Distributed Frame Buffer for Rapid Dynamic Changes to 3D Scenes

Derek Coppen, Mel Slater, Allan Davison, David Hawes

ABSTRACT
This paper describes a distributed frame buffer architecture, based on the Tiling Algorithm for dynamic modification, and designed to achieve fast display updates in response to dynamic transformations of graphical objects. We report on the overall architecture and some detailed design issues.

1.1 Introduction

Hardware for computer graphics usually focusses exclusively on the problem of how to render as many polygons as possible per second, given the constraints of a particular rendering model. When dynamic changes are made to a displayed scene, the strategy usually adopted is to re-render the entire changed scene fast enough to give real-time frame rates. This paper describes an alternative approach, designing and implementing a graphics architecture which from the outset is designed to solve the dynamic changes problem, and attempts to minimise the amount of rendering caused by dynamic changes, while still maintaining a fast overall polygon rendering rate. Our alternative strategy employs a distributed frame buffer. This paper reports on work undertaken as a result of the SERC funded project, *Graphics Object Management with a Distributed Frame Buffer* (1989–91), reported in [2].

The distributed frame buffer is based on the tiling strategy, discussed in [3, 5, 4]. In this method the display space is conceptually divided into a rectangular grid, where each grid element is called a tile. Each tile references a set of identifiers of primitives (for example, polygons) that pass through it. Initially each tile references an empty set. When an object is added to the scene, the identifiers of its corresponding polygons are added to the sets of the tiles which they cover. The collection of tiles intersected by the object is called its *tile set.*

For hidden surface elimination, it is assumed that there is a Z-buffer. When an object is added to the scene the Z-buffer is updated in the usual way. When a target object is to be removed from the display all Z-buffer values in its tile set are overwritten by the maximum possible Z value. The target's identifiers are removed from its tile set. The set of all identifiers of polygons stored in its tile set is called the Active Primitive (or Polygon) Set (APS). The primitives in the APS belong to those objects that are likely to share display space with the target. Therefore all primitives in the APS are re-rendered, but with 2D clipping to the tile set of the target. This clipping ensures that there is no knock-on effect outside the range of the *damaged area,* and also, of course, minimises the amount of redrawing necessary.

In the next section we discuss the basic ideas of the system. Section 1.3 gives the hardware system overview. Section 1.4 discusses rendering, and Section 1.5 the current status.

