Parallel Line: a Unified Solution

Ivo Považan* Tomáš Hrúz†

October 28, 2000

Abstract

This paper describes a new class of parallel algorithms for line interpolation in raster space. A unifying approach to the line interpolation algorithms is also presented. All algorithms are derived from one inequality.

1 Introduction

To create an algorithm approximating a line in raster space is simple. It turned out not to be so simple to find an algorithm optimized with respect to data types and a number of arithmetic operations. A widely used algorithm was proposed by Bresenham [3] in 1965. Since then, further improvements have been published (See [4] and references there). The number of operations for those algorithms mentioned above is linear with respect to the number of points approximating a line. The recent trend in research of sequential line interpolation algorithms consists of various attempts how to reduce an overall number of operations. These algorithms determine more than one point during an iteraion [7, 10]. Another possibility how to speed up the algorithm is to preserve certain structure of raster space in a table [5].

The paradigm of parallel architectures have brought an attention to another aspect of line interpolation i.e. on the possibility to make the interpolation parallel. In [13] the author breaks down a given line into smaller parts according to the number of processing elements. The main problem here is to determine initial conditions for all processors. Another approach is presented in [9] where a processing element is assigned to each pixel in raster space. The Connection Machine library [1] implements an algorithm that assigns a processing element to every line segment, then all line segments are rendered in parallel. The *scan operation* as is defined in [2] can be also used for parallel line interpolation.

The line interpolation problem in raster space can be split into two parts. Firstly, there is a problem how to determine quickly the integer coordinates of the points

*Ivo Považan, Institute of Control Theory and Robotics, Slovak Academy of Sciences, Dubravská cesta 9, 842 37 Bratislava, Slovak Republic, Phone: +42-7-3782985, e-mail: utrrpova@savba.savba.cs

†Tomáš Hrúz, Slovak Technical University, Faculty of Mechanical Engineering, Department of Automatic Control and Measurement Námestie Slobody 17, 812 31 Bratislava, Slovak Republic, Phone: +42-7-3594-571, 497193 e-mail: hruz@vm.stuba.sk Fax: +42-7-495315

approximating a given line segment. Secondly, contents of a video buffer must be updated in these points. To achieve maximal speed up both parts should be parallelized. However, in our contribution we deal explicitly with the first part of the problem, but implicit consequences for the second part can be derived as well.

In figure 1 is a line segment and its raster space approximation. It is intuitively clear that for the line in the first octant (the derivative is in the interval of $< 0, 1 >$) we obtain good approximation if on every vertical line of the mesh lies exactly one point of the line approximation. Its y-coordinate is then determined by the distance of the intersection and the nearest grid point. An accurate formulation of the line segment approximation (or digitalization scheme) can be derived from general principles (for references and also for the approximation with lattices see [14]).

Intuitively one can say that the set M of grid points approximating a line should have the following properties: starting and ending points have to be in M, cardinality of M must be minimized as well as the distance from the line. On the other hand M should satisfy some connectivity condition in a discrete sense.

We call raster space a set $\mathbf{Z}^2$ (all integer pairs) which is interpreted as embedded in the Euclidian plane $\mathbf{R}^2$. Sometimes we call elements of raster space grid points. The first octant is a set of integer pairs (x, y) defined as: $\{(x, y) \in Z^2 \subset R^2 \mid 0 \leq y \leq x\}$. As is usual, we suppose that the start-point and the end-point of a given line are integral. Moreover, without loosing generality we can suppose that the start-point $S = (0, 0)$ and the end-point $E = (H, V)$ lie in the first octant. Otherwise, a well known simple transformation [3] can be used which maps the line to the first octant.

The standard digitalization scheme which we employ here is sometimes called mid-point digitalization scheme [6]. It leads to the above mentioned situation where on each vertical line of the mesch we have exactly one point. The approximation of a line by the mid-point digitalization scheme is a set of all integer pairs (x, y) which fulfill the following inequality

$$y - \frac{1}{2} < \frac{V}{H} x \leq y + \frac{1}{2} \tag{1}$$

where x satisfies the condition $x \in \{0, 1, 2, ..., H - 1, H\}$. Linearity implies also that $y \in \{0, 1, 2, ..., V - 1, V\}$.

We can solve the inequality (1) by reformulating it to the form of a system of $H + 1$ inequalities where x is running through the set $\{0, 1, 2, ..., H - 1, H\}$ of by reformulating it to the form of a system of $V + 1$ inequalities where y is running through the set $\{0, 1, 2, ..., V - 1, V\}$.

A unified approach to the interpolation algorithms which we propose means that all algorithms in the following sections are obtained by a certain reformulation of the basic inequality (1) to a form of an inequality system.

