UNIVERSITY OF WATERLOO
UNIVERSITY OF WATERLOO
UNIVERSITY OF WATERLOO
COMPUTER SCIENCE DEPARTMENT
COMPUTER SCIENCE DEPARTMENT
COMPUTER SCIENCE DEPARTMENT

Computing A New Class of Paint Brushes

Dan Field

CS-85-29

August, 1985

Computing A New Class of Paint Brushes

Dan Field

Computer Graphics Laboratory
Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1

ABSTRACT

Smearing, shifting, and embossing effects can all be achieved with digital filtering techniques. These effects could be a powerful artistic tool for paint and image processing systems but are typically not included because their computation is too costly. A technique for simulating these and other filters is described that speeds computation and reduces lookup table size over conventional filter evaluation. This is accomplished by retaining information about pixel contents that have already been swept by the brush and by mapping the 360 degree range of brush paths to a single direction. At the heart of the technique is an algorithm, interesting in its own right, that orders pixels according to their perpendicular distance from the cross section of the brush independent of its path.

1. Introduction

Typical paint systems provide brushes that deposit a set of pigments at specified locations along the brush swath independent of surrounding pixels. Many more interesting effects can be achieved by allowing the brush to use the existing image (or possibly images) in the process of deciding what the new color of every pixel along the swath is to be. The explode and ripple operators of Warpitout[Veed83] and the smear and slide brushes of the NYIT paint program [Smit78] are examples of brushes in this category. In their full generality a paint brush and path can be considered as a set of digital filters located at every pixel of the original image. Because of their computational expense, general brushes would rarely be implemented in a paint system.

This paper describes a class of brushes that is simpler and computationally more tractable than a fully general brush, yet provides more versatility than those available with a typical paint system. A few of the effects that can be achieved with these brushes are:

Smearing

A brush is dragged across a wet canvass causing paint to be moved a distance along the brush path.

Shifting

The image under the brush shifts left or right across the perpendicular of the brush path. The image can be made to ripple back and forth, fall off the side of the brush and cycle back to the other, or just fall off the side and remain. See Fig. 1.

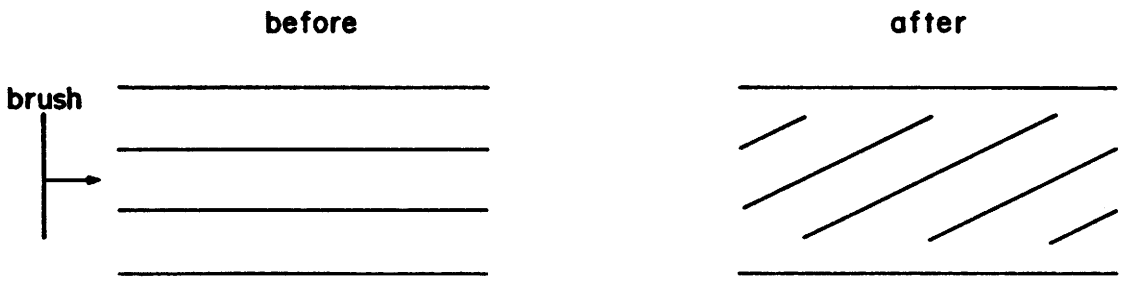

Figure 1. The shift brush move horizontally from left to right across the image on the left producing the image on the right.

Embossing

A sheet of paper is placed over a bumpy surface and a pencil is wiped over the paper transferring an image of the surface to the paper.

A conventional implementation for these brushes would involve convolving a neighborhood of pixels for each pixel under the brush. This results in multiple read operations for the same pixel. We save time by maintaining a 1 dimensional state table containing pertinent information about all pixels swept by the brush. Thus these brushes are one-sided in the sense that portions of the image not yet swept by the brush do not affect pixels currently being modified. A consequence is that the brushes are highly directional; sweeping the same path in opposite directions will produce different results.