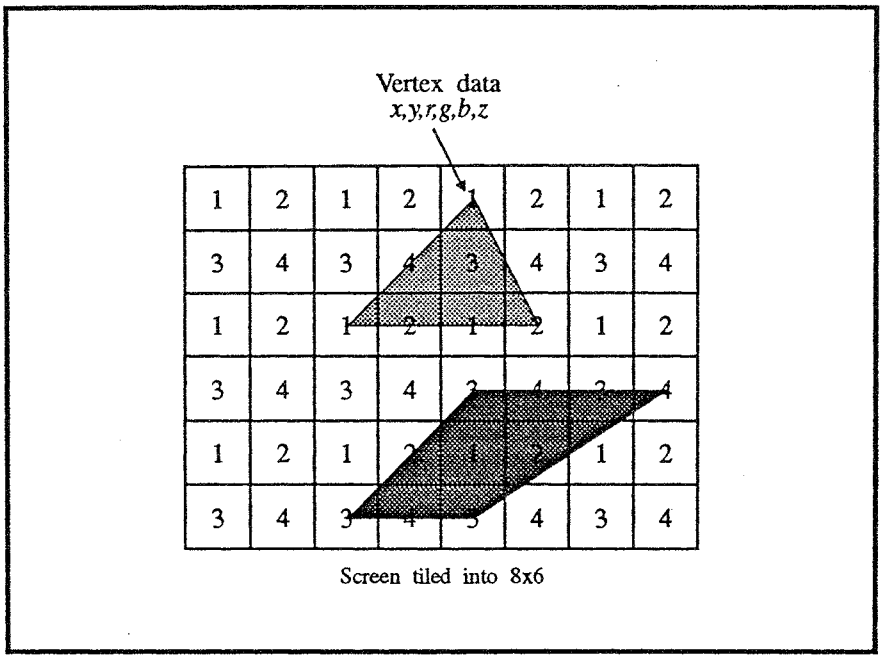

FIGURE 1.1. Tiled Polygons

1.2 Basic Ideas

The new work makes use of a *distributed frame buffer*, that is, where there are a number of processors, each assigned a number of tiles. The tiling algorithm used in this parallel system differs somewhat from that outlined in the previous section, in order to make effective use of the processors in the distributed frame buffer and to eliminate the need for extensive communication between them. Previously damage repair was achieved by taking all the tiles in the damaged area of the screen, combining all the lists of polygons in the corresponding tiles to produce the APS of polygons which are then rendered after clipping to the tile set. If this method were used on the parallel system the processors would need to communicate with each other to access the tile and object lists. An alternative method was therefore developed which does not require this coordination between processors during rendering.

Each polygon which covers more than one tile is clipped to the boundaries of those tiles to produce a set of fragments which are the parts of the polygon covering each individual tile. These fragments are produced by a host interface processor and then distributed to the processors controlling the tiles to which they correspond where they are stored for later use. When damage repair is required, as a result of deleting an object from the display, the processors controlling the tiles which the object covers are instructed to remove the fragments representing that object and redraw all the other fragments in those tiles. In this way the processors in the distributed frame buffer array can each render the scene into their own sections of the frame buffer independently of one another. Figure 1.1 shows an arbitrarily tiled screen with the number of renderers set at 4. The numbers indicate the tile to renderer assignment.

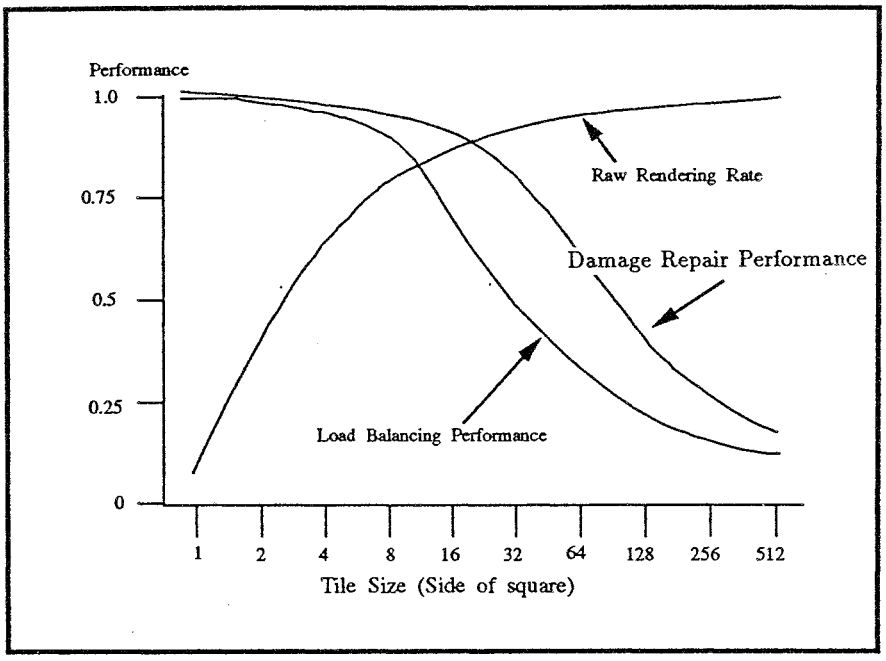

FIGURE 1.2. Tile Factors

Great importance was placed on overall rendering efficiency and final rendered image quality. These requirements were satisfied by the following:

- Tile size
- Shading
- Tile rendering
- Anti-aliasing
- Screen double buffering

1.2.1 Tile Size

Rendering simulations were undertaken on various animated scenes with different tile sizes in an attempt to determine an *optimal* choice for tile size. Of course, any optimal tile size is only actually valid for one particular scene! Figure 1.2 illustrates the three major contributors to tile size determination.

