An approach to real-time scan conversion*

by FRANKLIN C. CROW

University of Texas
Austin, Texas

INTRODUCTION

Scan conversion—that is, the transformation of line segment endpoint coordinates into a collection of scanline segments suitable for raster display—is important because raster displays have many advantages over random-scan, or calligraphic, displays. The calligraphic displays require extensive special-purpose hardware to generate line segments, or "vectors," and to drive the beam deflection circuits of the CRT. Furthermore, by its very nature, the calligraphic display is subject to damage caused by software defects; a program which directs the beam to the same portion of the CRT face for too long can damage the phosphors, creating a permanent dark spot.

On the other hand, the raster display can be driven by simple digital signals and is immune to software-induced damage. The raster display uses a technology shared by millions of television receivers around the world. This means lower costs through mass production and more flexibility through associated devices designed to store, transmit, project and make hard copies from video signals. For these same reasons, research into new displays is almost entirely concentrated on TV-compatible proposals. Thus, inexpensive displays for computer graphics are most likely to use raster displays in the future.

There are a number of current products offering raster-graphic displays using digital image memories with a bit for every picture element of the display. Such memories provide a very straightforward way to perform scan conversion and are quite appropriate for primarily static images. However, dynamic images pose difficult problems since moving portions of the image must be cleared from the memory and then re-drawn for each successive frame. Furthermore, any static portions of the image which coincide with cleared portions of the display will themselves be partially cleared, leaving unsightly gaps (Figure 1). It would be preferable to generate all dynamic lines together, in scan order, 30 to 60 times a second.

There are a number of current products offering raster-graphic displays using digital image memories with a bit for every picture element of the display. Such memories provide a very straightforward way to perform scan conversion and are quite appropriate for primarily static images. However, dynamic images pose difficult problems since moving portions of the image must be cleared from the memory and then re-drawn for each successive frame. Furthermore, any static portions of the image which coincide with cleared portions of the display will themselves be partially cleared, leaving unsightly gaps (Figure 1). It would be preferable to generate all dynamic lines together, in scan order, 30 to 60 times a second.

Ideally, static portions of an image should be stored in an image memory while the moving portions are dynamically scan-converted. However, there are arguments for dynamically scan-convertimg the entire image, assuming that scan-conversion can be made to run fast enough. If vectors are to be colored or grayscale tricks used to smooth the lines (as discussed later in this paper) then several bits must be used to define the characteristics of each picture element. This requires a rather large amount of memory for an entire image of any worthwhile resolution (307,200 times N bits for a 640 by 480 element image).

There is a reasonably clear trade of memory size against processor power in the decision between an image memory and real-time scan conversion. The image memory needs a processor for generating vectors, etc. But, it doesn’t need the power necessary for the techniques discussed here. Buying considerably more processor power could eliminate a few megabits of memory. It is not clear, given the rapid pace of development in both processor and memory systems, which alternative will be more economical ten years from now.

There are, of course, compromises. The image memory can be divided into character-sized cells and memory allocated only to those cells through which a line passes. This would allow important savings when using color or grayscale. It is also possible to use separate image memories for static and dynamic portions of the image, allowing the dynamic portion to be cleared after every frame display. This sort of functionally-divided form of display has been used successfully with direct-view storage tubes.

There have been at least two previous efforts to develop systems using real-time scan conversion. Check reported a system in which vectors were chopped into short lengths and then grouped into horizontal strips of the display. Lindner and Tozzi have worked on a system which takes an approach similar to that of this paper but appears much more complicated. The approach taken here is heavily influenced by experience with scan-ordered hidden-surface algorithms; similarity between adjacent scanlines is depended upon to minimize computations. The basic scan-conversion algorithm, described next, was initially used, by the author, in a software implementation at the University of Utah in 1973.