Lookup tables are often used to reduce the time involved in computing a brush. Alternatively, a set of rules describing the computation for each pixel may also be used. For the class of brushes considered here, there are 3 degrees of freedom in the description of each pixel touched by the brush; angle and distance from the current brush position and angle of brush path. See Fig. 2. However, 2 of these variables may be eliminated since the brushes are one-sided. All that is needed to identify a pixel is its perpendicular distance p from the center of the path. See Fig. 3. Once a pixel has been swept by the brush its information is retained in the state table. Thus lookup table size and/or the number of rules describing update operations is greatly reduced.

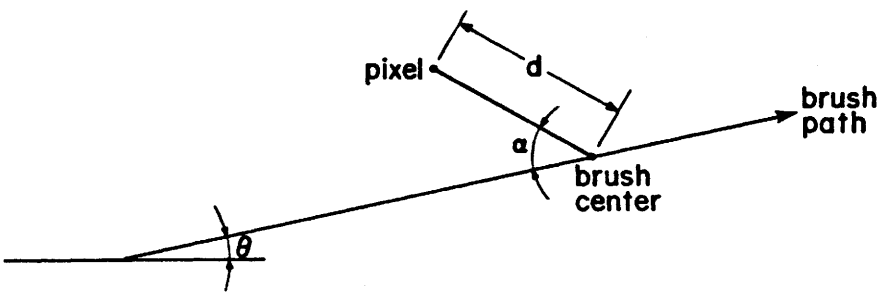

Figure 2. The location of a pixel swept by the brush is is described by the angle θ of the brush path from the horizontal, the angle α between the brush and the brush path, and the distance between the brush path and the pixel when swept across by the brush.

Four operators are applied to update the state table:

1) U:
How the state table changes as the brush moves from pixel to pixel,

2) I:how a new pixel changes the current state,

3) C:
how the current state determines the contents of a pixel once it has been swept over, and

4) D:
how the current state is updated by the modification of a pixel.

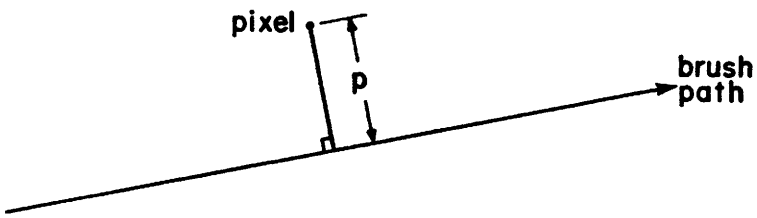

Figure 3. For a one-sided brush, only the perpendicular distance between the brush path and the pixel is necessary to identify the location of a pixel.

The richness of U, I, C, and D, and the order in which they are performed make many different brushes possible. A difficult problem that is not addressed here is that of providing a convenient mechanism for the artist to define new brushes in the class.

At the heart of the technique for updating the state table is an algorithm that orders pixels according to the time they cross the brush. The algorithm is interesting in its own right and may be used to speed computation of more conventional brushes.

2. Computing the Brush

The *brush path* is an oriented segment defined by a initial point (x_0,y_0) and a final point (x_1,y_1). The *sweep line* is a line segment of length w that is centered on and runs perpendicular to the brush path. The *brush swath* is the set of pixels that is touched by the sweep line as it moves from (x_0,y_0) to (x_1,y_1). See Fig. 4. Unlike typical brushes that have area and cause some pixels to be modified multiple times[Fish84], our brush modifies pixels in the swath exactly once when they are crossed by the sweep line.

For the moment, assume that there is a queue Q containing records of the form (t,p,x,y,r,g,b). Each record corresponds to a pixel in the brush swath where t is the perpendicular distance of the pixel from the initial position of the sweep line, p is the perpendicular distance of the pixel above or below the brush path ($-w/2 \leq p \leq w/2$), (x,y) is the location of the pixel, and r,g,b are its red, green, and blue components. Assume also that the records in Q are ordered by increasing t. The algorithm to maintain Q is described in section 4.