To render any object not only do we have to transform it to the screen co-ordinates, but we also have to clip it to the chosen tile size. Very small tiles are inefficient due to the high cost of generating, clipping, and rendering many small polygon fragments. With small tile sizes the effect is perceived as a reduction in raw rendering rate. In this case large tile sizes are to be preferred.

Where polygon shape can be closely approximated by the tiling operation, the amount of damage repair is reduced. Large tiles are inefficient as larger screen areas than necessary have to be redrawn and so in this case, small tiles are indicated.

For a multi-renderer the choice of tile size affects load balancing. The load balance between processors is lower for large tiles as most polygons will overlap a smaller number of tiles. As in the previous case, small tiles are desirable.

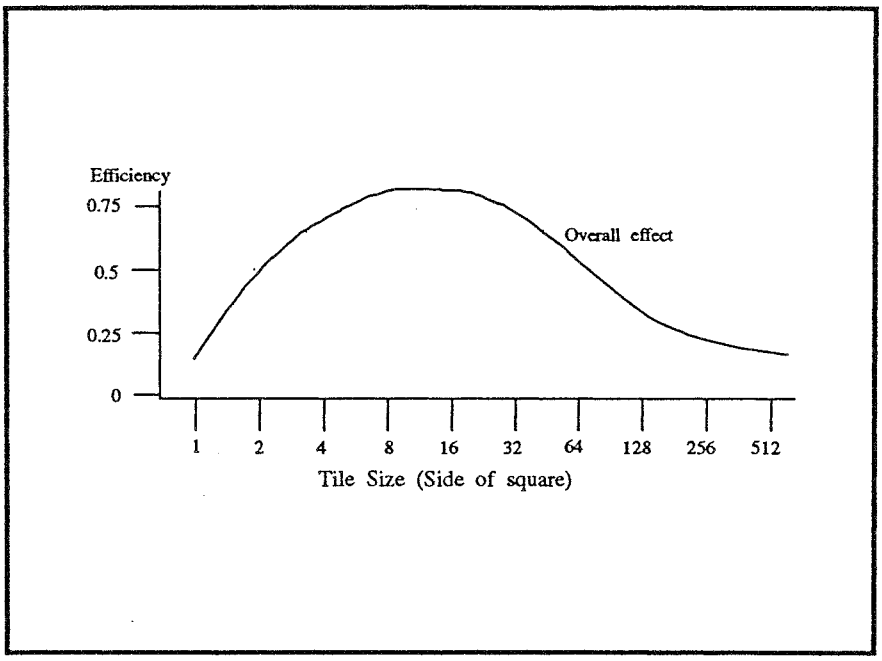

FIGURE 1.3. Tile Rendering Efficiency

Taken together, the above three lead to the concept of tile efficiency, depicted in Figure 1.3. The effect of different hardware implementations (which could cause a skew in the graph) is not included in the diagram.

1.2.2 Shading

Gouraud shading was the minimum level of colour approximation acceptable. Phong shading would have been too difficult to implement within the time and cost constraints of the project, however some thought was to be given to using a second order colour interpolation technique.

1.2.3 Tile Rendering

The clipped polygons are rendered as trapezia with top and bottom edges horizontal. Some scan conversion techniques, as in [6], first iterate along the edges containing the partially covered pixels of a polygon and then subsequently convert the central, fully covered, part. This approach was rejected as the edge iteration can be viewed as a large collection of short lines which can be inefficient with some hardware. The trapezoid rendering in this project was accomplished with a single pass with scan conversion being done left to right.To simplify the stepping from line to line, scan conversion always starts with the leftmost vertex so that some trapezia are scanned top to bottom and others, bottom to top. No pixel count is kept for the length of each line; line scan conversion terminates either when the tile boundary is encountered or when the horizontal scanning intersects with an iteration trajectory of the right hand edge.

