Anti-Aliased Line Drawing on a Distributed Cell Store System

A.A.Moore, C.M.Ng, D.W.Bustard

Department of Computering Science

University of Ulster

Cromore Road

Coleraine BT52 1SA

Northern Ireland

July, 1992

Abstract

One of the principle drawbacks with traditional parallel image composition architectures is the lack of support for transparent images. This paper introduces the Distributed Cell Store System, an architecture based on image composition principles, but which provides explicit support for transparency via its Serial Bus System. The transparency support is exploited in a scheme for the generation of smooth-edged lines, which avoids the need for any anti-aliasing calculation in software. The benefits of segmenting lines so that different segments may be rendered in parallel in different processing units are identified and quantified, and the paper concludes with a discussion on the benefits for incremental image specification systems which could be gained from implementation on such a hardware platform.

1 Introduction

Interactive graphics systems providing a quick response to user input clearly require high system performance. One way to attain such performance levels is to perform multiple operations concurrently. Multiprocessing architectures have been used for several years to meet the demanding computational requirements of interactive graphics systems. Simple operations such as the rendering process can be easily divided into a sequence of pipelined operations. If independent processors are used to handle these operations, very high performance can be attained [Clar82]. However, more complex operations such as surface tesselation and illumination calculations can not easily be handled by a highly pipelined architecture because of the difficulty in reconfiguring the pipeline to execute a wide variety of graphics operations efficiently.

Parallel processing architectures have been explored by many researchers for performing drawing operations [Fuch81] and for front-end geometric and arithmetic operations [Torb87]. However, a major drawback of highly parallel architectures is the low utilization of each processing unit in a fully instantiated processor-per-pixel [Eyle88] or processor-per-polygon [Deme80] system. In some cases the utilization of the processing units may be less than one percent of their capacity [Fole90]. One way to achieve higher utilization is to build only a fraction of the hardware, but to allocate its resources dynamically as they are needed. Two variants of this technique exist: virtual buffers [Ghar89] and virtual processors [Bunk89]. Since both systems visit a region only once in a frame time, bucket sorting is required, and this sorting must be performed on a frame-by-frame basis at video rates. The impact of the sorting process is to increase the latency of the system by one frame time which can be detrimental in real-time systems.

Another level of parallelism is available in virtual buffer systems by having several buffers work in parallel on different conceptual regions [Fuch89]. This notion of combining images from different regions lead to the formulation of another category of multilevel parallel system, the image-composition architecture [Moln88], [Shaw88]. The central idea is to distribute drawing primitives over a number of complete rendering systems. Each rendering system then processes its allocated primitives independently and stores the output in its own frame buffer. The output from the frame buffers is synchronised so that a tree of compositors can be used to combine the pixel streams of the partial image to form the final output. Very high performance systems can be built with this distributed processing technique by using a large number of parallel rendering systems. However, the image composition operation does not support transparent images, and aliasing can often be introduced into the final output.

The Distributed Cell Store system, being designed at the University of Ulster, is an extensible architecture based on the image composition principle which combines the functionality of a digital painting system with that of a video effects unit. The following sections describe this architecture and consider the possibilities for an increase in system throughput by making use of the hardware-implemented transparency function, and the distributed nature of the processing units.

2 The Distributed Cell Store System

The Distributed Cell Store (DCS) System (Fig. 1) is based on the image composition architecture. In order to overcome the aliasing and transparency deficiencies identified in the previous section, each image is assigned a display priority and each pixel is associated with an 8-bit transparency value. The Serial Bus System (image compositor) provides management of overlapping images, and mixes such images of similar display priority on a pixel-by-pixel basis according to the relationship defined by their transparency values. The number of processing centres (cell stores) in the system is determined by a trade-off between support for multiple overlapping images, and the resultant complexity of the image composition mechanism. For reasons of cost and efficiency, the current system uses four cell stores.

A cell store is a self contained rendering unit consisting of a graphics processing engine, a frame buffer, and a display processor to handle cell store output. The graphics engine in each cell store is a Texas Instruments TMS34020 Graphics System Processor (GSP) [Texa90a]. This is a powerful general-purpose CPU, providing instruction set support for graphics applications, a 512-byte instruction cache, and an on-chip memory controller which allows the GSP to interface