THE BASIC SCAN CONVERSION ALGORITHM

A line segment has a great deal of "coherence." That is, given one part of a line segment, the rest is easily extrapo-
Figure 1—the effect of selective erasure using a digital image memory.

lated. Thus very simple changes suffice to update the scan segment description for a vector from one scanline to the next. Digital vector generators use this property to reduce vector drawing to a series of incremental operations. The algorithms developed here differ from previously published methods in that all vectors are generated in an interleaved order dictated by the raster scan pattern. Earlier methods generate vectors individually, using the most convenient order for the algorithm involved.

The scan conversion algorithm is composed of three reasonably distinct tasks. First, lines in the display list must be sorted by the order in which they first appear in the scan. Second, for each scanline the position and length of the scan segment representing each vector crossing that scanline must be computed. Finally, the scan segments for each scanline in turn must be sent in proper order to the display.

The conversion process is spread over a pipeline consisting of a general purpose image update processor, a Y-sorted buffer, a microprogrammed scanline processor, an X-sorted scanline buffer and, finally, a hardwired picture element processor (Figure 2). The two buffers are implicitly sorted by writing into predetermined slots. one for each scanline in the Y-sorted buffer and one for each picture element in the X-sorted buffer.

The design process was heavily influenced by the desire to use general purpose processors wherever possible in order to be in the best position to take advantage of future advances in microprocessor components. Thus, while the scanline processor could no doubt be more effectively implemented in random logic, a bit-sliced microprocessor was chosen to maximize flexibility.

Very modest design goals have been set for the first implementation, a machine capable of maintaining a few hundred vectors with no more than about 50 intersecting any one scanline. For this effort a display resolution of 320 by 240 picture elements at 60 fields per second will be used, roughly the resolution available from an inexpensive home television set. The eventual design goal is at least 1000 by 750 picture elements, or about an order of magnitude improvement. For the moment, the more modest goal allows concentrating on the algorithms and minimizes the sort of difficulties which arise from pushing digital circuitry to state-of-the-art limits.

THE IMAGE UPDATE PROCESSOR

At the head of the scan conversion pipeline, the image update processor is dedicated to keeping track of the vectors
to be displayed (the "display list") and loading the Y-sorted buffer. The display list may be formatted to suit the application at hand, the architecture of the host computer, the architecture of the software system or whatever other constraints exist. In short, the display list organization is of no concern here. Suggestions for structuring display lists can be found in Reference 16.

The important task for scan conversion is loading the Y-sorted buffer. For the convenience of later, more tightly time-bound elements of the pipeline, the Y-sorted buffer stores vector descriptions in a different form. Only the X-coordinate of the higher end of the vector is stored, the Y-coordinate being implied by the position of the entry. An increment for the X-coordinate serves to define the direction of the vector since the Y.increment is assumed to be one. All that is left to completely define the line is a length measure. This is supplied as the number of scanlines spanned by the vector.

The calculations involved in computing the buffer entries from vector endpoint coordinates are dominated (in small processors at least) by one division step. This division is necessary to computing the increment. The upper endpoint X-coordinate is available directly, after a compare of the Y-coordinates. Nearly as simply, the number of scanlines spanned is given by the difference of the Y-coordinates plus one. However, the increment is the difference in X-coordinates divided by the number of scanlines spanned.

Updating an image consisting of 200 lines at a rate of 30 times a second allows 166 microseconds per vector. Current 16-bit microprocessors with built-in multiply and divide can execute the necessary instructions in about that same time. Since the image update rate can vary from 60 times a second down to around 20 times a second without destroying the smoothness of the motion, there is some leeway available. Use of a minicomputer or one of the more powerful 16-bit microprocessors now appearing should supply adequate power for the image update function.

THE Y-SORTED BUFFER

The vector entries, as produced by the image update processor and stored in the Y-sorted buffer, are organized to make it easy for the scanline processor to access the information it needs. Specifically, the scanline processor must be able to readily retrieve all the vectors whose upper endpoints lie on a given scanline. Therefore, the Y-sorted buffer is organized as a fixed length array of list heads, each of which is either null or points into a memory containing linked lists of vector entries (Figure 3).

