such as Apple's MacDraw [APPLE85]) encode a picture as the objects which were used to create it (e.g. as a sequence of line segments, polygons, etc.). This is the same approach that is used in presentation systems such as Telidon [CSA83]. Again, for real-time animation functions, this approach is too slow. In fact, one of the early objectives of the Parks Canada system was that it was to be much faster than existing museum systems based on Telidon.

A lesser degree of space encoding is provided by run-length encoding, in which an image is stored as a set of tuples,

each containing a colour (or grey-scale) sequence, and an associated repetition factor. Such a technique combines a reasonable level of space compression, with a simple, and hence quick, decoding algorithm. This paper presents a variation of run-length encoding, which is faster, but is similar in the degree of space encoding, and which is particularly adaptable for the type of animation described above.

THE ENCODING SCHEME

Instead of storing a bit pattern (usually a byte or a word), and a repetition factor, one can make use of particular characteristics of the central processor found in the IBM PC. In this processor, (the INTEL 8088)[INTEL81], processor instructions exist to replicate a byte or word through memory, or to copy a string from one location to another. A byte or word pattern to be replicated into the frame buffer may thus be encoded as the machine language instruction sequence required to place that pattern directly into the frame buffer. For example, the following instruction sequence puts the byte containing the hexadecimal value 56 into 250 successive locations of memory (assuming that the appropriate segment registers have been loaded).

```
    MOV   AL,56H
    MOV   CX,250
REP STOSB
```

This instruction sequence occupies 7 bytes of memory, and thus we can use 7 bytes to encode 250 bytes of the image. Since this instruction sequence automatically increments an address register to point to the next location in memory (in the frame buffer in this case), we may follow it immediately with another sequence which inserts the next pattern of the image into place, and so on. Thus, the entire image may be encoded as a machine language program, which, when executed, will generate the original image directly into the frame buffer.

For portions of the image which have a large number of very small runs, we can use another sequence in the generated program. Having generated a portion of the image in the frame buffer, it is possible to copy any section of this into a later portion of the image, where the same sequence appears again. On the 8088, an instruction sequence to do this consists of:

```
    MOV   SI,source offset
    MOV   CX,length
REP MOVSW
```

This sequence occupies 8 bytes of memory, and also updates the appropriate address registers. One other instruction sequence is used. This is a variation of the above, which inserts a new string into the frame buffer, by copying it from a copy which is placed in-line in the program, as in:

```
    MOV   SI,OFFSET LABELx
    MOV   CX,length
REP MOVS  WORD PTR[DI],WORD PTR CS:[SI]
    JMP   LABELy
LABELx: DW  the string
LABELy:
```

This sequence occupies 11 bytes plus the string length.

In order to apply these sequences, the following algorithm can be used. The original image is stored in main memory, as it will appear in the frame buffer. Starting with the first word of the image, successive words of the image are scanned to see if the image starts with a repeated word. If so, the appropriate machine language code is generated, and the next word of the image is scanned in the same fashion. If the word being examined is not repeated, then the section of the image that has been generated to date is examined to find the largest possible string matching that at the current location. Only if no reasonable length such matching string exists is the third alternative used, and then only long enough to bridge the gap until a repeated word, or a previously encountered string is again encountered.

THE RESULTS

Table 1 summarizes the results, as applied to three sample images from the presentation sequence. The first image is a landscape scene, the second contains a large amount of detail and a large amount of fairly small text, and the third is a map, with a fairly small amount of detail. This table gives the number of occurrences of each of the three instruction sequences, the maximum length of string generated by each, the total number of words generated by each type, the total resulting size of the machine language program, the space reduction factor, (as a percentage of the original size of 128,000 bytes), and the size of the image when encoded using both an object representation, and using run-length encoding. As can be seen from the table, this encoding scheme does use more space than run-length encoding. However, the generation speed is approximately two times faster than using the run-length encoding, due to the lack of overhead spent decoding the tuples of the encoded image. For the three images used, the picture generation averaged 0.3 seconds, while for run-length encoding, the average time was 0.6 seconds.