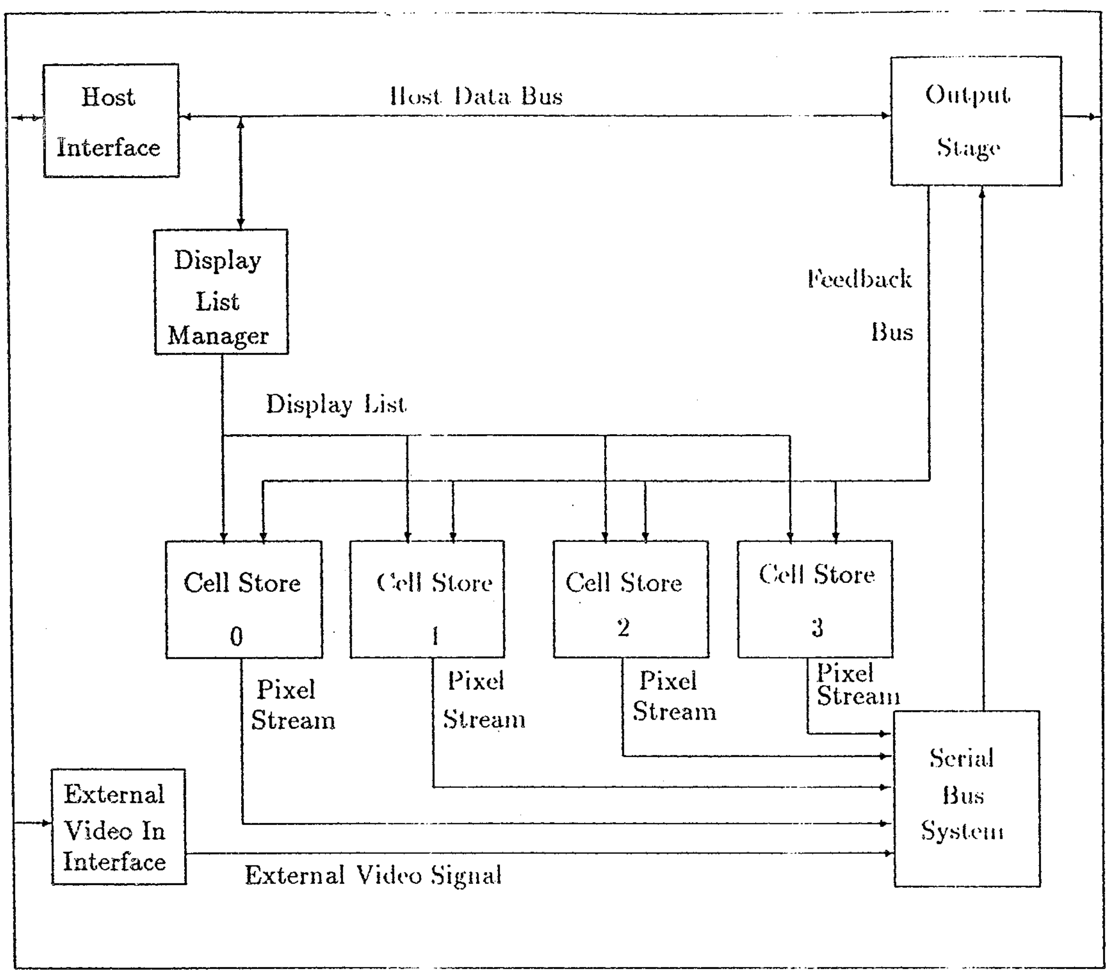

Figure 1: *Distributed Cell Store System: Function Block Diagram*

directly with the frame buffer, so completely removing the graphics burden from the host. Within a cell store, the GSP is associated with four TMS34082 floating point coprocessors [Texa90b], so increasing the system throughput for 3D and other floating point intensive applications. Moreover, the four coprocessors can be employed to improve cell store performance by processing the four pixel components (red, green, blue and transparency) in parallel.

An important capability of the newer, graphics-based user interfaces, originally developed at Xerox PARC [Thac81], is the use of windows to organise information on the display. The Distributed Cell Store System provides hardware support for window management. On screen, the window format is broken down into a series of horizontal strips or bands, each containing a number of rectangular areas, or tiles. A tile can be as large as the display itself, or as small as

