3D Graphics Tile-Based Systolic Scan-Conversion

Dan Crisu*, Stamatis Vassiliadis*, Sorin D. Cotofana* and Petri Liuha†
*Computer Engineering Laboratory, EEMCS, Delft University of Technology, 2628 CD Delft, The Netherlands
E-mail: {dan, stamatis, sorin}@ce.et.tudelft.nl
†Nokia Research Center, Visiokatu-1 SF-33720, Tampere, Finland
E-mail: petri.liuha@nokia.com

***Abstract*— A 3D graphics systolic scan-conversion unit is presented that solves existing problems associated with tile-based hardware rasterization algorithms. In our proposal no searching overhead is needed to find the first hit position inside the primitives. Furthermore "ghost" primitives are handled efficiently with a small constant delay irrespective of the primitive size. Finally, hit positions (communicated in a spatial pattern to increase texture cache hit ratios) can always be mapped to different memory banks in the Z-buffer or color-buffer breaking the "read-modify-write" dependency associated with depth test and color blending. Hardware synthesis in a commercial 0.18μm process technology has indicated that the hardware implementation requires an area of $269964\mu\text{m}^2$, it can be clocked at a frequency of 200MHz and consumes 33mW. The rendering and the fill rate achieved are 2.4 million triangles/s and 460 million pixels/s for graphics scenes with typical average triangle area of 160 pixels.**

I. Introduction

Tiling or chunking architectures [1] were proposed as a way to save memory bandwidth on frame buffer accesses, and to counteract the huge increase in storage of full-scene antialiasing. In a tiling architecture, the screen is divided in a number of non-overlapping regions, or tiles, which are processed serially. For every frame, primitive geometry is sorted first by screen location and dumped into one or more bins, one bin per tile. Geometry that overlaps a tile boundary is referenced in each tile it is visible in. When all the primitive geometry has been specified, it is rendered from bin N to the tile N before moving to the tile $N+1$. The advantage of the tile-based architectures is that all the data (colors, depth) can be maintained in on-chip tile-sized buffers.

Although many algorithms [2][3], based on edge functions [4], were proposed to rasterize efficiently primitives on traditional full-screen architectures, to the best of our knowledge none was proposed for efficient rasterization in a tile-based architecture. All of the proposed algorithms are based on the following conceptual algorithm: while not all the positions inside the primitive are exhausted do 1) save the rasterization context, 2) move to a new rasterization position, 3) test the edge functions value for that position to see if the position is a hit, 4) if it is inside communicate this hit position to the pixel processing pipelines and update the rasterization context else restore the rasterization context, 5) based on the edge functions computed earlier try to predict a new hit position. Computationwise, the main difficulty in tile-based rasterization with this algorithm is to find the first hit position in the to be rasterized primitive. To establish the overhead needed to find the first hit position we performed experiments with heuristics. We included testing that determines if any of the primitive vertices or the primitive gravity center can be considered the starting rasterization position or the hit point. Our experiments indicated that the overhead can be between 50%-300% of the primitive rasterization time. In addition, primitives are assigned to tiles in the software driver based on the test results of the primitive bounding box and its relationship with the tile boundary. Consequently, there is always overhead associated with "ghost" primitives (depicted in Figure 1), which are primitives that are assigned to the current tile when they do not intersect it. In full-screen rasterization this overhead is inexistent due to the fact that a starting point inside the primitive can always be found, e.g., the gravity center. Apart of the overhead associated to locating the hit position, the traditional full-screen rasterization adapted for tile-based rasterization also exhibits random primitive pixel rasterization order. As several studies [5][6][7] indicate, the primitive pixel rasterization order is crucial for low-cost tile-based architectures that don't have dedicated texture memories (pull texture architectures). Also, a certain primitive pixel rasterization order may allow the interleaving of memory banks in the on-chip tile frame buffers. If this interleaving can be achieved then the dependencies introduced by the "read-modify-write" operation associated with the depth test and color blending can be removed. As a result the throughput of the system can be increased.

To address the previously mentioned drawbacks and open issues, we propose an efficient systolic primitive scan-conversion hardware unit to accelerate primitive rasterization in 3D graphics tile-based rasterizers. The main contributions of our proposal to tile-based rasterization can be summarized by the following:

- the first hit position inside the primitives is found with no overhead,
- "ghost" primitives are efficiently handled, because they are discarded after a small constant delay irrespective of the primitive size. This contrasts with the exhaustive search of the tile boundary required by tile-based rasterizers that adapt the full-screen rasterization approach.

