# A Frame Buffer Architecture for Parallel Vector Generation

Xiaolin Wu *
Department of Computer Science
University of Western Ontario
London, Ontario, Canada N6A 5B7

**Abstract** – An intelligent frame buffer architecture is proposed for parallel and distributed line scan-conversion. The architecture is of the type of wavefront array processors. It is demonstrated that the new frame buffer architecture achieves extremely high throughput and yet with very low frame buffer bandwidth requirement in scan-conversion, providing a solution to the bottleneck problem of pushing pixels into the frame buffer.

## 1 Introduction

Line is probably the most important computer graphics primitive. Research on plotting digital lines has remained active since the early days of computer graphics [2, 3, 7, 8, 9]. The author and his former professors at the University of Calgary developed very fast line scan-conversion algorithms that significantly outperform the classic Bresenham's algorithm. However, the impact of our achievements on interactive computer graphics is restricted by the fact the bottleneck in raster image generation is no longer in scan-conversion but in frame buffer access. In short, the modern image engines can compute images at a much higher speed than that of writing pixels into the frame buffer. Many hardware architectures were proposed to break the frame buffer bandwidth bottleneck [5, 4, 6]. This paper presents a new frame buffer architecture to facilitate parallel two-dimensional vector generation with the number of frame buffer accesses as few as the number of lines which presents a substantial reduction from the number of pixels.

The new frame architecture is a fundamental departure from the previous ones by sending only line end points into the frame buffer and letting pixels computed right inside the frame buffer. Furthermore, the new architecture allows distributed drawing of non-mutually-intersecting lines rather than just computing pixels of a single line in parallel, increasing the degree of parallelism from the previous pixel level to vector level. Despite its high performance, the new frame buffer architecture is conceptually and structurally simple, thus suitable for VLSI implementation. The algorithm for parallel line drawing is a variation of classic DDA line generator, and it is wired into wavefront array processors. Programming in the new frame buffer architecture is exactly the same as in today's standard frame buffer environment. Although the architecture is designed in this paper for very high throughput and low frame buffer bandwidth requirement in vector generation, it can be easily extended to support parallel and distributed processing for polygon scan-conversion, smooth shading and depth computations in Z-buffer algorithm.

The paper is organized as follows. First we derive in the next section a variation of the conventional DDA line generator, called integer DDA. The new DDA algorithm is particularly suitable to be implemented by wavefront array processors in parallel. In section 3 we design the parallel architecture of wavefront array DDA processors that can be integrated into a frame buffer. Then in section

*The author gratefully acknowledges the financial support of the Canadian Government through NSERC grant OGP0041926.

4 we demonstrate how the new frame buffer architecture can revolutionize line scan-conversion with high throughput but low frame buffer bandwidth requirement. Section 5 contains some discussions about the new architecture, its comparison with the previous designs, and its extension to support incremental graphics algorithms in general.

## 2 DDA and Integer DDA

The classic DDA (digital differential analyzer) is a simple and straightforward scheme for line scan-conversion. Without loss of generality we consider only lines of slopes between 0 and 1. Lines of other slopes can be accommodated by symmetry. In DDA we march along $x$ axis in unit step, i.e., $x_i = x_{i-1} + 1$, and increment $y$ by a floating value $k$ that is the slope of the line, i.e., $y_i = y_{i-1} + k$. The floating value $y_i$ is rounded to its closest integer, and the pixel $(x_i, \lfloor y_i + 0.5 \rfloor)$ is plotted in the $i$-th iteration. In this section we first introduce a simple but important modification of the original DDA to eliminate the floating arithmetic involved. This can be done by changing the floating addition to a module addition. The error introduced can be controlled by limiting the line lengths. The algorithm operates on an $n$-bit integer $D$. To draw a line from $(x0, y0)$ to $(x1, y1)$, $x0 < x1$, $y0 < y1$, $0 \leq k = (y1 - y0)/(x1 - x0) \leq 1$, we initialize $D = 2^{n-1}$, $x = x0$ and $y = y0$. The line is scan-converted from left to right. For every increment of $x$ the integer

$$d = \lfloor k(2^n - 1) + 0.5 \rfloor \qquad (1)$$

is added to $D$. The addition is carried out in module $2^n$ and the overflow of $D$ is recorded. Whenever an overflow occurs, $y$ gets incremented by 1.