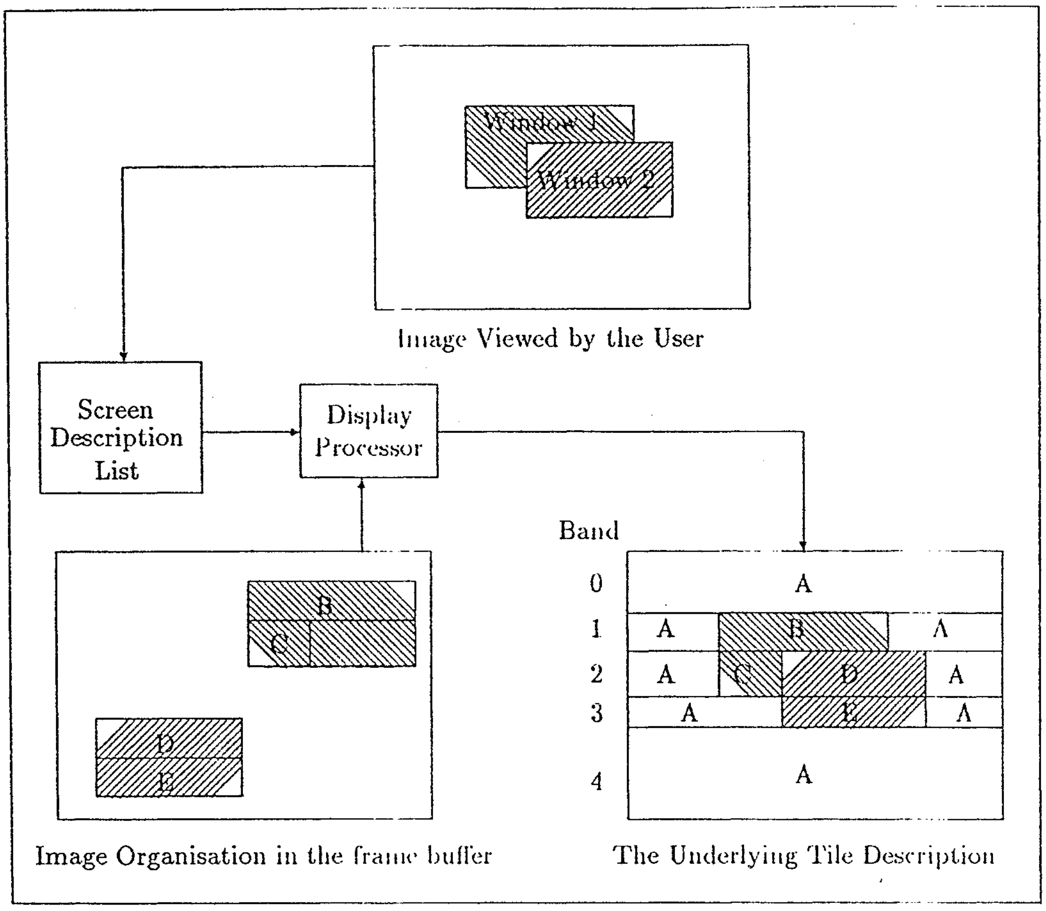

Figure 2: *Screen Description Mechanism for a Display with Two Overlapping Windows*

a single pixel. The images for each tile are read directly from the appropriate areas of the frame buffer on the fly during the active display time. A tile representation of a screen with two windows is shown in Fig. 2.

In Fig. 2, tiles labelled A represent the background area, and may be an arbitrary colour. Data need not be fetched from the frame buffer for these tiles, since the background colour pattern is stored in a register in the display processor. The areas of tiles B and C map the visible portion of window 1. The display processor fetches data for these tiles from the frame buffer region where the image for this window is located. Tiles D and E represent the visible portion of window 2. As an example, consider the composition of the first (uppermost) scan line in band 2. This band consists of 4 tiles; labelled A, C, D and A. The corresponding action for the display processor is to generate the background pattern for the area corresponding to tile A; then to read from the frame buffer, at the appropriate place, image data for tiles C and D; and finally to output the background pattern for the remainder of the scan line. If the user puts window 1 on top, the pointers specifying the width of tile C, the frame buffer starting address and the width of tile D will be changed, so instructing the display processor to fetch more data for tile C and less for tile D.

The tile description of the display screen is updated by the Display List Manager. This updated list is then downloaded to the Display List Processor of the corresponding cell store. As the tiles are defined by a set of pointers, operations such as panning and scrolling images within a window, or even resizing and moving the window about the screen can be performed simply by manipulating these pointers. Since the images are read directly from the frame buffer, the changes are always presented to the user in a single frame time.

The Serial Bus System serves three main purposes: The first concerns the provision for output of images at video rate in response to the display processor. This involves the resolution of overlapping images, and combination of images of equal priority according to the mixing relationship defined by their transparency values. Secondly, the Serial Bus System provides a method for the capture of an external video signal. This is a function of the Serial Bus System rather than the data bus, as the latter cannot meet the required video rate and cannot be occupied for long periods of time without affecting the memory and screen refresh functions of the GSP. Thirdly, the Serial Bus System controls the merging of a captured live image with that stored in the frame buffer.

