

# Data Structures for Raster Graphics

Proceedings of a Workshop  
held at Steensel, The Netherlands, June 24-28, 1985

Edited by  
L.R.A. Kessener F.J. Peters M.L.P. van Lierop

## Problems with raster graphics algorithms \*

*Wm. Randolph Franklin*

Electrical, Computer, and Systems Engineering Dept.  
Rensselaer Polytechnic Institute  
Troy, NY, 12180, USA

### ABSTRACT

Assorted unusual problems with raster graphics digitization or scan conversion algorithms, where the implementation is much harder than the concept, are considered. They include digitizing a line that is on a plane so that the line is always visible, and digitizing a subset of a line correctly. Raster graphics is harder than vector graphics, in a deep sense.

#### 1. Floating point problems

Before we consider raster algorithms in particular, a review of general arithmetic problems on computers is in order, since many graphics difficulties have an underlying algebraic aspect.

Many raster graphics problems arise from computer floating point numbers. In fact, it is to avoid these complexities that integers are sometimes used. Floating point numbers are a model of the real number field which is defined by certain axioms, (Spiegel, 1963). In fact, for almost every real number axiom, there is a computer implementation for which it is violated:

*Distributivity*  $A \times (B + C) = A \times B + A \times C$

For a violation of this rule, consider a decimal floating number system which stores one decimal digit for each number. Calculations are performed exactly and then rounded to one digit. Thus 9 and 10 are exactly representable in this system, but 11 is not. Because of rounding,  $2 \times 4 = 10$  but  $2 \times 8 = 20$ . Let  $A = 2$ ,  $B = 6$ , and  $C = 2$ . In this number system, distributivity is not satisfied, since  $A \times (B + C) = 20$ , but  $A \times B + A \times C = 10$ .

*Associativity*  $(A + B) + C = A + (B + C)$

\* This research is based upon work supported by the National Science Foundation under Grant No. ECS-8351942, and by the Schlumberger-Doll Research Lab. Ridgefield Conn.

This is violated on any machine with fewer than 30 significant digits if  $A = 1.E30$ ,  $B = -1.E30$ ,  $C = 1$ .

*Commutativity*  $A + B = B + A$

This can be violated on systems which can hold temporary numbers in registers at a higher precision than they can be stored in memory, and further allow a register - memory operation if the register is the first operand. When the second operand is stored in memory, it will lose precision. Four different machines with overlength floating point registers that violate commutativity are given in (Malcolm, 1972) and (Gentleman and Marovich, 1974).

*Reciprocal*  $\underline{E}(Y : X \times Y = 1)$ .

For example, in the APL system at RPI, there is no multiplicative inverse for 3 since  $1 - 3 \times (1/3) = 1.388E-17$ .

These problems can be reduced by using a properly designed floating point standard, but cannot be entirely removed. The IEEE standard, designed by numerical analysts, is the best. In contrast, those systems dating from the 1960s can be quite poor.

## 2. Raster coordinate standards

Two different coordinate systems are possible:

- having the coordinate system pass between the pixels, which are considered to be little squares and
- having it pass through the centers of the pixels, which are considered to be points.

These two standards are of equal power, but are incompatible in various *off by one* ways:

- Between  $X = 1$  and  $X = 10$ , there are 9 pixels by the first standard, but 10 pixels by according to the second.
- If we define a circle such as  $x^2 + y^2 = 4$ , then the number of pixels inside it depends on the standard.
- These differences are also reminiscent of the different methods of antialiasing. For example, if a pixel is a point then any object smaller than the pixel spacing will be invisible. Thus if pixels are points, we must filter and bandwidth-limit the objects. On the other hand, if pixels are squares (or more complicated convolution functions) then the objects need not be pre-filtered.

We will switch between the two standards as appropriate.



### 3. Why raster is harder than vector

Many people assume that raster graphics is easier than vector graphics since we are working with simple integers, perhaps from 0 to 1023, rather than with an infinite number of real numbers. This first impression is false, as consideration of the following problems may show:

- drawing a slanted line, and
- filling a polygon

Drawing a vector line is trivial, but a raster line requires a scan conversion algorithm. Filling a vector polygon requires only intersecting the crosshatch lines with the edges. However a raster polygon raises questions such as 4-connectedness versus 8-connectedness, and how to find a seed in each component of the polygon.

People feel that raster graphics should be simple because integers are simpler than real numbers. This is also false. Consider the problem intuitively. One can describe integer problems, that were posed several centuries ago, and are to date unsolved, to an intelligent primary school student. Take Goldbach's conjecture, or Fermat's Last Theorem, for example. Goldbach's conjecture is that every even number is the sum of two primes. There are no comparable problems in the real numbers where there is such a large gap between a problem's statement and its solution.