To see the above logic we can imagine that the integer $D$ is a fractional number with the decimal point is to the left of its highest significant bit. Thus its initialization $D = 2^{n-1}$ is 0.5 in fraction. If we treat $D$ as the imagined fraction number, its increment $d$ is then an approximation of the line slope $k$, with the approximation error being

$$e = k - d2^{-n}. \qquad (2)$$

Clearly the error can be bounded by $|e| \leq 2^{-n}$. Since $D$ has an initial value of 0.5, when $D$ overflows at $x_i$, the line crosses the $x_i$ column above the midpoint between the two adjacent pixels $(x_i, \lfloor kx_i \rfloor)$ and $(x_i, \lceil kx_i \rceil)$, if we ignore the error term which is less than $i2^{-n}$. Thus $y$ should get an increment of 1 due to rounding. We call the above integer arithmetic for line drawing integer DDA to distinguish it from the traditional DDA. The integer DDA algorithm for lines of $0 \leq k \leq 1$ is described by the flowchart of Fig. 1.

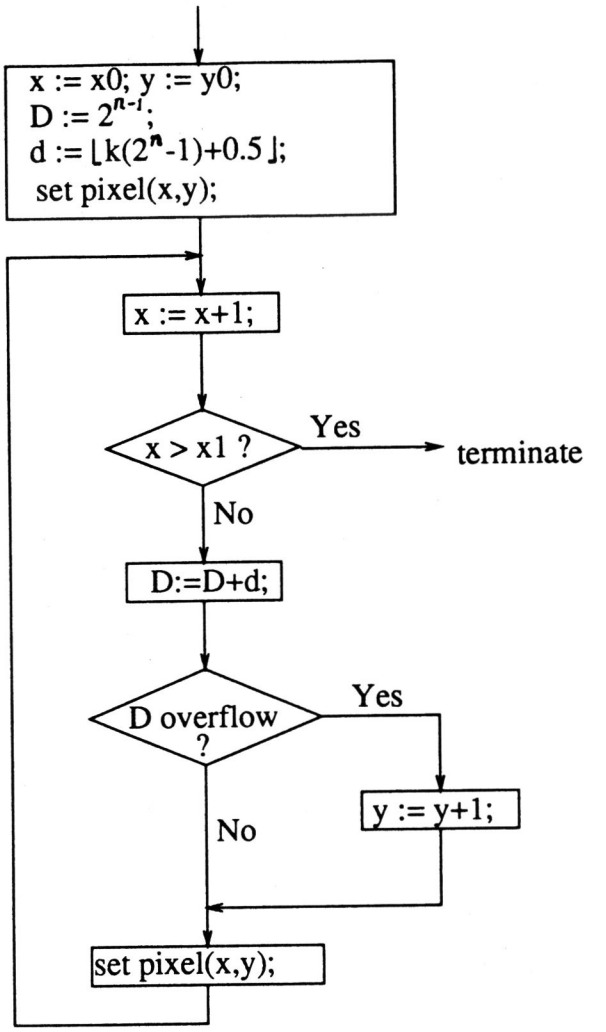


Figure 1: The integer DDA line generator.