The display priority of an image defines its conceptual distance from the viewer. The opacity of an image is defined by an 8-bit value (transparency factor) at each pixel. If the image of highest priority is non-opaque (transparency factor < 255) at any pixel position, it will be mixed with the image of next highest priority. This depth traversal continues until either a fully opaque pixel or the background is reached. A hole in an image is specified by pixel transparency factors of zero; such pixels will contribute nothing to the displayed image.

The output stage of the system consists of an output buffer and a transformation unit. Final system output can be redirected to a buffer either preceding or following the transformation engine.

The image stored in this buffer can be regarded as the current output as frozen, thus enabling a mix between an old frame and the new output. This is a cost-effective alternative to using a second screen output system.

The transformation engine consists of a GSP and a multiplier-adder tree. The transformation of an image from a source space to a target space using a 4-by-4 transformation control grid [Nibl79] requires 32 multiplications and 30 additions for each pixel. Such a throughput is impossible in a 70nS pixel time, so the transformation engine takes advantage of the fact that pixels of the same scan line have similar vertical coefficients and control parameters. These values can then be calculated during the horizontal blanking period prior to the processing of that scan line. Horizontal coefficients are accessed when needed from an SRAM store, indexed by a counter (Horizontal Offset Generator). The control parameters, and vertical and horizontal coefficients are then passed to the multiplier-adder tree, from which the address in source-space of a pixel can be output in a single 70nS pixel time.

3 Serial Bus Anti-Aliasing of Lines

One of the most widely used benchmarks against which the performance of computer graphics systems is measured is the rate at which straight lines can be generated. Many architectures provide hardware support for line generation and the TMS34020 GSP at the heart of each cell store even provides explicit instruction set support for this operation. The quality of the lines produced by such hardware supported techniques, however, is usually impaired by aliasing. One of the most common and effective methods for elimination of aliasing is brush extrusion - a technique first suggested by Whitted [Whit83] from an idea espoused by Crow [Crow78].

By brush extrusion, a matrix of intensities greatest at the centre and decreasing towards the edges, (representing a brush) is conceptually dragged along the path of the line. The line is rendered by stamping the brush on the raster at each position produced by the line drawing algorithm. The stamp is performed on a pixel-by-pixel basis and comprises a weighted mix of the drawing colour and the existing pixel value at that point. Colour systems which specify pixel values in terms of quantities of red, green and blue require three such mix operations for each pixel. The Serial Bus System of the DCS provides hardware support for non-opaque images, and, by employing some of the capabilities of the GSP at the heart of each cell store, a brush extrusion technique which avoids time-consuming pixel mixing in software can be implemented.

The anti-aliasing scheme takes advantage of the fact that the Serial Bus System performs pixel mixing in hardware at the image composition stage. By this technique, the line is drawn only on the transparency plane of the frame buffer, thus specifying the degree of opacity for each pixel of the image represented on the colour planes. For a line of constant colour, the red, green and blue planes need only contain a block of that colour which bounds the path of the line. This block may be created by the GSP FILL instruction which can set the values of up to 40 million 32-bit pixels per second. The stamping of the brush along the path of the line can also take advantage of a hardware-supported GSP instruction.

The PIXBLT (pixel-block transfer) instruction implements a general bitBlt [Pike84] operation, of which the transfer rate can approach 4 million 32-bit pixels per second. The GSP also provides pixel processing operations which can be used to specify the exact manner in which the PIXBLT instruction affects the destination raster. These range from a replacement of the destination array with the source, to a number of boolean and arithmetic functions such as AND, OR, XOR, ADD, SUB, MAX, and MIN. Anti-aliasing applications are supported by the ADDS (Add with Saturate) option, by which the pixel values in the PIXBLT source array are added to those of the destination region, but where each pixel sum is bounded to the maximum displayable value (Fig. 3).