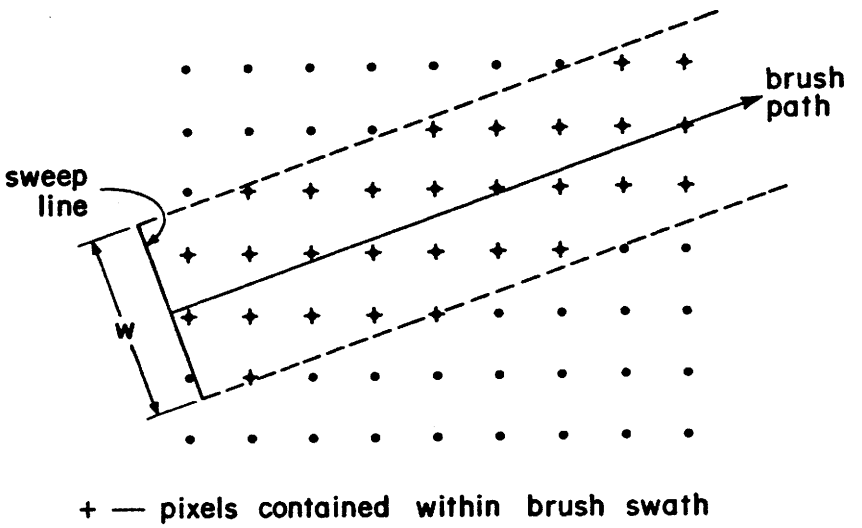

Figure 4. Pixels marked by a + are contained within the brush swath, all others are not.

The following pseudo-algorithm describes how the brush is computed.

for

each (t,p,x,y,r,g,b) in Q

do

- update the state table with U
- insert (t,p,x,y,r,g,b) in the state table with I
- obtain (r',g',b') for pixel (x,y) with C
- update the state table with D

od

The brush sweeps from the initial to final points performing the updates in the **for** loop. The order of the updates may be interchanged for different effects.

2.1. The State Table

The state table T is an array containing information about pixels that have been crossed by the sweep line. Each index position i in T corresponds to a unique distance p above or below the brush path; information about r,g,b values for the portion of the image that lies p away from the path is stored at T(i). See Fig. 5.

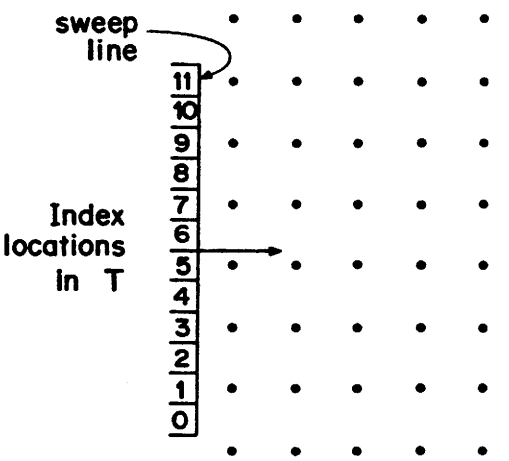

Figure 5. When a pixel is swept by the brush, the distance from the brush path is used to determine the appropriate state table entry to be updated.

Pixels crossing the sweep line do so at various distances p from the path depending upon the path angle and length. When a pixel crosses the sweep line, the index i of T that most closely corresponds to its distance p may be found with the following formula:

$$i = \left\lfloor (p+w/2)\frac{n}{w} \right\rfloor \tag{1}$$

The sub-pixel resolution of T is w/n; increasing the number of table entries causes the correspondence between pixel distance from the path and table index to be more accurate at the expense of greater update times. Even for large n, pixels will fall between the table entries and a form of aliasing occurs. Aliasing artifacts can be overcome by distributing pixel values over several adjacent table entries when using operator I.

2.2. The U Operator

As the sweep line moves from a pixel at t_0 to a pixel at t_1, T is updated to reflect a move of distance t_1-t_0. One possibility for U is to modify each entry by a function f independent of index number and adjacent table entries:

$$T(j) = f(t_1-t_0, T(j)) \tag{2}$$

for $0 \leq j < n$. Smearing is accomplished by defining f to be a decreasing linear or exponential function of parameter t. Another possibility is to shift the contents of T k index positions depending upon the distance t_1-t_0. Values falling off the ends may either disappear, accumulate in the last index position or wrap around.