In sections 2 and 3 we use the above mentioned technique to obtain algorithms similar to the original Bresenham algorithm [3] and to the Bresenham SLICE algorithm [4]. We do not focus here on an optimization of these algoritms, instead we proceed with a derivation of a new parallel class of algorithms in sections 4 and 5.

2 Bresenham algorithm

Let us reformulate the basic inequality (1) to the form of $H+1$ inequalities:

$$-2H < 2Vx_i - 2Hy - H \leq 0$$

where x_i is running through the set $\{0,1,2,....,H-1,H\}$. This system of inequalities can be solved sequentially. Firstly, we solve 0-th inequality (for $x_0 = 0$) which has only one solution $y = 0$. $(i+1)-th$ inequality can be efficiently solved when we use the values computed during the $i-th$ inequality computation, especially the value of the expression $2Vx_i - 2Hy - H$ stored in a special control variable D. From this analysis the following algorithmic primitive BRES can be directly derived.

Algorithm A	Algorithm A'

var x, y – contain grid points
var D – contain $2Vx - 2Hy - H$
 	
1. *initialization*
  $x = 0; y = 0; D = -H$
 	
2. $output(x, y)$
2. $output(x, y)$
3. $if(x = H)$
    $finish$
4. $x = x + 1; D = D + 2V$
5. $if(0 \geq D > -2H)$
    $goto 2.$
6. $y = y + 1; D = D - 2H$
7. $goto 2.$
	var x, y – contain grid points
	var D – contain $2Vx - 2Hy - H$
	
	1. *initialization*
	  $x = 0; y = 0; D = 2V - H$
	  $Incr1 = 2V; Incr2 = 2V - 2H$
	2. $output(x, y)$
	2. $output(x, y)$
	3. $if(x = H)$
	    $finish$
	4. $if(0 \geq D)$
	    $D = D + Incr1$
	  $else$
	    $D = D + Incr2$
	    $y = y + 1$
	5. $x = x + 1$
	6. $goto 2.$

The algorithm A' from the above figure is the original Bresenham's algorithm [3] with notation from [6]. One can concern the algorithm A' as a *formal program tranformation* of the algorithm A. The theory of these transformations with respect to graphics algorithms is developed in [12] and also in [11]. Further, we focus ourselves only to the algorithmic variants directly following from some reformulation of the main inequality (1).

In order to express a complexity we will use operation $output(x, y)$ as an atom for the operation counting. Complexity is described as a function of a number N of the grid points approximating given line. The algorithms A and A' have complexity $N = H$.

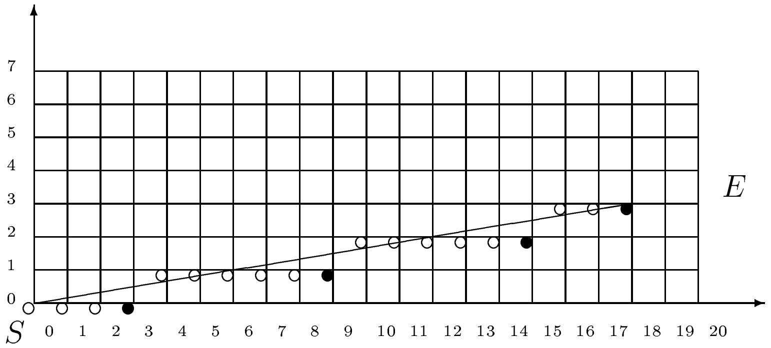

Figure 1:

The transition points are labelled with a bullet. Point E is also a transition point because of a constraint in the inequality (1).

3 Bresenham SLICE algorithm

Let us define $\frac{H}{2V} = c_0 + \frac{r_0}{2V}, \frac{2H}{2V} = c_1 + \frac{r_1}{2V}, r_0 = mod(H, 2V), r_1 = mod(2H, 2V)$. Then the basic inequality (1) can be modified to the form of $V + 1$ inequalities:

$$c_1 y_j - c_0 + \frac{r_1 y_j - r_0}{2V} < x_i \leq c_1 y_j + c_0 + \frac{r_1 y_j + r_0}{2V}$$

where y_j is running through the set $\{0, 1, 2, ..., V - 1, V\}$. This system of inequalities can be solved sequentially. It holds that every inequality has at least one solution x_i, some of them can have more. First of all, the algorithm search for the greatest element x_i for every inequality. At the same time the algorithm uses the property that the smallest solution for the $(i+1)-th$ inequality is exactly the largest solution of the $i - th$ inequality plus one. We will call the transition point the grid point (x_i, y_j) where x_i is the greatest solution for the $j - th$ inequality mentioned above. Transition points are coloured black in figure 1. We would like to stress that the $(V + 1) - th$ transition point has x-coordinate H because of a constraint in the basic inequality (1). This analysis directly leads to the algorithm B in following picture. In fact the algorithm enumerates x-coordinates of transition points which are equal to $\lfloor c_1 y_j + c_0 + \frac{r_1 y_j + r_0}{2V} \rfloor$.