Consider a line from (x_0, y_0) to (x_1, y_1) (where $x_0 < x_1$ and $y_0 < y_1$) created using a brush of diameter d. The path of the line is traced by repeatedly PIXBLTing the brush array to the transparency plane, with one PIXBLT centred on each point generated by the line drawing algorithm. The ADDS option will ensure that those pixels affected most often by PIXBLT's will exhibit the highest intensity, while the intensity of those pixels on the extremities of the line will be lower. Since the first and last PIXBLT's will be centred on the start- and endpoints of

	a	b	c	d	e
1	22	50	80	50	22
2	50	115	170	115	50
3	80	170	255	170	80
4	50	115	170	115	50
5	22	50	80	50	22

(a) Brush to be stamped

Values represent the 8-bit intensity level of the brush at that pixel

	a	b	c	d	e
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

(b) Traget Raster

Zero values indicate a virgin (black) background before the first stamp operation

	a	b	c	d	e	f	g
1	22	50	80	50	22	0	0
2	50	115	170	115	50	0	0
3	80	170	255	170	80	0	0
4	50	115	170	115	50	0	0
5	22	50	80	50	22	0	0
6	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0

(c) The effect of the 1st brush Stamp

This diagram shows the effect of the first stamp, centered on pixel (3,c), where the maximum intensity is 255. The shaded area will be affected by the next stamp.

	a	b	c	d	e	f	g
1	22	50	80	50	22	0	0
2	50	137	220	195	100	22	0
3	80	220	255	255	195	50	0
4	50	195	255	255	220	80	0
5	22	100	195	220	137	50	0
6	0	22	50	80	50	22	0
7	0	0	0	0	0	0	0

(d) Result of the 2nd brush stamp

This diagram shows the effect of the second stamp, centered on pixel (4,d). Note how the ADDS function has resulted in some pixels reaching (but no exceeding) the maximum value, while those further from the centre are at lesser intensity.

Figure 3: *The ADDS Function with Overlapping Brush Stamps*

the line, respectively; the smallest rectangle which bounds the area affected can be described by $(x_0 - d/2, y_0 - d/2)$ and $(x_1 + d/2, y_1 + d/2)$. These coordinates specify the window from which the line is displayed.

The position of the line on the screen is passed to the Display List Manager which adjusts the tile description of the output image to map the window containing the line to the appropriate place on the screen. During image composition, the Serial Bus System resolves the transparency for the window, so allowing only those pixels which have been affected by the PIXBLT's (those which play a part in the line) to contribute to the final image. The amount contributed by these pixels is dependent on their transparency value, and as can be seen from Fig. 3 the resultant pattern will be that of a line segment with intensity greatest at the centre and tending to zero at the endpoints and edges.

4 A Parallel Line Drawing Technique using the DCS

The example of the previous section illustrates how the hardware support for transparency inherent to the Serial Bus System can be used to implement an efficient anti-aliasing mechanism. However, such a scheme requires only a single cell store, and makes scant use of the hardware windowing facility. We now present a line drawing methodology which employs the full functionality of the Distributed Cell Store System to greatly improve the throughput of an image creation system.

Consider a 4-cell store DCS architecture as described in Fig. 1. Provision of an image creation environment would require that one of the cell stores be dedicated to maintaining the state of the current image so far. Therefore, at most 3 fully functional cell stores remain for rendering of future primitives. If an anti-aliased line is to be added to the image, the operating system can determine exactly how many cell stores are available and then equally subdivide the line, distributing one segment to each available cell store for rendering in parallel. The line segments can be rendered on any available area of the frame buffer in a cell store, since the Display List Manager keeps track of the frame buffer locations of all entities and their corresponding screen coordinates.

Cell Store	X Coordinate	Y Coordinate
0	$(x_0 - d/2)$	$(y_0 - d/2)$
1	$(x_0 - d/2) + (dx/3)$	$(y_0 - d/2) + (dy/3)$
2	$(x_0 - d/2) + (2 * dx/3)$	$(y_0 - d/2) + (2 * dy/3)$

Table 1: *The display origin of the line segments*

When all line segments have been rendered, the Display List Manager distributes the updated display lists to the cell stores. For each cell store, the display list specifies the frame buffer origin, width, and height of the window containing that cell store's line segment, and the display origin from which that window will be output. The display lists are processed by their respective display processors, which fetch the pixels from the frame buffers and output them to the Serial Bus System at the appropriate time.

Suppose a line from (x_0, y_0) to (x_1, y_1) created with a regular brush of diameter d is to be added to an image in cell store 3. If all other three cell stores are available, the line can be equally subdivided into three segments. The rectangular area required to bind a segment has a dimension of dx by dy, where $dx = (x_1 - x_0 + d)/3$ and $dy = (y_1 - y_0 + d)/3$. The display origin to which each cell store's segment window is mapped is then given in table 1.

The display lists are downloaded to the display processor during the field blanking period after the entire line rendering process has been completed. By mapping the individual windows containing the line segments to the origins given, the complete line can be viewed in the next displayed frame. As the display image output from the Serial Bus System can be routed through an output stage frame buffer, the composite image can then be directed, via the Feedback Bus, to cell store 3 from where it can be used as the background for the next primitive to be added. Such an incremental approach to image construction has applications in a wide range of areas from painting toolkits to CAD systems. Fig. 4 illustrates this distributed approach to the rendering of primitives.