In general, T(0),T(1),T(2),... can be considered as a waveform sampled at intervals along the sweep line. The U operator first reconstructs the continuous signal, modifies it, and resamples back into T. In some cases, much of the time involved in computing U can be eliminated through the use of precomputed lookup tables that are independent of the brush path.

2.3. The I Operator

T must be updated to reflect the r,g,b values of pixels entering the brush. In the simplest case, I simply adds r,g,b from the current queue record to $T(i)$ where i is from Eq. (1). To eliminate aliasing artifacts caused by single-valued correspondences between i and p, it is best to have I operate over a neighborhood of table indices centered at i. Gaussian and $\frac{\sin x}{x}$ weighting distributions do quite well at eliminating aliasing[Smit83]. The weights would be symmetric around 0 and sum to 1. The new table T would then be expressed:

$$T(j) = T(j) + w_{in}(i-j) f(r,g,b) \tag{3}$$

where $0 \leq j < n$, w_{in} is the weighting function, and f is either a single or multivalued function of the input pixel value. Initial shift effects can be produced by adding constant or variable offsets to p before computing i.

2.4. The C and D Operators

The C operator produces the r,g,b values for a new pixel. If p is the distance from the pixel to the brush path, we extract information from T near the corresponding index i. These values can be expressed as a convolution:

$$(r,g,b) = \sum_j w_{out}(i-j)\mathrm{T}(j) \tag{4}$$

where $0 \leq j < n$, and w_{out} is a set of output weights. Typically, the convolution would be an average of several surrounding table entries because T is essentially a supersampled version of the image under the brush. In the simplest case, the D operator sets entries $\mathrm{T}(j)$ to zero anywhere $w(i-j)$ is nonzero, indicating that the pixel has been removed from the table. Various shift effects can be produced by using different values for i of C and D.

3. An Example: Embossing

If a pencil lead is rubbed across a sheet of paper on a smooth surface, a uniform shade of gray is produced. If we now place a flat object of greater height under a portion of the paper and rub the lead from the lower surface to the higher surface, a uniform shade of gray appears followed by a dark band at the joint between the two heights followed by another uniform shade of gray. Rubbing the lead in the opposite direction produces a lighter band at the joint.

This artistic technique can be simulated digitally in the following manner. Consider a 1 dimensional signal x_t sampled at $t=0,1,2....$ A new signal y_t is produced from the x_t modeling the behavior of the pencil lead. Let $d_t = x_{t+1} - x_t$. If all $d_t = 0$, then $y_t = b$ for all t. The quantity b is called the *bias* and corresponds to a uniform gray shade on a smooth surface.

If $d_t \neq 0$ then the surface has either bump or a pit, and the lead produces a darker or lighter region. Typically this region persists slightly past the surface irregularity and eventually settles back to the uniform gray color. This behavior can be modeled by defining

$$y_{t+1} = b + sign(y_t - b)\,|y_t - b\,|^{1/c} + d_t \tag{5}$$

where $c > 1$ and indicates how far y is affected past the irregularity in x. Various quantities in Eq. (5) can be scaled to achieve different effects. For example, the importance of a change in x can be exaggerated by scaling d_t by a positive constant greater than 1.

The embossing effect can be produced by modifying the procedure described in the previous section. For simplicity, assume that the underlying image is defined by luminosities l between 0 and 255 and the r,g,b fields in the queue records have been replaced by l. The image can be thought of as a surface defined by the coordinates (x,y,l) with luminosity changes corresponding to variations in surface height.

Let t be the current position of the sweep line and (t_0,p,x,y,l) be the next queue record. The U operator causes the state table to settle back to the bias value:

$$T(i) = b+sign(T(i)-b)\,|T(i)-b\,|^{\frac{1}{c+t_0-t}} \tag{6}$$

Note that exponential behavior can be achieved by multiplying $T(i)$ by different fractions less than 1 depending upon t_0-t.