1.2.4 Anti-aliasing

An anti-aliased image was required so the opportunity was taken to extend the rendering hardware to support this. Simulations of the following approach have shown good effect. This is discussed fully in [1].

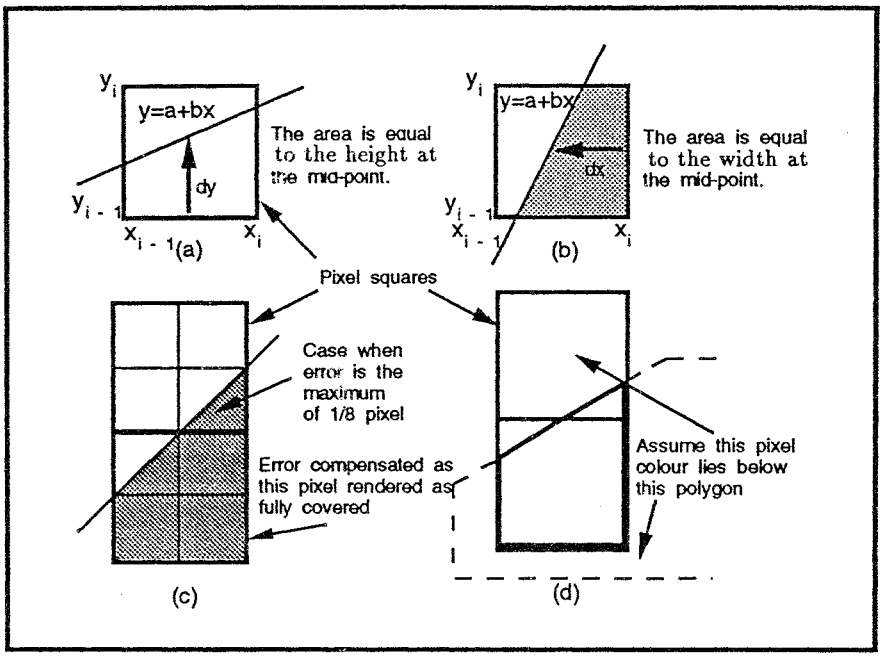

FIGURE 1.4. Anti-aliasing Techniques

Each pixel data, which contains colour and z information, was extended to include a tag field. Where the pixel is fully covered the tag value is set to zero. For partially covered pixels, the x and y error terms, which would normally be used to compute the area coverage, are saved in the tag field along with the edge type, either left, right, top or bottom. Essentially, the anti-aliasing is being deferred until later; in this case, when the whole tile has been rendered. Anti-aliasing can then be achieved with a single pass over the rendered image. Pixels with a zero tag value are left unaltered, whilst the underlying colour of partially covered pixels is inferred by looking at the colour of one of the orthogonally adjacent pixels, as directed by the tag edge entry. Colour mixing is then controlled by the x and y error terms as for the usual pre-filtering, anti-aliasing approach. Figure 1.4 illustrates this.

There are two further extensions to this technique. First, where the pixel is less than 50 percent covered, only the tag data is updated, not the colour information, though this variation is noted by an extra tag bit. Second, it is possible to flag, say, both left and right edges passing through a single pixel so that lines of sub-pixel width may be rendered.

1.2.5 Screen Double Buffering

To achieve a flicker free display, screen double buffering is necessary. Given the tiling implementation it is possible to access the image to be displayed through a pointer indirection. However, in general, there is a mis-mapping between the tile organisation and VRAM memory architecture and in practice it is simpler to copy the whole screen using the memory shift register as an intermediate buffer. This provides a gross copying rate of more than 3G pixels/sec which allows screen updates to be achieved in frame blanking time using the video shift registers which would otherwise be idle.