One potential problem in this scheme concerns the treatment of the points at which the line is subdivided. The overlapping nature of the brush stamps comprising a line results in pixels being potentially affected by a sequence of PIXBLT instructions. If, however, the line is segmented,

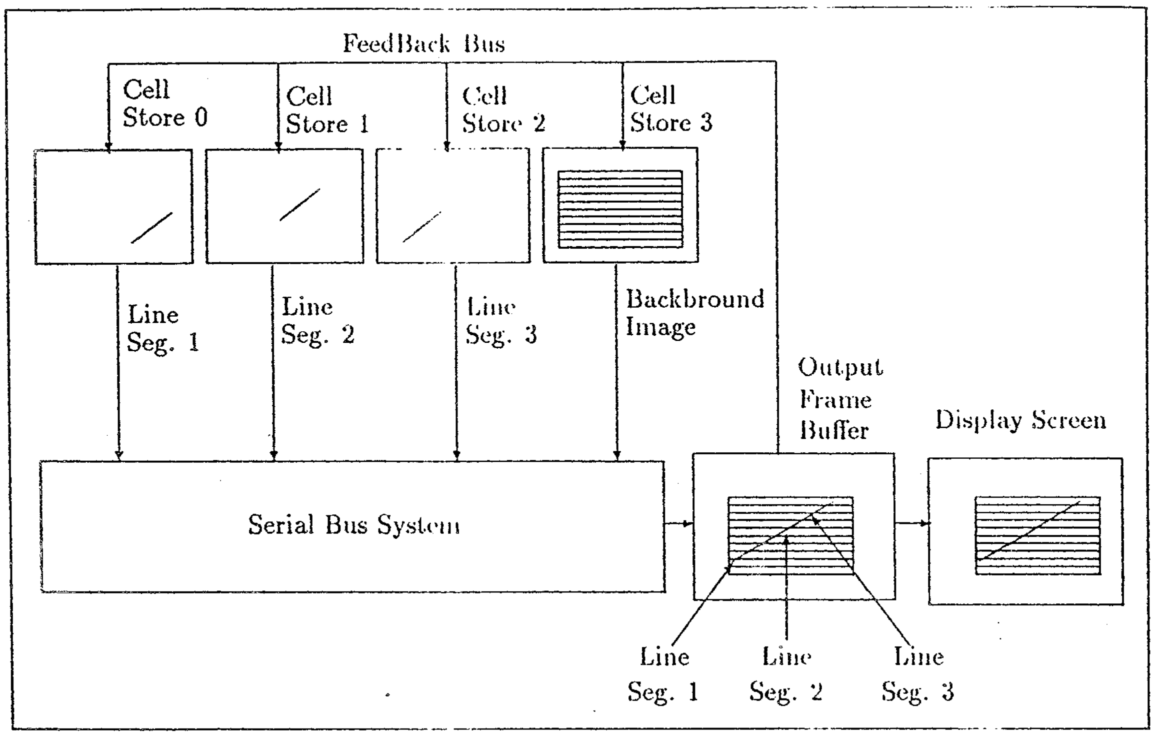

Figure 4: *Distributed Line Rendering. Cell Stores 0, 1 & 2 each render one third of the line which the serial Bus System displays as a whole on the background supplied by Cell Store 3. The resultant image is then captured by the Output Frame Buffer and routed to Cell Store 3, where it becomes the background onto which the next primitive is added*

then stamps immediately before and after the point segmentation will not influence all the pixels that would have been affected had segmentation not taken place - since different segments reside in different cell stores. The solution is to extend the segment in each cell store past the point of subdivision, and clip the segment to the original subdivision when passing the window dimensions to the Display List Manager. In this way, each pixel in each segment is visited by the brush exactly as it would in a non-distributed implementation, yet the line appears as if it were rendered in a single cell store. If the brush is stamped once for each horizontal or vertical pixel along the extent of the line, the required extension in a cell store holding a segment from the middle of the line is equivalent to the diameter of the brush - half of which is at either end of the segment. It is therefore important to ensure before subdivision that the overhead incurred by the segment extension does not outweigh the advantages of segmenting and distributing the line.