To find the variations in surface height, an additional state table S is maintained that retains the most recent surface height. The difference between the recent height in S and l of the current pixel is then added to T; the old height in S is then replaced by l. The weights w_{in} distribute l over several table indices to reduce aliasing effects.

Let i be the table index found from p by Eq. (1). The height of a pixel should affect only table entries indexed by j for which $w_{in}(i-j)\neq 0$. The I operator computes

$$T(j) = T(j)+S(j)-w_{in}(i-j)l \tag{7}$$

and then updates S;

$$S(j) = w_{in}(i-j)l \tag{8}$$

for these non-zero entries.

The C operator simply averages several adjacent values in T around index i using w_{out}. The results of averaging are clipped to the range [0,255].

The D operator is not used; in some sense it has been incorporated within I in the update of S.

Fig 6 contains an embossing using Mount St. Helens as the underlying surface. The brush was swept horizontally from left to right and covers the entire image. Each of the r,g,b channels were treated separately; different biases, decay exponents, and scaling constants were used for each channel.

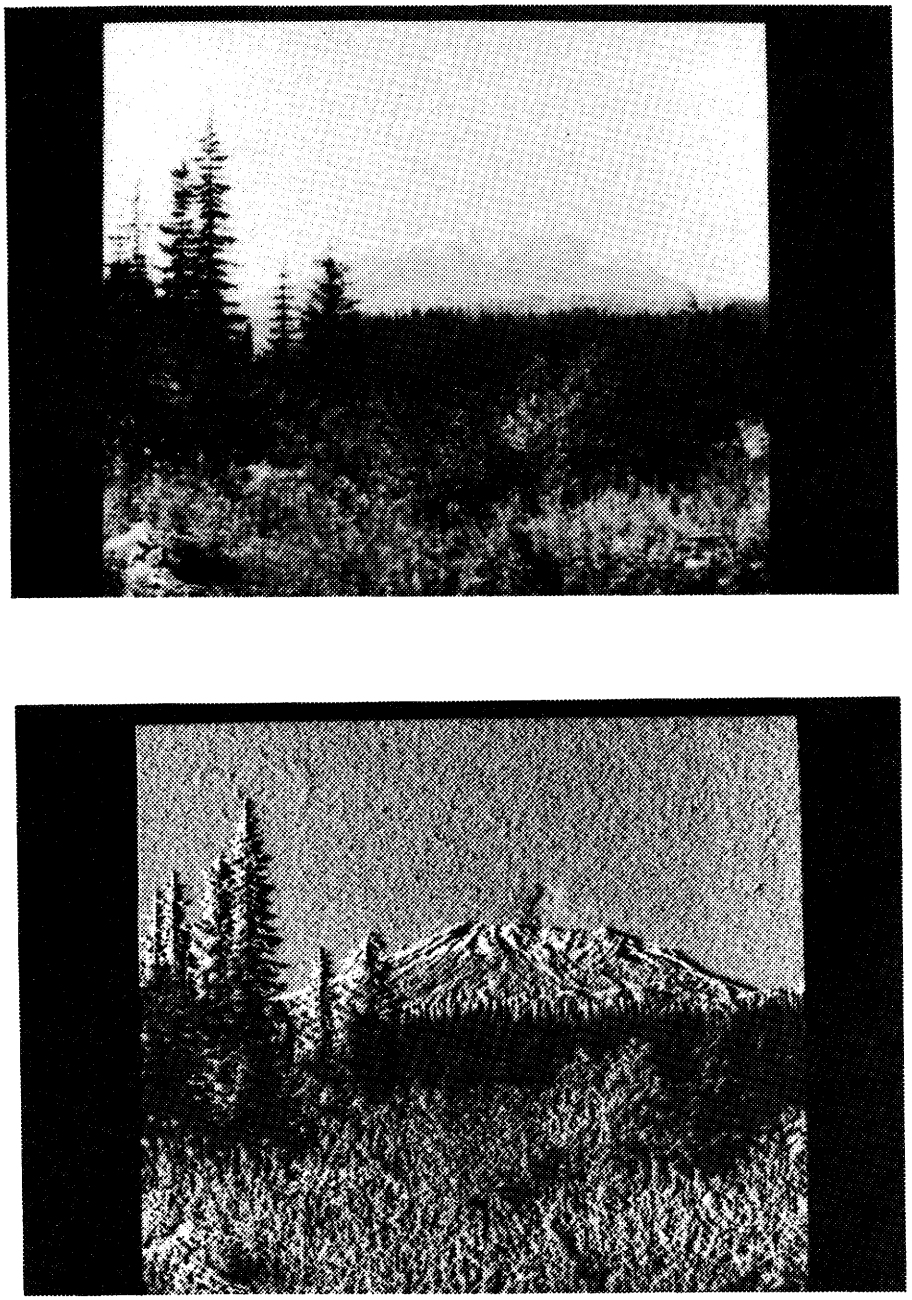

Figure 6. The original image of Mount St. Helens above, and the result of embossing below.

4. An Algorithm For Maintaining Q

The queue Q contains pixels in the swath ordered by their perpendicular distance from the initial position of the sweep line. Instead of identifying and sorting these pixels all at once, an incremental update algorithm is employed. After Q has been initialized, every deletion from the front causes one or two new pixels to be added to the back.