Additionally, our proposal imposes a rasterization order with the following benefits:

- hit positions are communicated in a spatial pattern that has the potential to increase the hit ratio of texture caches

0-7803-8622-1/04/$20.00 ©2004 IEEE

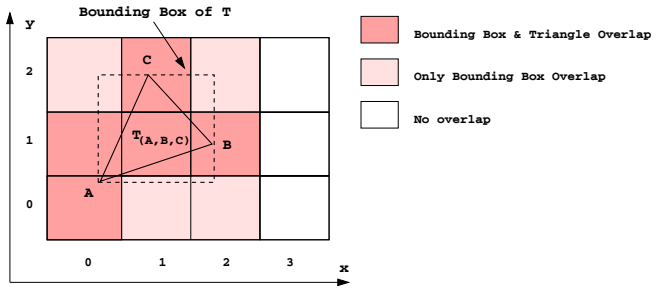

Fig. 1. "Ghost" triangle for tiles $(0,2)$, $(1,0)$, $(2,0)$, and $(2,2)$.

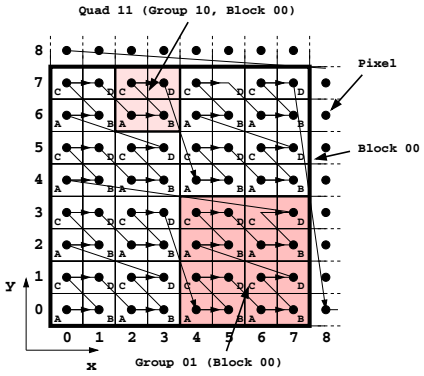

Fig. 2. Proposed pixel rasterization order in tile.

in pull texture architectures;

- hit positions can always be mapped to different memory banks in the Z-buffer or color-buffer breaking the "read-modify-write" dependency associated with depth test and color blending.

The rest of the paper is organized as follows. The systolic primitive scan-conversion subsystem is described in Section II. In Section III, hardware implementation results are presented. Finally, in Section IV, the conclusions are drawn.

II. Systolic primitive scan-conversion

For clarity of explanation and without loss of generality we assume a standard QVGA display size (with a resolution of 320×240) used in mobile terminals, divided in tiles with a size of 32×16 pixels. The screen coordinates x and y of the primitive vertices for a QVGA display are represented as unsigned fixed-point numbers in the format 9.4 (meaning 9 integer bits and 4 fractional bits). We assume that the arithmetic computations are performed in two's complement notation.

The quest to an efficient hardware algorithm for rasterization has to start from finding a suitable pixel rasterization order. In Figure 2 the pixel grid of the tile around the origin of the tile coordinate system is depicted and a proposed space-filling path indicated with arrows starting from the origin is presented. Space-filling paths are known to improve the texel coherency generating high hit-ratio in texture caches [1]. In addition, if 2×2 regions of fragments can be generated during rasterization they can be mapped on different memory banks

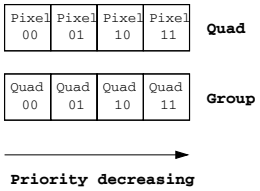

Fig. 3. Pixel and Quad coding.

A, B, C, D. Supposing that the shape or stencil of a triangle has been already coded in a memory representing the bi-dimensional tile, now hit locations have to be forwarded to the pixel processing pipeline. The only way to select between many hit locations according to the space-filling path traversal order is via priority encoding. After the hit location was communicated, the bit for that location has to be reset in order for a priority encoding scheme to work further. Referring to Figure 2, an (x, y) offset position can be encoded in terms of block positions (8×8 fragment regions), group positions (4×4 fragment regions), quad positions (2×2 fragment regions), and positions in quad. Assuming a 32×16 pixel tile, the location $(x, y) = (x_4x_3x_2x_1x_0, y_3y_2y_1y_0)$ can be encoded as $(Block, Group, Quad, Pos) = (y_3x_4x_3, y_2x_2, y_1x_1, y_0x_0)$. With this encoding, priority can be restated hierarchically: hit locations in a block (respectively group, quad) encountered earlier on the space-filling path have a higher priority than any hit locations in a block (respectively group, quad) encountered later on the path (see Figure 3).