Assume that the length of the original line is L, the diameter of the brush is d, and n cell stores are used to process the line in parallel. With n line segments, there are $(2n - 2)$ extensions, and the total extra length due to these extensions is $(n - 1) * d$. If the line is equally divided, each cell store will need to process a line segment of average length $[L + (n - 1) * d]/n$. In order to achieve any speed advantage from the distributed approach, the length of the line segment rendered by each cell store must be less than the original length of the line, which could after all be rendered by a single cell store without any extensions. i.e.

$$[L + (n - 1) * d]/n < l \tag{1}$$

rearranging,

$$\begin{aligned}
[L + (n - 1) * d]/n < L &= [L + (n - 1) * d] < L * n && (2)\\
&= (n - 1) * d < L * n - L && (3)\\
&= (n - 1) * d < (n - 1) * L && (4)\\
&= d < L && (5)
\end{aligned}$$

This result indicates that for any performance improvement to be gained from distribution, the length of the line to be rendered must be greater than the diameter of the rendering brush. If this is not the case, then the line will be rendered more quickly by a single cell store.

It is interesting to note the effect on the performance of the distributed scheme of the ratio of the length of the line to the diameter of the brush. Assuming that the time required to render a line segment is proportional to the length of the segment, then the ratio of the time required to process the whole original line by a single rendering unit to the time required for each individual cell store to process an equally divided line segment is:

$$L : [L + (n - 1) * d]/n \tag{6}$$

Expressing the ratio of line length to brush diameter as $M = L : d$ and substituting M into the

		Ratio of Line Length to Brush Diameter										
		1	2	4	8	16	32	40	64	80	100	120
Number of rendering cell stores	2	1.00	1.33	1.60	1.78	1.88	1.94	1.95	1.97	1.98	1.99	1.99
	3	1.00	1.50	2.00	2.40	2.67	2.82	2.86	2.93	2.96	2.98	2.99
	4	1.00	1.60	2.29	2.91	3.37	3.66	3.72	3.86	3.93	3.96	3.98

Table 2: *Performance Improvement Obtained by Distribution of Line Drawing*

ratio above gives:

$$Md \ : \ [Md + (n-1) * d]/n \tag{7}$$

$$= Mnd \ : \ Md + nd - d \tag{8}$$

$$= Mn \ : \ M + n - 1 \tag{9}$$

Table 2 illustrates how this ratio is affected by the line length to brush diameter ratio and by the number of rendering cell stores. As the length of the line greatly exceeds the brush diameter, so the overhead incurred by segment extension becomes negligible, and the performance improvement offered by distribution becomes a direct function of the number of rendering cell stores.

5 Performance Evaluation

Table 3 provides simulated results (in mS) for the rendering of a range of lines of different lengths, with rendering carried out by a varying number of cell stores. The rendering time comprises:

1. the time required for the host to determine the number of available cell stores and subdivide the line accordingly;
2. the time for commands to be issued sequentially by the host to each participating cell store; and
3. the time for the cell stores involved to perform the area fill, point generation, and brush stamping functions required for each line segment.

	Length of the Line (in pixels)							
	4	8	20	40	80	160	320	640
No Distribution	0.319	0.634	1.654	3.606	8.474	22.050	64.562	211.026
1 Cell Store	0.335	0.650	1.670	3.623	8.490	22.067	64.579	211.042
2 Cell Stores	0.495	0.574	1.064	1.945	3.947	8.911	22.679	65.575
3 Cell Stores	0.499	0.578	0.902	1.499	2.712	5.063	13.152	35.119
4 Cell Stores	0.503	0.582	0.824	1.242	2.139	4.173	9.201	23.097

Table 3: *Simulated Execution Time of Line Drawing (in mS)*

The precise time for the area fill operation is dependent on the orientation of the line and hence the size of its bounding rectangle. The results in Table 3 reflect the worst case when the line lies on the true diagonal and the size of the bounding box is at its greatest. The brush stamping function assumes a square brush matrix of dimension 7 pixels.

For the purposes of comparison, the top line of results indicates the performance of this technique where no attempt is made to distribute rendering and the lines are generated in entirety by a single cell store. The difference between the results in this case and those where an attempt is made to distribute but only a single cell store is available may be expressed as the overhead incurred during distribution. Where distribution is available, it can be seen that as the line length grows, so the speedup gained by distribution increases dramatically, up to a 90% reduction in the rendering time for a 640 pixel line when produced by 4 cell stores.

6 Conclusions

The efficient generation of high-quality images can be greatly enhanced by hardware support for anti-aliasing, but such support is not usually provided by image composition architectures. This paper has introduced the Distributed Cell Store System - an architecture based on the image composition principle, but which implements a pixel-level transparency function in hardware via its Serial Bus System. The provision for windowing by dynamic mapping of regions of distributed frame buffers to the display screen, lends itself to the specification of images by their component parts - each rendered in parallel by an independent processing unit. The DCS feedback bus can be used to route the composite image back to one of the independent units in a single frame time.