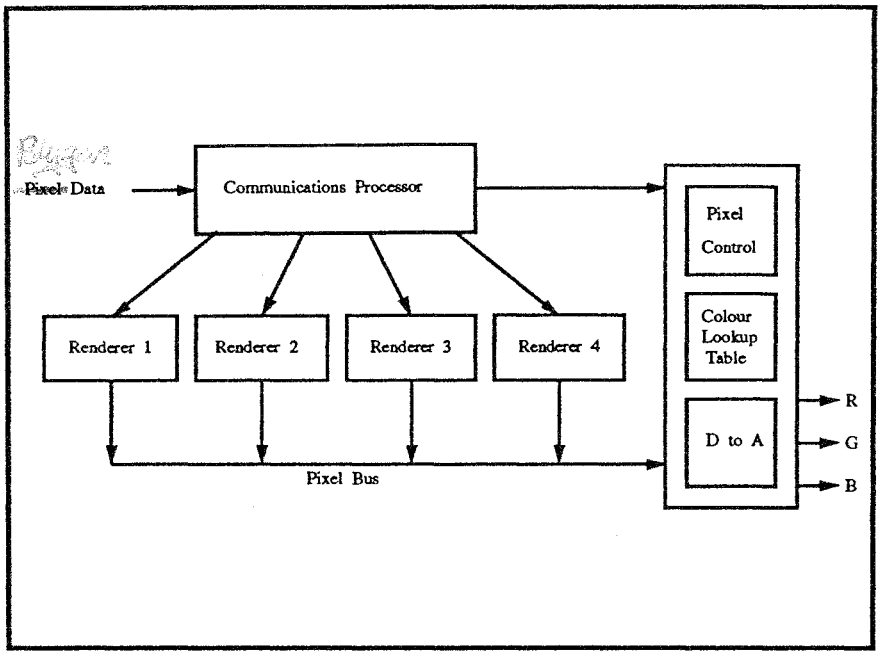

FIGURE 1.5. System Overview

1.3 System Overview

Figure 1.5 shows the system overview. The communications processor simply broadcasts the polygon information to all of the n renderers, where, in fact, n is 4. Each renderer works on a subset of the screen tiles, transforming polygons that cross those tiles and also performing the anti-aliasing computation on those tiles. The communications processor, a T800, also controls hardware which selects and manages the pixel data streams applied to the pixel data bus. For a feasibility study a small screen size of 512×512 pixels was deemed sufficient and the correspondingly modest pixel rates allowed the pixel bus to support just odd and even pixels only. That is, for 8-bit red, green and blue colour values, the pixel bus is 48-bits wide.

1.4 Renderer Architechture

Each renderer, Figure 1.6, consists of a T800 transputer with 4M bytes of local memory, an AD2105 16-bit fixed point DSP, triple ported video memory, and rendering hardware based on the Xilinx 3000 range of logic cell array chips. With a multi-renderer design the communication link structure of Transputers is particularly attractive as it makes expansion of the the number of renderers straight forward. The reprogrammable nature of the Xilinx components is also attractive as it allows incremental changes and improvements without board-level hardware modifications.

The transputer performs the transformations and tile clipping required and it then passes the polygon data, now with screen co-ordinates, through the FIFO to the DSP. This directly controls 5 Xilinx chips; three for the colour incrementers, one for the Z depth calculations, and one for address and tag generation. Starting values and first and second order increments are computed by the DSP and although divisions are required, the small tile sizes allow these to be calculated by multiplication by pre-computed inverses. Such multiplications are single cycle operations.