The comparative difficulty of the integers has a deep foundation in first order logic. Consider the set of all formulae with variables in some domain, and using the universal and existential quantifiers ( $\underline{A}$  and  $\underline{E}$ ). We are also allowed certain primitive functions, such as addition. For example, the following formula says that every number has a half:

$$\underline{A} ( x : \underline{E} ( y : y + y = x ) ) \quad (1)$$

Now if our domain is the integers with addition and the successor function, we have Presburger arithmetic. There is a decision procedure for automatically proving the truth or falsity of any such formula, (Yasuhara, 1971), in double exponential time. However, if we also allow multiplication, then there is no such decision procedure, and some formulae are undecidable. Since Goldbach's conjecture could be expressed with addition and multiplication, if a decision procedure existed, then the theorem would be trivial.

However, if the variables are real numbers, then all formulae with addition and multiplication are solvable using Tarski's decision procedure, in double exponential time. Further, formulae with just addition can be proved much more quickly over the reals, in single exponential time, than over the integers. Although every integer solution is also a real solution, it is not contradictory for the integer formulae to be slower to solve. The truth of a formula depends on the domain of the variables. The above formula (1) is false over the integers but true over the reals. The really difficult cases, such as Fermat's Last Theorem, have a continuum of real solutions but only a few integer solutions, if any. Further, in this logic, there is no way to write a formula which selects the integers only. To do so would require a primitive floor function or remainder function which is not provided. Thus, in a deep sense, bumpy integers are harder than smooth reals.



#### 4. Simple Z-buffer drawing

The goal of any algorithm should be to produce the intuitively correct result for a user who is unaware of the algorithm's internals. For example, if object A is in front of object B in real life, then A should be in front of B in the Z-buffer. Many of these problems arise in attempting to form a three dimensional generalization of the two dimensional Bresenham line drawing algorithm.

##### 4.1. Digitizing a line into a Z-buffer

In two dimensions, the obvious way to scan convert or digitize a line is to mark all pixels within distance 0.5 of the line. However, in three dimensions, consider the line

$$L : x = 3y, z = 9y$$

It is easy to give pixel (0,0) the value  $z = 0$ . However, what about pixel (1,0)? Since this point is not exactly on the line, we cannot substitute into the equation to get  $z$ . Various solutions are possible:

- Drop a perpendicular from (1,0) to the closest point on the line, (0.9, 0.3), and use that  $z$ -value,  $z = 2.7$ .
- Since the line's slope is less than one, we can run a vertical line straight up to L, at (1, 1/3) and use the corresponding  $z = 3$ .
- If we know that L is on some relevant plane, such as  $2x + 3y - z = 0$ , then we can substitute (1,0) into this to get  $z = 2$ . The problem with this is that L may be at the common intersection of several planes.
- If instead we consider the pixel to be a square, around the point, then we can put bounds on  $z$  as L passes through the square. Here the square's lower left and upper right corners are at (0.5, -0.5) and (1.5, 0.5) respectively. Line L enters the square on the left with  $z = 1.5$  and leaves on the right with  $z = 4.5$ . This is some help if we are working with interval arithmetic, but doesn't help us get one particular number for the Z-buffer. Sometimes there are particular reasons for using the minimum or maximum  $z$ , as we shall see.

Thus in three dimensions, unlike 2-D, there is no single obvious way to digitize lines.

##### 4.2. Digitizing a plane into a Z-buffer

Determining what  $z$  to assign for each pixel covered by a plane is easy if the pixels are considered to be points. However, if the pixels are to be squares, then the plane covers a range of  $z$ -values, from which we might choose the min or max perhaps.

##### 4.3. Two crossing lines

Consider two lines,  $L_1$  and  $L_2$ , in three dimensions. If  $L_1$  crosses in front of  $L_2$ , as seen by the viewer, then we might expect that if  $L_1$  and  $L_2$  cause the same pixel to be marked in the frame buffer, then that pixel's final colour should be that of  $L_1$ . This is impossible in general.

For example, consider the following two lines:

$$\begin{aligned} L_1 : x = 1, z = 6y + 2 \\ L_2 : x = 3y, z = 9y \end{aligned}$$

In continuous space, they cross at the point  $(1, 1/3)$ , where  $z_1 = 4 > z_2 = 3$ . However, they overlap at the pixel  $(1,0)$ , where, using the second digitizing method described above,  $z_1 = 2 < z_2 = 3$ . Thus although  $L_2$  is really closer than  $L_1$ , it appears farther in the Z-buffer.

The situation gets worse. Consider a set of lines,  $M_i$ , covering every pixel, and a single line,  $L$ :

$$\begin{aligned} L : x = 1, z = 6y + 2 \\ M_i : y = x/3 + i, z = 3x + 6i \end{aligned}$$

Now  $L$  crosses  $M_i$  at  $(1, i + 1/3)$ , where  $z_L = 6i + 4 > z_M = 6i + 3$ . However in the raster domain, they cross at pixel  $(1,i)$ , where  $z_L = 6i + 2 < z_M = 6i + 3$ . This means that although  $L$  is behind all of the  $M_i$  in the continuous domain, it is completely visible in the Z-buffer.