All unused vector entries in the buffer are similarly linked in a separate list to make allocation and deallocation of the fixed-sized vector entries a simple operation. Algorithms for this sort of memory management can be found in Knuth. 12

The Y-sorted buffer can be used in one of two modes. If the drawing displayed is very dynamic, it is simplest to recreate the entire set of vector entries for each image update. However, for partially-static drawings and those with only translational motion, processor cycles may be saved by modifying the existing structure. Using a doubly-linked list for each scanline, a vector entry can be removed from one scanline list and appended to another. If only the position and not the direction of the vector has been changed, it suffices to change the upper end X-coordinate in the vector entry. All other numbers remain the same.

The latter mode, of course, involves contention for access to the buffer. Both the image update processor and the scanline processor must access the buffer. Since the scanline processor is under greater time constraints, it is given priority.

THE SCANLINE UPDATE PROCESSOR

The contents of the Y-sorted buffer are used to generate a set of scan segments for each scanline. The scan segments
consist of a start position and a run length (the number of picture elements to intensify) for each vector which intersects a given scanline. The scanline processor generates the scan segments for each scanline in turn, moving from the top of the drawing to the bottom.

A scan segment is easily produced from the current X-coordinate of a vector and the increment giving the position of the vector at the next scanline. The start position is just the vector position; the run length is just the integer part of the sum of the increment and the fractional part of the vector position.

As the scanline processor works its way down the picture, a "scanline array" containing those vectors which intersect the current scanline must be maintained. As each new scanline is processed, three operations are necessary to maintain the scanline array. First, old vectors in the list which lie entirely above the current scanline must be discarded. Then, vectors which do intersect the current scanline must be updated to find the current point of intersection. Finally, new vectors whose topmost end coincides with the current scanline must be added to the array.

Recall, the Y-sorted buffer contains three data on each vector: (1) The horizontal position of the topmost end, (2) the increment which will give the position at the next scanline and (3) the number of scanlines spanned by the vector. Three similar quantities must be maintained by the scanline processor for all vectors intersecting the current scanline (Figure 4).

At each scanline, the scanline array is processed. For each array position with a valid entry, the increment is added to the fractional part of the vector position. The integer portion of the result is then stored in the X-sorted scan buffer as the run length. The increment is then added to the vector position and the result stored for use at the next scanline. The number of scanlines spanned is then decremented. If the result is zero, the entry is tagged invalid.

New vector descriptions are inserted where invalid entries are found or created while generating a scan segment. At each such occurrence, the Y-sorted buffer is checked for a new vector which, if found, is then loaded into the available array position and processed to generate a scan segment.

Given the standard scan rate of 15.750 lines per second, the scanline processor has 63.5 microseconds to produce a scanline or 1270 nanoseconds per vector, assuming the initial design goal of 50 vectors per scanline. Pipelining the microinstruction fetch, a vector can be processed in six microcycles: (1) Fetch the entry, (2) sum for the run length, (3) store the run length, (4) sum for the position on the next scanline, (5) decrement the scanlines spanned and (6) store the updated entry. Adding a new entry requires two or three more cycles to transfer the entry and update a list pointer. Thus a somewhat relaxed 200-nanosecond cycle time can be used without overly specializing the processor. This is well within the capabilities of current four-bit processor slices.

### THE X-SORTED SCAN BUFFER

The scan buffer is actually two buffers. One receives scan segments from the scanline processor while the other is read by the picture element processor. After each scanline is processed the two buffers are functionally switched (Figure 2).

The scanline processor sends a run length to be stored at an address given by the accompanying vector position (Figure 5). A read-modify-write cycle on the given address is used to fetch the previously-stored run length, compare it with the incoming run length and store the larger of the two. This process automatically resolves the problem of overlapping scan segments starting at the same picture element.