# 3 Wavefront Array DDA Processors

The integer DDA derived earlier was not intended as a competitive alternative for incremental line algorithms, although it can be one. The real power of the integer DDA algorithm is that it is very suitable for massive parallel hardware realization. Indeed, in each iteration of the integer DDA algorithm the arithmetic and logic operations are extremely simple, namely, an addition and an overflow check; furthermore, those two operations are done on the same integer variable $D$. If we associate with each pixel in frame buffer with an adder and interconnect a pixel to its four 4-connected neighbors locally, then we can propagate the control integer variable $D$ from a pixel to its neighboring pixel on the course of a digital line based on the simple integer DDA algorithm. That is, we give each pixel some intelligence to determine whether it is on or off the digital line locally without being told by a central processor and hence eliminate frame buffer writing all together. Let us examine the array of integer adders for DDAs as described by Fig. 2. For the simplicity in explaining our design, only the connections necessary for drawing lines of $0 \leq k \leq 1$ are shown in the figure. Each cell in the proposed architecture performs very simple computations (see Fig. 2(a)). It takes integers $d$ and $D$ with its overflow bit $O$. If the overflow bit $O = 0$, the pixel at the current cell is set, the cell to the right is fired, and values $d$ and $D + d$ are sent to the right neighboring cell; in the case of $O = 1$, the cell to the top is fired, and values $d$ and $D$ with its overflow bit $O$ reset to 0 are sent to the upper neighboring cell. Clearly described above is an asynchronous architecture that may be classified as wavefront array processors. It is easy to see that the proposed architecture and the computations of each cell implement the integer DDA algorithm. To draw a line from $(x0, y0)$ to $(x1, y1)$, the cell for $(x0, y0)$ is fired first with $D = 2^{n-1}$ and $d = \lfloor k(2^n - 1) + 0.5 \rfloor$ as its input data. From this point on the cells on the digital line will be fired sequentially and set the pixels along the line. To terminate the line propagation at $(x1, y1)$ we need to set a flag at the cell at $(x1, y1)$ so that it will not fire any of its neighbors when it gets fired at the last.

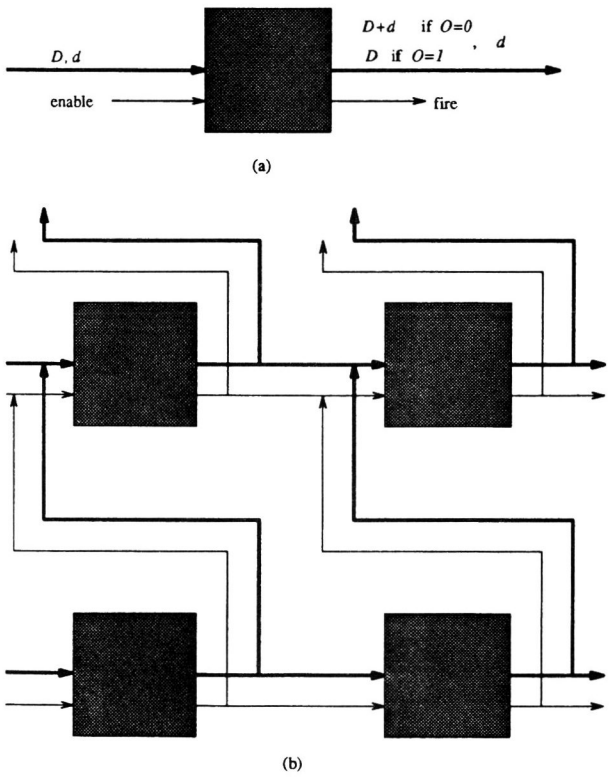


Figure 2: DDA cell (a) and DDA array (b) for computing lines of $0 \leq k \leq 1$.

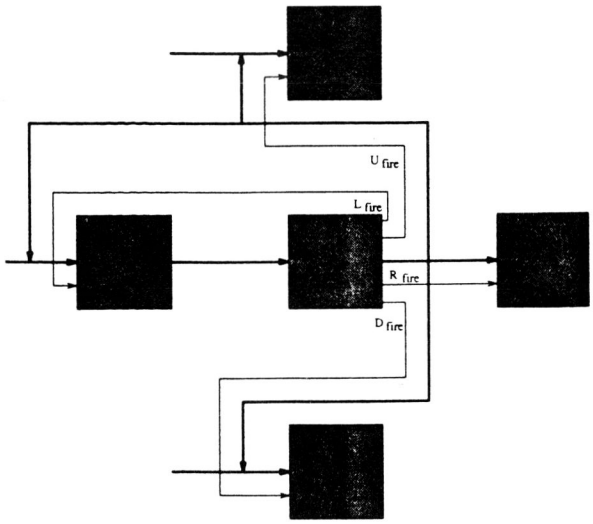


Figure 3: The connection of wavefront array processors for line scan-conversion.