The subsystem is part of a larger rasterization system that contains also a logic-enhanced memory. The systolic primitive scan-conversion subsystem, using edge functions, works on a sliding window of 8×8 locations and outputs every clock cycle the primitive shape in the space-filling path order presented in Figure 2 (encoded with one bit per location: ones represent tile pixels covered by primitive, zeros represent pixels not covered) for a different 4×4 pixel region inside the currently processed block. The window is moved to cover all the locations in the tile. The logic-enhanced memory works back-to-back with the systolic subsystem, contains the same number of bits as the number of pixels in the tile, and during rasterization time it will be filled up in several clock cycles by the systolic primitive scan-conversion subsystem with the stencil of the primitive. The hierarchical priority encoding scheme for the hit locations is enforced by the logic-enhanced memory. Once the shape of the primitive has been coded inside the memory, the memory internal logic is capable of delivering on request in one clock cycle at least one and up to four hit positions to the pixel processing pipelines, signaling when all the hit positions were consumed or if none existed. The logic-enhanced memory is described in [8]. For the current tile size of 32×16 pixels, the computations for an entire tile will take 32 clock cycles. Therefore a "ghost" primitive can be detected and discarded in only 32 clock cycles.

A primitive is rasterized using edge functions. In a tile-based rasterizer, for an edge vector $\overrightarrow{AB}$ the edge function can

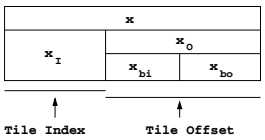

Fig. 4. Fields of the x screen coordinate.

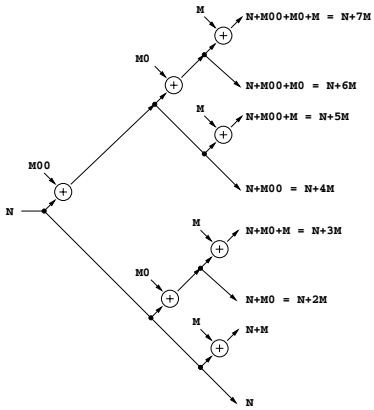

Fig. 5. Parallel computation graph of $x_{bo} \cdot M + N$ for every $x_{bo} \in [0, 7]$.

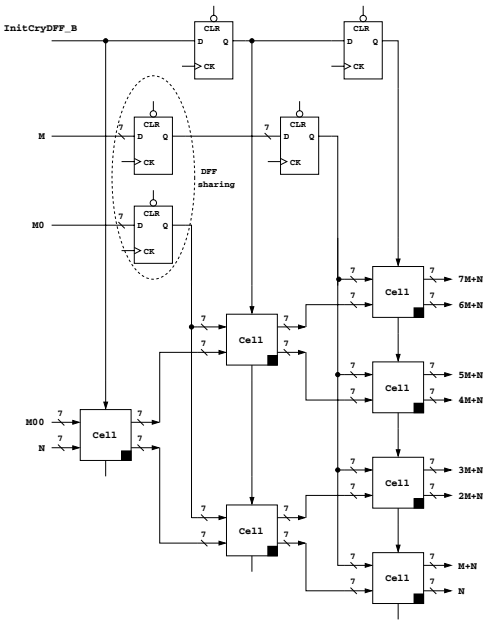

Fig. 7. Systolic computation of $x_{bo} \cdot M + N$ where $x_{bo} \in [0, 7]$.

be reformulated as follows:

$$\begin{aligned} E_{AB}(x, y) &= (x - x_A) \cdot \Delta y_{AB} - (y - y_A) \cdot \Delta x_{AB} \\ &= x_O \cdot \Delta y_{AB} - y_O \cdot \Delta x_{AB} + E_{AB}(x_I, y_I) \\ &= (x_{bi} \cdot 8 + x_{bo}) \cdot \Delta y_{AB} \\ &\quad -(y_{bi} \cdot 8 + y_{bo}) \cdot \Delta x_{AB} + E_{AB}(x_I, y_I) \\ &= x_{bo} \cdot \Delta y_{AB} - y_{bo} \cdot \Delta x_{AB} + (x_{bi} \cdot \Delta y_{AB} \\ &\quad -y_{bi} \cdot \Delta x_{AB}) \cdot 8 + E_{AB}(x_I, y_I) \\ &= x_{bo} \cdot M + y_{bo} \cdot P + N + Q \\ &= (x_{bo} \cdot M + N) + (y_{bo} \cdot P + Q) \end{aligned} \tag{1}$$

where (x_I, y_I) represent current tile coordinates on the screen, (x_O, y_O) represent offset coordinates in a tile, (x_{bi}, y_{bi}) are the block coordinates in the tile, and (x_{bo}, y_{bo}), $x_{bo} \in [0, 7]$, $y_{bo} \in [0, 7]$ represent pixel offsets in the block (see Figure 4).