At the end of each scanline, one buffer should contain all the scan segments for the next scanline stored in X-sorted order while the other should be zeroed, ready to accept another set of scan segments. This implies that the picture element processor must clear the memory as it reads it. Given 320 picture elements per scanline, the buffer must cycle at around 150 nanoseconds. If the picture element processor is to clear the memory in a read-modify-write cycle, then a 70-nanosecond memory is needed. Alterna-

---

**Figure 4**—Information stored in the scanline processor's data memory.

**Figure 5**—Structure of the X-sorted buffer.
tively, the memory can be interleaved on the least significant bit or a bulk clear executed during the beam flyback time (about ten microseconds).

THE PICTURE ELEMENT PROCESSOR

The final element in the picture production pipeline produces a sequence of pulses which, when mixed with synchronization signals for a video monitor, result in intensified scan segments properly placed on the display. This involves reading the $X$-sorted scan buffer, setting a flip-flop at the beginning of a scan segment and resetting that flip-flop at the end of a scan segment.

A counter is used to divide the visible portion of a scanline into 320 parts and to address the $X$-sorted scan buffer. A down counter running at the same rate is used to define scan segment lengths. When the down counter is loaded, the output flip-flop is set. When the down counter reaches zero, the flip-flop is reset.

The down counter is loaded directly from the $X$-sorted scan buffer. Before loading, however, the downcounter contents are compared with the run length from the scan buffer. Only if the magnitude of the incoming run exceeds the content of the down counter is the down counter reloaded. This ensures that overlapping scan segments are properly handled. A series of overlapping segments will produce a single long intensified strip on the display.

The 150 nanoseconds allowed by the 320-element scanline is ample time to perform the compare-and-load operation. Accesses to the scanline buffer are overlapped with the compare operation using an intermediate register.

SPEEDING UP THE IMPLEMENTATION—MORE LINES AND HIGHER RESOLUTION

It should be clear that the picture element processor is designed to handle the case where there is a new scan segment at every picture element. Therefore arbitrarily complicated drawings can be handled at the tail end of the pipeline. However, higher resolution may require producing a picture element as often as every 15 nanoseconds, requiring high-power circuitry and greater concurrency for the scanline buffer and comparator.

The word-processing industry is currently moving to high-resolution monitors in an effort to make the display look as much as possible like a standard 8½-by-11 typewritten page. Because of this, high-resolution raster monitors are now available at costs as low as a few hundred dollars. The semiconductor industry can be expected to eventually produce high-resolution versions of the display controller chips now being produced for standard-resolution monitors greatly simplifying the problems in driving the faster displays.

Current restrictions on the number of vectors which can be displayed lie in the scanline update processor. Bit-sliced microprocessors are not currently fast enough to allow more than 100 vectors or so to intersect a given scanline. The algorithm executed by the scanline processor is simple enough to be readily translated to random logic. Estimates indicate a potential for increasing the processing rate by as much as an order of magnitude by such means. This would allow up to 1000 vectors on a scanline on a 240-line display or roughly 300 on a 750-line display. Wild guesses suggest that the chip count and cost of the scanline processor would increase by a factor of three to five. However, this violates the philosophy of minimizing special-purpose circuitry.

The other approach to speeding up the scanline processor involves running several microprocessors concurrently. Unfortunately, adequately speedy processors are not yet cheap enough that more than one or two of them can be considered economical in a supposedly low-cost terminal. If cost considerations are ignored under the supposition that the semiconductor industry will solve that sort of problem in due course, then a collection of processors could be arranged to deliver updated vector entries at a rate of one per microcycle. The processor cost and complexity could be expected to increase by a factor of five to ten.

An order of magnitude increase in performance allows a 1000 element by 750 line display with up to 300 vectors crossing any one scanline. This would allow display of roughly 60 lines of 100 or so legible characters, or (equivalently) 18,000 short vectors, or 3600 vector-inches on a 16" by 12" screen (19" diagonal). These figures are for 60 frames per second. Stroke-writing displays with equivalent or better specifications currently start at around $20,000. The sort of display system discussed here should cost considerably less than one-half that amount. Whether it could be marketed at such a low price, however, is open to question.