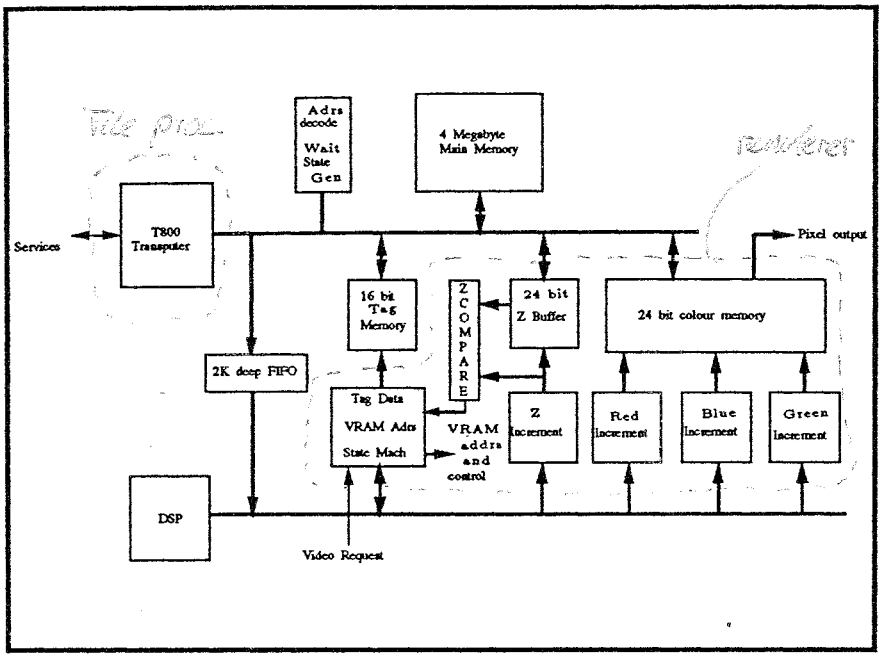

FIGURE 1.6. Renderer Block Diagram

The serial ports on the colour VRAM's are connected to the pixel bus, whilst the serial port on the Z buffer memory feeds comparators used for determining the Z priority and thus a Z read and write operation can be achieved within a single cycle. For practical reasons the tag memory uses the same type of VRAM though the serial ports are unused. Experiments have shown that 16-bit Z depths produced results which were visually unsatisfactory and so 24-bit depths are supported here. The tag memory is 16-bits wide.

After a tile has been completely rendered, control of the triple ported colour and tag memory is passed back to the T800 for anti-aliasing.

1.5 Current Status

The rendering hardware is nearing completion and some estimates of expected performance can be given. Low level rendering speeds are determined by the memory access times and by the Xilinx throughput rates and together these indicate an upper limit of 15M pixel/sec/renderer. Initial testing will be undertaken at 10M pixels/sec. There is some scope for optimisation since where the polygon currently being rendered lies behind an existing object only Z buffer memory reads and Xilinx operations are required. The prime limitation here is the speed of the Xilinx adders, the implementation of which has been found to be inefficient.

Acknowledgements:

This work was supported by the UK Science and Engineerinlatg Research Council (SERC) under grant number GR/F01741. Thanks to Andrew D. Nimmo for his help in preparing the camera ready copy.

1.6 References

[1] D. Coppen, A. Davison, D. Hawes, and M. Slater. Anti-aliasing by postfiltering with precomputed weights. In preparation, 1992.

[2] D. Hawes, M. Slater, A. Davison, and D. Coppen. Graphics object management with a distributed frame buffer. Department of Computer Science report, Queen Mary and Westfield College, Department of Computer Science, Queen Mary and Westfield College, University of London, Mile End Road, London, E1 4NS, 1992.

[3] M. Slater. Segments on bit-mapped graphics displays. *Software—Practice and Experience*, 16(11):965–980, 1986.

[4] M. Slater. An Algorithm to Support 3D Interaction on Relatively Low Performance Graphics Systems. *Computers and Graphics*, 16(3), 1992. Forthcoming.

[5] M. Slater, A. Davison, and M. Smith. Liberation from rectangles: a tiling method for dynamic modification of objects on raster displays. In D. A. Duce and P. Jancene, editors, *Eurographics 88*, pages 381–392. Eurographics, North-Holland, 1988. Republished in Computers and Graphics 13(1) 1989, pp83-89.

[6] R. W. Swanson and L. J. Thayer. A Fast Shaded-Polygon Renderer. *Computer Graphics*, 20(4):95–100, 1986. Proceedings of SIGGRAPH '86.