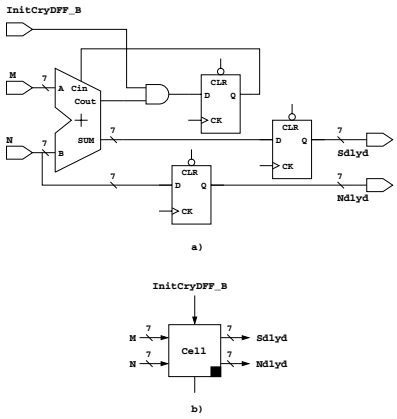

Fig. 6. *Cell* processing element circuit diagram.

The values Δx_{AB}, Δy_{AB} and the quantity $E_{AB}(x_I, y_I)$ are computed at primitive setup time. The quantity $(x_{bi} \cdot \Delta y_{AB} - y_{bi} \cdot \Delta x_{AB}) \cdot 8$ is computed before any computations are started on a new block window and it can be performed efficiently as multi-operand addition (carry-save addition followed by carry-propagate addition). Therefore the last two quantities can be regarded as constants N and Q and what we propose is to compute in parallel the expression $(x_{bo} \cdot M + N) + (y_{bo} \cdot P + Q)$ for every $x_{bo} \in [0, 7]$, $y_{bo} \in [0, 7]$ (when antialiasing is considered the required normalized edge functions [9] can be obtained with a correction of the constant Q with the term $1/2 \cdot (|\Delta x_{AB}| + |\Delta y_{AB}|)$).

The first solution is to compute the expression $x_{bo} \cdot M + N$ and $y_{bo} \cdot P + Q$ for every $x_{bo} \in [0, 7]$, $y_{bo} \in [0, 7]$ using a direct hardware mapping of the graph depicted in Figure 5. In the tree, $M0$ and $M00$ are denoting left-shifted value of M with one position and two positions, they will be derived from M value by some multiplexers outside the tree circuit. The costs for the $E_{AB}(x, y)$ computation in the current block will be prohibitive in both area and latency for this method: it will require 78 28-bit adders (for all three edge function computation this requires 234 28-bit adders) and the critical path will span 4 28-bit adders.

We are proposing a second solution that is more economical in cost and has a very low latency. This is the systolic subsystem described in the following. First, the expresions $x_{bo} \cdot M + N$ and $y_{bo} \cdot P + Q$ can be computed in parallel for every $x_{bo} \in [0, 7]$, $y_{bo} \in [0, 7]$ using a tree of *Cell* processing elements. The *Cell* processing element is depicted in Figure 6 and contains a 7-bit ripple-carry adder, and three D flip-flops: one in which the carries are stored between additions, one to store the result of the current addition and one to delay

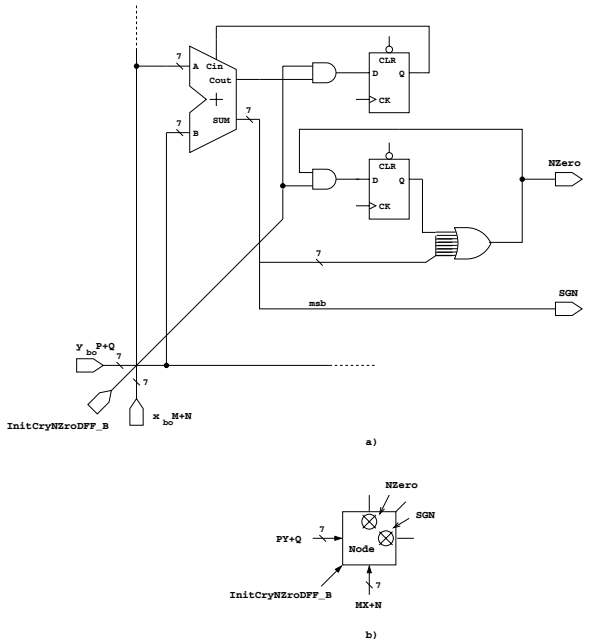

Fig. 8. *Node* processing element circuit diagram.

with one clock cycle one of the operands. The systolic tree computing $x_{bo} \cdot M + N$ is presented in Figure 7. Every clock cycle 7 bits of the 28-bit result are output by the systolic tree starting with the least significant 7 bits.