HIGHER QUALITY LINES

One good look at Figure 1 will reveal the major aesthetic problem with scan-converted vectors. They have ugly kinks which are all too evident at any but impractically high resolutions (compare with Figure 6 made on a calligraphic display). Getting rid of these kinks is a difficult, but not impossible proposition. The observer of the display can be tricked into perceiving smooth lines on the display by the judicious use of grayscale techniques.7,8

When using these techniques, the scan segments become gray-level functions instead of just run lengths. This necessitates a more complicated picture element processor. However, the scanline processor and $Y$-sorted buffer need not be changed.

Gray levels for producing any scan segment may be stored in a single table which stores the universal intensity profile for all scan segments. Any given scan segment may be produced by supplying an index into the table for the first picture element, the number of elements and the distance between table entries to be retrieved.

Therefore it becomes the job of the picture element processor to generate these numbers from the scanline position and position increment maintained by the scanline processor. The index of the first table entry is computed from the fractional part of the vector scanline position. The number of elements and the distance between table entries, in turn,
must be computed from the vector position increment. The microcode for the scanline processor can easily be modified to deliver these two numbers instead of the truncated scanline position and run length.

The distance between table entries is obtained from the reciprocal of the vector position increment using a table of scaled reciprocals. The index for the first entry is given by the product of the fractional part of the vector position and the distance between table entries. The number of entries to be used is just twice the run length used previously.

Note that each scan segment must now be treated individually. The problem of overlapping scan segments becomes much more acute. Furthermore, each scan segment must now be twice as long as before. The likelihood that a number of nearly horizontal vectors will cause sufficient overlap to swamp the processor is quite high.

The gray levels of overlapping vectors must be arithmetically combined to determine the gray level for affected picture elements. To be absolutely correct about combining the intensities of overlapping vectors, some measure of the area each vector occupies in a picture element and the area of overlap between such vectors would be necessary. Although such calculations have been used for shaded raster images, the additional quality obtained isn’t worth the expense in this application. Experiments indicate that a simple sum, truncated to the maximum allowable intensity where necessary, gives acceptable results.

It appears unlikely that all this arithmetic can be performed on the fly as the line is scanned out. Therefore, a buffer is needed in which the grey levels to be displayed are stored. Two such buffers may be used. While one is providing grey levels to the display, the other may be used for building the next scanline (Figures 7, 8).

The scan segment information provided by the scanline processor is acted upon by a picture element processor which loads its output into the scanline buffer via a read-sum-write cycle, accumulating intensity at a pixel until saturation. This arrangement eliminates the strict timing constraints involved in scanning directly from an X-sorted buffer of run lengths.

For simple images, the picture element processor should be less than three times as complex as in the initial design.

From the collection of the Computer History Museum (www.computerhistory.org)
A practical approach to real-time scan conversion has been described which can be implemented straightforwardly using currently widely available parts. Projected trends in LSI development indicate that high-resolution implementations competitive with low-end calligraphic displays could be produced at quite reasonable cost within a few years.

CONCLUSIONS

The algorithms for the approach to scan conversion presented here have been demonstrated in software. The translation to a hardware implementation for a limited number of lines and modest resolution should pose no problems. The expansion of the concept to higher resolutions and smooth vectors is expected to provide some challenge, but no insurmountable problems.

The failure modes exhibited when the scan converter is overloaded are totally different from the flicker seen on an overloaded calligraphic display system. There are two choices for a failure mode: either repeat the last scanline, or leave out some scan segments. The former method will cause noticeable stripes on the screen, the latter will cause some vectors to disappear on certain scanlines and perhaps leave other vectors out altogether.

A safer failure mode could be engineered by going to lower resolution whenever overload is detected. At a 60-hz refresh rate one slightly bad frame produced while discovering overflow would probably be acceptable. The degraded mode of operation for low-resolution (320 by 240) would provide only 120 lines vertically. Surely, most users would find this intolerable. On the other hand, a high resolution implementation might run in the degraded mode quite successfully.