Experiments suggest that a substantial improvement in system throughput can be obtained by employing such hardware in an environment of incremental image specification. A specific scheme for distributed rendering of anti-aliased lines has been presented, but the basic approach could easily be extended to other primitives.

References

[Bunk89] Bunker, M, and Economy, R., "Evolution of GE CIG Systems", *SCSD Document*, General Electric Company, Daytona Beach, FL, 1989

[Clar82] Clark, J., "The Geometry Engine: A VLSI Geometry System for Graphics", *Computer Graphics*, Vol.16, No.3, 1982, pp127-133

[Crow78] Crow, F., "The use of Grayscale for Improved Raster Display of Vectors and Characters", *Computer Graphics*, Vol.12, No.3, August 1978, pp1-5

[Deme80] Demetrescu, S., "A VLSI-Based Real-Time Hidden-Surface Elimination Display System", *Master Thesis*, Department of Computer Science, California Institute of Technology, Pasadena, CA, May 1980

[Eyle88] Eyles, J., Austin, J., Fuchs, H., Greer, T., and Poulton, J., "Pixel-Planes 4: A Summary", in *Advances in Computer Graphics Hardware II* (1987 Eurographics Workshop on Graphics Hardware), Eurographics Seminars, 1988, pp183-208

[Fole90] Foley, VanDam, Feiner, Hughes, *Computer Graphics: Principles and Practice*, second edition, Addison Wesley

[Fuch81] Fuchs, H. and Poulton, J., "Pixel-Planes: A VLSI-Oriented Design for a Raster Graphics Engine", *VLSI Design*, 2(3), Q3 1981, pp20-28

[Fuch89] Fuchs, H. Poulton, J, Eyles, J., Greer, T., Goldfeather, J., Ellsworth, D., Molnar, S., Turk, G., Tebbs, B., and Israel, L., "Pixel-Planes 5: A Heterogeneous Multiprocessor

Graphics System Using Processor-Enhanced Memories", *Computer Graphics*, Vol.23, No.3, 1989, pp79-88

[Ghar89] Gharachorloo, N., Gupta, S., Sproull, R., Sutherland, I., "A Characterization of Ten Rasterization Techniques", *Computer Graphics*, Vol.23, No.3, 1989, pp355-368

[Moln88] Molnar, S., "Combining Z-Buffer Engines for Higher-Speed Rendering", 1988 Eurographics Workshop on Graphics Hardware, Sophia-Antipolis, France, September, 1988, in Kuijk, A.A.M., ed., *Advances in Computer Graphics Hardware III Proceedings* of 1988 Eurographics Workshop on Graphics Hardware, Eurographics Seminars, Springer-Verlag, Berlin, 1989

[Nibl79] Niblack, W., *An Introduction to Digital Image Processing*, Prentice-Hall International, 1979, pp140-141

[Pike84] Pike, R., "Bitmap Graphics", in *Course Notes 4 for SIGGRAPH 84*, Minneapolis, MN, July 1984

[Shaw88] Shaw, C., Green, M., and Schaeffer, J., "A VLSI Architecture for Image Composition", 1988 Eurographics Workshop on Graphics Hardware, Sophia-Antipolis, France, September, 1988, in Kuijk, A.A.M., ed., in *Advances in Computer Graphics Hardware III Proceedings* of 1988 Eurographics Workshop on Graphics Hardware, Eurographics Seminars, Springer-Verlag, Berlin, 1989

[Texa90a] Texas Instruments Inc., *TMS34020 User's Guide*, Texas Instruments, Dallas, TX, 1990

[Texa90b] Texas Instruments Inc., *TMS34082 User's Guide*, Texas Instruments, Dallas, TX, 1990

[Thac81] Thacker, E., McCreight, E., Lampson, B., Sproull, R.,and Boggs, R., "Alto: A Personal Computer", in Siewiorek, O, Bell, G., andNewwell, A, *Computer Structure: Readings and Example*, McGraw-Hill, NewYork, 2nd edition, 1981

[Torb87] Torborg, J., "A Parallel Processor Architecture for Graphics Arithmetic Operation", *Computer Graphics*, Vol.21, No.3, July 1987, pp197-204

[Whit83] Whitted, T., "Anti-Aliased Line Drawing using Brush Extrusion", *Computer Graphics*, Vol.17, No.3, July 1983, pp151-156