Algorithm B

variable Y – y-coordinate of a transition point
variable STARTX, ENDX – starting and ending x-coordinate
of the horizontal segment related to the transition point
variable MOD – contains mod $(r_1 y_j + r_0, 2V)$
 	
1. $initialization, Y = 0, STARTX = 0, ENDX = c_0, MOD = r_0$
2. $if(horizontal\_segment(STARTX, ENDX, Y) = FALSE)$
    finish
3. $STARTX = ENDX + 1, ENDX = ENDX + c_1,$
$MOD = MOD + r_1$
4. $if(MOD \geq 2V)$
    $ENDX = ENDX + 1$
    $MOD = MOD - 2V$
5. $Y = Y + 1$
6. $goto 2.$
 	
procedure $horizontal\_segment(STARTX, ENDX, Y)$
$initialization, X = STARTX$
$while(X \leq ENDX)$
    {
    $output(X, Y)$
    $if(X = H)$
        $return(FALSE)$
    $X = X + 1$
    }
$return(TRUE)$

The algorithm B is equivalent to the Bresenham's SLICE algorithm [4]. We do not focus here on the character of this equivalence. It can be again derived from theory in [12]. When we use $output(x, y)$ as an atom for the complexity counting, the complexity of the algorithm B is $N = H$. When we count passes through $horizontal\_segment()$ the complexity is V but the worst case is also $N = H$.

4 Parallel algorithm with $m = \lceil log_2(V + 1) \rceil$ complexity

Let us modify the basic inequality (1) to the form of the $V + 1$ inequalities:

$$\frac{H}{2V}(2y_j - 1) < x_i \leq \frac{H}{2V}(2y_j + 1) \tag{2}$$

We will describe a processor scheme which has $m = \lceil log_2(V + 1) \rceil$ rows of processors. In $k - th$ row we have 2^k processors. There is also one temporary processor dedicated to every row. The processor scheme is designed to compute efficiently $V+1$ transition points described by inequalities (2). The processor pattern is shown in the following picture. The processors in the specified position are signed by row and column values. In the parenthesis are values of the expression $(2y_j+1)$ which play an important role in the evaluation of the transition points as can be seen from inequalities (2). These values are only informal, because they are valid only after $k-th$ step of computation when $(k+1)-th$ row of processors contain values $\lfloor \frac{H}{2V}(1)\rfloor$, $\lfloor \frac{H}{2V}(3)\rfloor$, $\lfloor \frac{H}{2V}(5)\rfloor$, $\lfloor \frac{H}{2V}(7)\rfloor$, ... The temporary processors are signed only with row indeces with a similar comment about informal values.

row	processors				temporary
0	$P^{0,1}(1)$				$T^0(2^1)$
1	$P^{1,1}(1)$	$P^{1,2}(3)$			$T^1(2^2)$
2	$P^{2,1}(1)$	$P^{2,2}(3)$	$P^{2,3}(5)$	$P^{2,4}(7)$	$T^2(2^3)$
...	...				...
k	$P^{k,1}(1)$	$P^{k,2}(3)$	...	$P^{k,2^k-1}(2^{k+1}-3)$	$T^k(2^{k+1})$

When the computation proceeds every processor contains the value of the expression $\frac{H}{2V}(2y_j+1)$ in the variables C, R, E where $C = \lfloor \frac{H}{2V}(2y_j+1)\rfloor$, $R = mod(H(2y_j+1), 2V)$, $E = H$ and the temporary processors will contain values $\frac{H}{2V}(2^l)$ in C, R in the same way. The contents of variable $E = H$ is only copied to all processors, it will be used for testing the end condition. The processor scheme works as follows: there is an initialization and m steps of the computation. The initialization consists of a setting $P^{0,1}$ to $C = \lfloor \frac{H}{2V}\rfloor$, $R = mod(H, 2V)$, $E = H$, temporary T^0 is set to $C = \lfloor \frac{2H}{2V}\rfloor$, $R = mod(2H, 2V)$. The $k-th$ (k starts with 0) step consists of the following operations in the $k-th$ row:

1. Contents of all 2^k processors in $k-th$ row is copied to the $(k+1)-th$ row, i.e. $P^{k+1,j} = P^{k,j}$, $j = 1, \ldots, 2^k$
2. All processors in $k-th$ row do in parallel: $C_P = C_P + C_T; R_P = R_P + R_T; if(R_P \geq 2V)\{C_P = C_P + 1; R_P = R_P - 2V\}$ where indexes P resp. T indicate belonging to the P resp. T processors.
3. Contents of all 2^k processors from $k-th$ row is copied to the $(k+1)-th$ row beginning at the position 2^{k+1}, i.e. $P^{k+1,j} = P^{k,j}$, $j = 2^k+1, \ldots, 2^{k+1}$
4. The temporary processor in $k-th$ row do: $C_T = C_T + C_T; R_T = R_T + R_T; if(R_T \geq 2V)\{C_T = C_T + 1; R_T = R_T - 2V\}$
5. The contents of the temporary processor in $k-th$ row is copied to the temporary in the $(k+1)-th$ row.