The extension of the above design to include lines of all slopes is straightforward. In Fig. 3 illustrated is the connection of of wavefront array DDA processors for lines of all directions. Of course, we have to put more logic into each cell to accommodate lines of all orientations. To make the decision as to which of the 4-connected neighbors to be fired in the line propagation, we need to add to the data flow a two-bit direction code $C = c_1c_0$: $c_1c_0 = 00$ for $0 \leq k \leq 1$, $c_1c_0 = 01$ for $-1 \leq k < 0$, $c_1c_0 = 10$ for $1 < k < \infty$, and $c_1c_0 = 11$ for $-\infty < k < -1$. Let $R_{fire}$, $U_{fire}$, $L_{fire}$ and $D_{fire}$ be the signals to fire the right, upper, left, and lower cell. Then the logic of firing the next cell becomes:

$$\begin{aligned} R_{fire} &= (\overline{O} \wedge \overline{c_1}) \vee (O \wedge c_1\overline{c_0}) \\ U_{fire} &= (\overline{O} \wedge c_1) \vee (O \wedge \overline{c_1 c_0}) \\ L_{fire} &= O \wedge c_1 c_0 \\ D_{fire} &= O \wedge \overline{c_1} c_0 \end{aligned} \tag{3}$$

The reason for the 4-connection rather than 8-connection of the wavefront array DDA processors is to reduce the amount of inter-cell connections. As the result, a diagonal pixel move is accomplished by two axial moves. Our design has all the desirable features for easy VLSI implementation, namely, simple connection, modularity and regularity. Each cell and inter-cell connections are identical. The processor at each cell consists of an integer adder plus the logic circuit for Eq(3). Integrating the above wavefront array processors into the frame buffer, one processor per pixel, configures an intelligent frame buffer.

# 4 High-Speed Distributed Line Drawing

The proposed architecture has massive parallelism. The wavefront array DDA processors can compute all lines in a scene simultaneously as long as they do not intersect among each other. The time required is proportional to the length of the longest line, independent of the total number of pixels set. This level of parallelism makes it possible to compute wired-frame images in the time linear to the resolution of the display not to the complexity of the image.

The requirement of no two lines under scan-conversion intersecting each other is due to a resource contention problem. To guarantee the correctness of generated lines, no DDA cell may be fired by two sources at the same time. Since our design is asynchronous it is better to adopt the conservative prevention of disallowing line intersections. But this strategy will not restrict the parallelism of the architecture but rather shift the computational burdens to the central processor to solve the bottleneck problem in writing the frame buffer. The central processor will not compute the pixel screen coordinates and push the pixels into the frame buffer. Instead it will use the saved computing power to find the line intersections, and partition the lines into non-intersecting smaller line segments, and submit those shorter lines simultaneously to the wavefront array DDA processors integrated into the frame buffer. So the lines can be computed right inside the frame buffer in parallel, achieving extremely high throughput with very few frame buffer accesses from the central processor. The number of frame buffer accesses in the new architecture is the number of

line scan-conversion jobs submitted rather than the number of pixels in the scene.

# 5 Discussions

For the new intelligent frame buffer architecture clipping against the screen becomes unnecessary. We simply terminate the line propagation at the cells on the boundaries of the screen.

The type of incremental computations suitable for the proposed wavefront array architecture is common in computer graphics, for instances, the inner loops in polygon scan-conversion algorithm, in incremental depth computations of Z-buffer algorithm, and in smooth shading algorithms, are all of the form $D := D + d$ along the $x$ axis. Therefore, with some elaborations on the above architecture, it is possible for all these algorithms to be implemented in parallel and right in frame buffer. All the current mentioned algorithms that propagate on a line-by-line basis can then proceed along all scan lines simultaneously, with their running time bounded by the widest extent of the polygon (the longest scan line) not by the polygon area.