Without loss of generality, it will be assumed that the brush path lies in the first octant; that is, $0 \leq y1-y0 < x1-x0$. At any point, Q contains the pixels in the swath that are in an 8-connected region lying to the right of the sweep line that block the line from the rest of the swath. More formally, if pixel (x,y) lies to the left of the sweep line and pixels $(x+1,y)$ and $(x,y+1)$ lie to the right of the sweep line, then $(x+1,y)$, $(x,y+1)$, and $(x+1,y+1)$ will be in Q if they are in the swath. See Fig. 7.

When the sweep line crosses a pixel at (x,y), the hole in the 8-connected region to the right of the line must be closed. This is accomplished by placing pixel $(x+1,y+1)$ on the tail of Q as long as it lies within the swath. Containment in the swath is tested using the values of t and p of the original pixel at the head of Q. Formulas for performing this test are described below.

It is possible for pixel (x,y) but not $(x-1,y-1)$ to be in the swath. Such pixels lie along the lower edge of the swath and would not be inserted in Q by the above scheme. See Fig. 7. They can be included by maintaining a *check pixel* (x_c,y_c) that is in the 8-connected region of the sweep line but is just below the swath boundary. Every time the brush moves the check pixel is tested to see if it has been swept across. If the pixel is in the swath, then before the record at the head of Q is removed, pixel (x_c+1,y_c+1) is tested to see if it is in the swath. If the pixel is in the swath, then it is added to the tail of Q and (x_c+1,y_c) becomes the new check pixel. If not, it becomes the new check pixel. See Fig. 7.

Let t_0 and t_1 be the perpendicular distances from pixels (x,y) and $(x+1,y+1)$ to the initial position of the sweep line. It can be shown that:

$$t_1 = t_0 + \sqrt{2}\cos\rho \tag{9}$$

where $\rho = \frac{\pi}{4} - \theta$, and $\theta = \tan^{-1}\frac{y1-y0}{x1-x0}$. That is, θ is the angle between the brush path and the x axis. If t_2 is the perpendicular distance from pixel $(x+1,y)$ to the initial position of the sweep line, then:

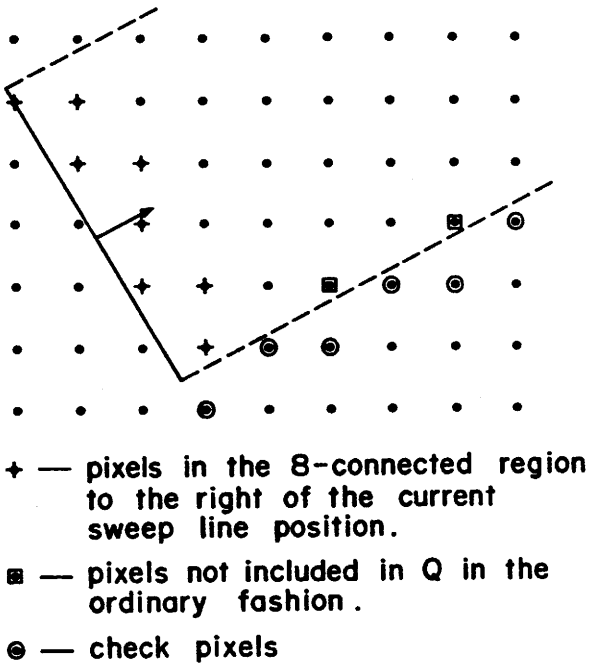

Figure 7. The diagram indicates the relationship between pixels, the current sweep line position, and brush swath. Pixels labelled + are in the 8-connected region to the right of the current sweep line position. Pixels labelled by a square are not included in Q in the ordinary fashion. Check pixels are labelled by an encircled dot.

$$t_2 = t_0 + \cos\theta \tag{10}$$

Let p_0 and p_1 be the perpendicular distances from pixels (x,y) and $(x+1,y+1)$ to the brush path. It can be shown that:

$$p_1 = p_0 + \sqrt{2}\sin\rho \tag{11}$$

where ρ and θ are as defined above. If p_2 is the perpendicular distance from pixel $(x+1,y)$ to the brush path, then

$$p_2 = p_0 + \sin\theta \tag{12}$$

A pixel is contained in the swath if and only if its corresponding t and p values satisfy the following inequalities:

$$-w/2 \leq p \leq w/2 \tag{13}$$

$$0 \leq t \leq \sqrt{(x1-x0)^2+(y1-y0)^2} \tag{14}$$

We can now prove the following lemmas asserting that the algorithm for maintaining Q is correct.

Lemma: Every pixel in the swath is inserted in Q.
Proof: Assume that Q has been initialized to contain the pixels in the 8-connected region to the right of the initial position of the sweep line. For some y, let l be the smallest integer such that pixel $(l,y+l)$ is in the swath but never in Q. The rule for insertion in Q implies that $(l,y+l)$ must lie along the lower edge of the swath and that the edge of the brush must have swept between $(l-1,y+l-1)$ and $(l-1,y+l)$. But this implies that $(l-1,y+l-1)$ must have been a check pixel and that $(l,y+l)$ would have been inserted in Q.

Lemma: Every pixel is inserted in Q in increasing order of t.
Proof: Let (t_0,p,x,y,r,g,b) be the record at the front of Q. Assume that Q is ordered correctly at the initial position of the sweep line. Initially, we have $t_i-t_0 \leq \sqrt{2}\cos\rho$ for all the records i in Q since the pixels in the 8-connected neighborhood lie within $\sqrt{2}\cos\rho$ of the line. The pixel placed on the end of Q generated by the record at the head has $t=t_0+\sqrt{2}\cos\rho$ and therefore $t \geq t_i$ for all records i in Q. Pixels may also be inserted in Q when the check pixel is crossed by the sweep line. But all the pixels in Q are within $\sqrt{2}\cos\rho$ of the sweep line at the check pixel and the pixel inserted in Q has $t=t_c+\sqrt{2}\cos\rho$, where t_c is the perpendicular distance of the check pixel from the initial position of the sweep line. Thus, $t \geq t_i$ for all pixels i in Q. Since all pixels inserted in Q have larger values of t than those already there, and Q was initially ordered, every pixel must have been inserted in Q in the correct order.

We mention 2 possibilities for obtaining the initial contents of Q for the starting position of the sweep line. The first determines the pixels that belong in Q and then sorts them by their t values. The second is able to generate the pixels in the correct order.

1) Consider the left endpoint of the brush path (x_0,y_0) as the current point, CP, and insert it on a list L of unordered pixels. Let CP=(x,y) and check if $(x-1,y)$ is in the brush swath. If it is, set CP=$(x-1,y)$, and append it to L. If it isn't, then it is above or to the left of the swath. If it is to the left, then $(x,y+1)$ must be in the swath; set CP=$(x,y+1)$ and append it to L. If it is above, then all pixels above the brush path that initially belong in Q have been appended to L. The pixels below the brush path are generated and appended to L in a similar fashion. The list L is then sorted by increasing

value of t. This technique works by staying as close as possible to the initial position of the sweep line.