Next let us fit a plane through all the  $M_i$ , we get

$$M : x + 6y - z = 0$$

The plane  $M$  is in front of  $L$  everywhere in the buffer, so there is a difference whether we use a plane or a set of lines in the plane.

#### 4.4. Line and plane

Another principle that users might want is that if we have a plane and a line on it, then the line should be visible in front of the plane in every pixel that it covers in the Z-buffer. Many of the digitization methods for lines described above will have an undesirable effect: the line will be in front of the plane for several pixels, and then behind it for several pixels. This is a supreme jaggy effect.

One simple solution is to digitize the lines and planes thus:

- Consider the pixels to be squares, and the lines and planes mark any pixel whose square they pass through.
- When a line passes through a pixel, choose the minimum z-value that the line has anywhere in the pixel.
- When a plane passes through a pixel, choose the maximum z-value that the plane has anywhere in the pixel.

With this standard, any line that is exactly on a plane will appear in front of it in the Z-buffer. However, lines that are slightly behind the plane may also appear in front at certain pixels. This is unavoidable. This method was designed assuming that slightly behind lines are less common than lines on the plane.



#### 4.5. Line and point

Alternatively, we may have lines and points that we may wish to insert into the Z-buffer such that if a point is on a line, then it overrides the line at that pixel. One method that works is this:

- Consider the pixels as squares again.
- If a point is in a certain pixel then use the point's z-value to mark that pixel.
- If a line passes through a certain pixel, then use that line's *maximum* z-value, not the minimum as before.

Now a point on (or even slightly behind) a line will appear in front of the line in the Z-buffer. The problem with this method is that it is incompatible with line and plane method in the previous section. It is an open question whether points, lines, and planes can all be painted into a Z-buffer, with the points in front of the lines that they are on, and the lines in front of the planes that contain them.

#### 5. The subset line problem

Even in two dimensional raster graphics it is difficult to determine an unambiguous part of a line to erase it. This problem can be formalized thus:

- We wish a line digitization algorithm to have the following property:
- Assume that we have digitized a line,  $L$ , between two pixels  $A$  and  $B$ , that is we have determined a set of pixels,  $S$ , in between  $A$  and  $B$  that are to be marked to signify the line.
- Assume that we pick two pixels,  $C$  and  $D$ , from  $S$  and digitize a line,  $M$ , between them by setting the pixels in the set  $T$ .
- Then, since  $M$  is clearly ideally a subset of  $L$ , we want our digitization algorithm to cause  $T$  to be a subset of  $S$ . The problem is not trivial since we must digitize  $M$  without any knowledge of  $L$ . Thus for all lines  $L$  whose set of pixels,  $S$ , contains  $C$  and  $D$ , we require that  $T$  contains  $S$ .

It is easy to show that the Bresenham algorithm doesn't satisfy this property. In fact the following stronger property holds.

Let a *translation invariant* digitization algorithm be one which just depends on the difference between the line's endpoints and not on their absolute coordinates. Then no translation invariant algorithm has the subset line property unless it is undesirable in certain other ways.

It is easy to get a digitization algorithm with the subset line property if we relax the constraint that the pixels marked be within one half pixel of the actual continuous line. For example, we could draw a line from  $A(x_1, y_1)$  to  $B(x_2, y_2)$  by going horizontally from  $A$  to  $(x_2, y_1)$  and then vertically to  $B$ . However this is not a useful digitization.

Although there are some preliminary results, this is still an open problem: it is not even known whether a good algorithm exists, let alone how complicated it might be. It is conceivable that the simplest good algorithm might have to explicitly store the digitizations of all  $1000^4$  lines on a 1000 by 1000 screen, and use table lookup. Such a solution would be of theoretical interest only.

## 6. Summary

The study of low level raster algorithms is important because first, theoretical gaps in them will cause problems in any higher algorithms that use them, and second, until the simple algorithms are understood, there is no reasonable hope of a detailed understanding of more complex algorithms. We have seen how some problems, such as handling crossing lines and planes, have no perfect solution, and how other problems, such as the subset raster line problem, remain open.

## References

(Gentleman and Marovich, 1974)

W.M. Gentleman and S.B. Marovich. "More on Algorithms That Reveal Properties of Floating Point Arithmetic Units", *Comm. ACM* 17, (5), (May 1974), pp. 276-277.

(Malcolm, 1972)

M.A. Malcolm. "Algorithms to Reveal Properties of Floating Point Arithmetic", *Comm. ACM* 15, (11), (November 1972), pp. 949-951.

(Spiegel, 1963)

M.R. Spiegel, *Theory and Problems of Advanced Calculus*, Schaum's Outline Series, McGraw-Hill, New York, (1963).

(Yasuhara, 1971)

A. Yasuhara, *Recursive Function Theory and Logic*, Academic Press, New York, (1971).