When the process finishes ($k = m$) every element in the last row contains x-coordinate of the transition point. Now, one operation of copying a processor value to the neighbour leads to the situation when all processors contain sufficient information to do the *horizontal_segment*() operation. The appropriate notion of complexity is to count steps. We have $m = \lceil log_2(V+1)\rceil$ steps.

5 Parallel algorithm with 1 complexity

We start with the same inequalities (2) as in the previous section. Unlike the three algorithms above this algorithm does not use the values computed for the neighbouring inequalities in the transition point computation. A natural way to compute these $V+1$ inequalities is to have a column of processors each entry for one inequality. The algorithm has three steps:

1. The values H, V are spread in parallel to all processors
2. All processors compute $\lfloor \frac{H}{2V}(2y_j+1) \rfloor$, these are the x-coordinates of the transition points
3. All processors send their values to the neighbours so $horizontal\_segment()$ operation can be done

6 Conclusions

The algorithm from section 5 can be implemented on the Connection Machine [8, 1] in a straightforward way. We use PARIS instructions to show schematically how different structures of the Connection Machine can be used to implement individual steps of the algorithm. Firstly, we can define a column of $V+1$ processors by the *create-geometry* instruction. This allows us to use the efficient communication pattern of NEWS. The *write-to-processor* and *spread* instructions can be used to load the values H, V to all processors. Then all processors compute the value $\lfloor \frac{H}{2V}(2y_j+1) \rfloor$. Finally, the operation *send-to-news* can be used to send the information about transition points to its neighbours.

Section 4 describes only a basic scheme which can be varied in several ways depending on the topology of the parallel system. For example the scheme can be optimized according to the number of processing cells. It is possible to use distinct memory units and units computing modulo $2V$ add operations. This can lead to $\frac{V+1}{2}$ processing units and $V+1$ memory units.

We would like to stress that it prooves to be very fruitful to transfer as much as possible of the given graphic problem to raster space. Then further abstraction is appropriate: i.e. to see these problems as the problems of a structure of the set of integers.

References

[1] Connection Machine, Model CM-2 Technical Summary. Thinking Machines Corporation, May 1989.

[2] G. E. Blelloch, Scans as Primitive Parallel Operations. IEEE Transactions on Computers, Vol. 38, No. 11, p. 1526-1538.

[3] J. E. Bresenham, Algorithm for computer control of digital plotter. IBM Systems Journal, Vol 4, No.1, pp. 25-30, January 1965.

[4] J. E. Bresenham, Incremental Line Compaction. The Computer Journal, Vol. 25, No.1, pp. 116.

[5] N. D. Butler, A. C. Gay, J. E. Bresenham. Line Generation in a Display System, US patent 4,996,653, Feb. 26, 1991.

[6] James D. Foley, Andries Van Dam, Steven K. Feiner, John H. Hughes, Computer Graphics, Principles and Practise, second edition, Addison Wesley, Reading, Mass., November 1991.

[7] K. Y. Fung, T. M. Nicholl and A. K. Dewdney, Run-Length Slice Line Drawing Algorithm without Division Operations, Proceedings of EUROGRAPHICS '92, Vol. 11, No. 3, pp. C-267 - C-277, 1992.

[8] W. D. Hillis, The Connection Machine. MIT Press.

[9] A. T. Pang, Lina Drawing Algorithms for parallel machines. IEEE Computer Graphics & Applications, Vol. 10, No. 5, p. 54-59.

[10] J. Rokne and Y. Rao, Double-Step Incremental Linear Interpolation, ACM Transactions on Graphics, Vol. 11, No. 2, pp. 183-192, April 1992.

[11] R. F. Sproull, Using Program Transformation to Derive Line-Drawing Algorithms. ACM Transactions on Graphics, Vol. 1, No. 4, p. 259-273.

[12] S. M. Ecker and J. V. Tucker, Tools for the Formal Development of Rasterisation Algorithms. New Advances in Computer Graphics. Proceedings of CG International 89, p. 53-89.

[13] W. E. Wright, Parallelization of Bresenham's Line and Circle Algorithms. IEEE Computer Graphics & Applications, Vol. 10, No. 5, p. 60-67.

[14] C. A. Wutrich and P. Stucki, An Algorithmic Comparision between Square and Hexagonal Based grids. Graphical Models and Image Processing, Vol. 53, No.4, p. 324-339, 1991.