2) In [Fiel84], algorithm *EXACT_F1* is described that draws anti-aliased line segments. The algorithm generates pixel locations ordered by increasing vertical and hence perpendicular distance from the line. Basic number-theoretic results are employed to rearrange the order of pixels produced so that the remainder values produced by Bresenham's algorithm [Bres65] are increasing. Thus, the initial contents of Q can be found without sorting.

The first algorithm is straightforward but requires sorting at most $2w$ values. For thin brushes the cost of sorting could be negligible compared to the overall cost of computing the image that results under the brush. The second algorithm doesn't require sorting, is more complicated to implement, and requires that the greatest common divisor of $y1-y0$ and $x1-x0$ be computed.

5. Analysis

A conventional means of finding the result of brushing over a single pixel (x,y) is to evaluate some function based on the contents of pixels lying in a neighborhood of (x,y). For the class of brushes described here, the neighborhood consists of pixels that have already been passed over by the sweep line. One complication of this scheme is that different functions and neighborhoods must be used for each brush path angle. Either extra preprocessing time for each brush stroke angle or large numbers of lookup tables are required. Another problem that must be overcome is that the pixels used to compute the new value of (x,y) have already been modified by the brush. This necessitates the use of a pixel buffer or cache [Whit84] to retain their old contents.

Suppose that the neighborhood has size $w \times d$; that is, all pixels in the swath less than distance d behind the sweep line are examined to determine the new value for each pixel. If the brush path has length l, then the swath contains $O(wl)$ pixels. Thus, $O(dw^2l)$ operations are required to compute the result of the particular brush path.

In contrast, the algorithm we have described requires $O(n)$ work for each pixel in the swath because each entry in table T may be examined and/or modified every time the sweep line moves. Since $n=cw$, our algorithm requires $O(cw^2l)$ operations. It takes $O(w)$ time to initialize Q if the second alternative is employed. Thus, as long as $d>c$, our technique is faster than that described above. Typically, c is a small constant, say between 2 and 4, while d can vary greatly depending upon the filter being used.

6. Conclusion

A technique has been described for computing a class of paint brushes that is a faster alternative to methods of computing fully general brushes. An incremental algorithm for producing pixels in the order they cross the line perpendicular to the brush path is the basis of our brush computation technique. The algorithm can also speed the computation of more conventional brushes if the brush "footprint" [Fish84] can be collapsed to a single line.

The class of brushes described seems to be able to produce a wide variety of effects. More analysis and experimentation will be necessary if the class is to be fully categorized.

In general, paint systems do not allow the artist to simulate many of the effects producible in an analog medium. Paint splashing, texturing, shadow, and lighting effects have yet to be realized. It is also possible to simulate photographic effects available in the darkroom. It is a challenge of the future to provide the myriad of possibilities to the artist in the realm of computer graphics.

References

[Bres65] BRESENHAM, J.E. Algorithm for computer control of a digital plotter. *IBM Systems J. 4*, 1 (1965), 25-30.

[Fiel84] FIELD, D. Two algorithms for drawing anti-aliased lines. *Proc. Graphics Interface '84*, (May, 1984), 87-95.

[Fish84] FISHKIN, K.P. AND BARSKY, B.A. Algorithms for brush movement in paint systems. *Proc. Graphics Interface '84*, (May, 1984), 9-16.

[Smit78] SMITH, A.R., "Paint," Technical Memo No. 7, NYIT (July 1978).

[Smit83] SMITH, A.R., "Digital filtering tutorial for computer graphics," Technical Memo No. 27, Lucasfilm Ltd. (Mar. 1983).

[Veed83] VEEDER, J., *Warpitout*, Siggraph Video Review, Issue #8 1983.

[Whit83] WHITTED, T. Anti-aliased line drawing using brush extrusion. *Computer Gr. 17*, 3 (July 1983), 151-156.