By the above argument the wavefront array architecture should also fit the bresenham's incremental line algorithm. Although this is true, Bresenham's algorithm uses two different increments in its inner loop rather than one in the integer DDA algorithm. Thus in the wavefront array architecture for Bresenham's algorithm we need one more constant in the data flow than the architecture for integer DDA algorithm. This increases the complexity of VLSI implementation and decreases the processing speed. Of course, the integer DDA may introduce accumulated error for very long lines while Bresenham's algorithm is error-free. But since the accumulated error can be bounded by the line length, the central processor may chop a long line into pieces before submitting it to the intelligent frame buffer to control the error and also to increase the throughput. So we do not think the error as a disadvantage of the parallel integer DDA algorithm.

Among all previous logic-enhanced frame buffer architectures, the one closest to our design is Fuches *et. al.*'s Pixel-Planes architecture [4]. They used a one-bit ALU for each pixel and organized those one-bit ALUs into a binary tree. The proposed frame buffer architecture requires a full rather than one-bit adder per pixel, hence more hardware extensive. However, the wavefront array design has higher modularity and simpler connection, hence more suitable for VLSI implementation. The degree of parallelism for the new architecture is higher by magnitudes than Fuchs *et. al.*'s design since the former computes objects in parallel whereas the latter computes pixels in parallel. Admittedly Fuches *et. al.*'s architecture can support a wider range of graphics and image processing operations with more elaborated parallel algorithms. More parallel graphics algorithms in wavefront array architecture are under our investigation. Wavefront array architecture has been successfully used in image processing [1]. It is the author's belief that more research is needed to explore wavefront array architecture's applications in computer graphics.

# 6 Conclusion

An intelligent frame buffer architecture in the form of wavefront array processors is proposed. The new architecture can perform parallel line scan-conversion at extremely high throghput and yet with very low frame buffer bandwidth requirement. This architecture is advocated to parallelize other incremental graphics algorithms.

# References

[1] E. R. Dougherty and C. R. Giardina, *Matrix Structured Image Processing*, Prentice-Hall, 1987.

[2] J. E. Bresenham, "Algorithm for Computer Control of Digital Plotter," *IBM Syst. J.*, vol. 4, no. 1, 1965, p. 25-30.

[3] J. E. Bresenham, "Run Length Slice Algorithms for Incremental Lines," in *Fundamental Algorithms for Computer Graphics* (R. A. Earnshaw ed.), p. 59-104, NATO ASI Series, Springer-Verlag, 1985.

[4] H. Fuchs, J. Goldfeather, J. Hultquist, S. Spach, J. Austin, F. Brooks, J. Eyles, and J. Poulton, "Fast Spheres, Shadows, Textures, Transparencies, and Image Enhancements in

Pixel-Planes," *Proc. SIGGRAPH'85*, 1985, p. 111-120.

[5] S. Gupta and R. F. Sproull, "A VLSI Architecture for Updating Raster-Scan Displays," *Proc. SIGGRAPH'81*, 1981, p. 71-78.

[6] M. Potmesil and E. M. Hoffert, "The Pixel Machine: A Parallel Image Computer," *SIGGRAPH'89*, 1989, p. 69-78.

[7] X. Wu and J. Rokne, "Double-Step Incremental Generation of Lines and Circles", *Computer Vision, Graphics, Image Proc.*, vol. 37, 1987, p. 331-344.

[8] B. Wyvill, "Symmetric Double Step Line Algorithm", in Graphics Gems edited by A. Glassner, Academic Press, 1990, p. 101-104.

[9] B. Wyvill, J. Rokne and X. Wu, "Fast Scan-Conversion of Lines," *ACM Trans. on Graphics*, vol. 9, no. 4, Oct. 1990, p. 376-388.