	IMAGE 1	IMAGE 2	IMAGE 3
Number of repetition sequences	712	477	648
Max repetition size (words)	1367	530	902
Total number of words created	35008	13396	17824
Number of copied strings	2996	6521	3562
Maximum string size (words)	161	848	159
Total number of words created	27333	49432	44606
Number of new strings	970	868	994
Maximum string size (words)	14	11	11
Total number of words created	1659	1172	1570
Resulting encoded size in bytes	37343	61000	41346
Percentage space occupied	29%	48%	32%
Number of bits per pixel	1.17	1.91	1.29
Size of the run-length encoding	22712	56168	26256
Size of the object encoding	17470	N/A	7870

TABLE 1 - ENCODING RESULTS

REFINEMENTS

One problem exists with the given algorithm. On many of the display adapters using a frame buffer which is dual-ported between the display and the main memory, it is not possible to read from the frame buffer at any desired time. Instead, due to the interactions in the hardware, it is necessary to read the frame buffer only during the retrace intervals, to be sure that the data is correct. Thus the second sequence given above must be modified somewhat. An additional 2 bytes was added to call a subroutine which waits for the retrace to begin. In addition, the maximum length of the move must be limited. Rather than doing this, a refinement to the algorithm was introduced.

The simplest approach was to remove the use of the second sequence (copying already existing sequences). It was felt that this would probably lengthen the encoding somewhat, but the display time would be reduced. In fact, this did not happen. For images which did not contain a large amount of text, the encoding was actually smaller, provided that the encoding was done on a byte, rather than a word level. With this result, the third sequence (generating a new string) was also removed, with the same result. Thus the resulting encoding which is actually used is very simple. Runs of repeating bytes were encoded using only the first sequence, with an additional optimization using a special sequence for runs of length 1 byte, or 2 bytes. For the three images given above, the resulting encodings had sizes of 37878, 62717, and 38574 respectively. The overall total size across all images was slightly reduced, with no loss of speed.

One other aspect of interest for this encoding approach is its adaptability to other processors. The two major competitors to the 8088 processor are the Motorola 68000 [MOT80], and the National Semiconductor 16000 [NAT83]. While the algorithm can be adapted to both of these processors, it cannot be done in as simple a fashion as it is for the 8088. Neither of these processors provides an instruction which can replicate a value through a range of memory locations. Thus a loop is required. In order to achieve similar compaction levels, this requires the use of an out of line subroutine, with a subsequent loss of speed.

ANIMATION

This encoding scheme is easily adapted to support the type of animation described earlier. To provide animation, a sequence of related images is generated. Then, the above encoding scheme is used to insert the changes from each image to the next in turn into one machine language routine. In many cases, a slower transition from one image to another is desired, such as when scrolling text onto an image, or in having a river change its colour in a given direction, to portray flooding. In these cases, delays are added to the machine language code generated, to achieve the desired rate of speed.

Since dissolves from one scene to the next occur very often, a suite of routines to provide different orderings and timings of dissolves has been created. In the most general case, the user may add a line of any shape and size to an image and request an ordered dissolve from one image to the next moving outwards from the given line, or moving inwards, and he may at the same time specify the speed of movement. Standard top to bottom, left to right, etc orderings are also provided.

Another adaptation allows for repetitive events, such as the blinking of an arrow. In this case, code to insert the arrow, and then to remove it, is generated and then the machine language routine is simply repeatedly executed. Delay sequences are added to achieve the desired rate of blink.

CONCLUSIONS

An encoding scheme for bit-mapped frame buffers which are memory mapped into the central processor's address space has been described. This scheme provides for very fast decoding, while at the same time providing a reasonable level of space encoding.

A complete menu-driven interactive display system for Parks Canada has been constructed using this encoding scheme with the following results. A total of 121 images have been encoded. Of these, 65 primarily contain text (although text and graphics can be freely intermixed), and the remaining 56 primarily display landscape, or map information. In most cases, an image is generated from the previous one using a dissolve sequence. The 65 text images are encoded in 1,166,141 bytes, for an average of 17,940 bytes per image. The 56 graphics displays occupy 1,865,833 bytes, for an average of 33,318 bytes per image. The resulting encodings, including the animation timing delays, and all overhead occupy 19.6% of the original space of the images. The display speed is just slightly reduced from the maximum achievable display speed.

REFERENCES

APPLE85 Apple Computers, "Inside Macintosh", 1985

CSA83 Canadian Standards Association, "Videotex/Teletext Presentation Level Protocol Syntax (North American PLPS), Canadian Standards Association, December 1983.

FOLEY82 Foley,J.D. and Van Dam,A. "Fundamentals of Interactive Computer Graphics" Addison-Wesley, 1982

HALL79 Hall,E.L. "Computer Image Processing and Recognition", Academic Press, 1979

INTEL81 Intel Corporation, "iAPX 86,88 User's Manual", 1981

MOT80 Motorola, Inc. "MC68000 16-bit Microprocessor User's Manual", 1980

NAT83 National Semiconductor Corp. "NS16000 Instruction Set Reference Manual", 1983.