As 7-bit slices of the values $x_{bo} \cdot M + N$ and $y_{bo} \cdot P + Q$, $x_{bo} \in [0, 7]$, $y_{bo} \in [0, 7]$ are generated every clock cycle, they are combined by *Node* processing elements arranged in a 8×8 matrix. The *Node* processing element is depicted in Figure 8 (for drawing purposes, the outputs in the symbol are drawn with crosses meaning that they are perpendicular on the page). It takes two partial 7-bit results and combines them with a 7-bit ripple-carry adder outputing the edge function sign bit and the edge function not zero flag (to compute the primitive stencil, the values of the edge functions are not interesting per se but their relationship with 0). In addition, another D flip-flop is again required to store the generated carry.

The entire systolic subsystem for an edge function is presented in Figure 9. With additional information (edge quadrant, primitive edge orientation convention, antialiasing enabled or not) the two outputs of every *Node* processing element can be compressed in only one signal (the compression layer depicted in Figure 9). The 8×8 matrix of *Node* elements is generating results for a different sliding window of 8×8 locations every 4 clock cycles. With the addition of the D flip-flop layers depicted in Figure 9, results for a different group (4×4 locations) in the window are generated every clock cycle and with proper multiplexing they are available on the `Inside` output port. Assuming an identical systolic subsystem for each primitive edge, the `Inside` signals for each edge are anded and each bit in the 16-bit result indicates if that particular location in the group is covered by the primitive or not (one for inside the primitive, zero if outside). Every clock cycle, the groups generated are written to the logic-enhanced memory.

TABLE I
HARDWARE IMPLEMENTATION RESULTS

Technology	UMC *Logic18-1.8V/3.3V-1P6M*
Std. Cell Library	VST *e*Si-Route/11
Critical Path [ns]	2.155
Area [μm^2]	269964
Std. Cell Number	
D Flip-Flops (DFF area = 81.3μm^2)	1413
Full Adders (FA area = 65μm^2)	1638
Control Circuitry Gates	7071
Total	10122
Power Consumption [mW]	33
Energy Consumption [mW/MHz]	0.165
Performance	
Rendering Rate [triangles/s]	2.44×10^6
Fill Rate [pixels/s]	460×10^6

III. HARDWARE IMPLEMENTATION RESULTS

The systolic subsystem sliding window size was designed to lead to hardware costs that match the hardware size of a functionally equivalent full-screen scan-conversion unit. Larger sizes, although they may provide benefits from a performance viewpoint, were considered too costly for mobile terminals and were not implemented. We performed the hardware synthesis using Synopsys tools in a commercial 0.18μm IC manufacturing technology. The results for the systolic primitive scan-conversion subsystem for all three edge functions including the required control are presented in Table I. The critical path of the unit can be clocked at a frequency of at least 200 MHz when reasonable clock uncertainty is taken into account. The latency is one primitive stencil computed every 32 clock cycles but the stencil computation is completely hidden by the logic-enhanced memory operation that feeds the pixel processing pipelines. The power consumption was estimated assuming random vectors on the inputs and is presented also in Table I. It should be noted that in reality the actual figure of power consumption may be somewhat lower due to existing signal correlations that are not accounted for in our estimation. The system we are describing is already modeled in SystemC as part of GRAAL [10], a full-fledged 3D graphics OpenGL-compliant tile-based hardware rasterizer. The performance figures presented in Table I are computed for typical triangles with an average area of 160 pixels [11] and indicates the performance of the triangle setup stage and the maximum theoretical pixel fill rate (doesn't account for texture cache miss penalty) that can be achieved with the proposed systolic subsystem.

It would have been of interest to compare our scheme with other designs. Unfortunately implementation details that regard what we have developed (the primitive scan-conversion hardware algorithm) are not available from the existing literature (see for example [12]).

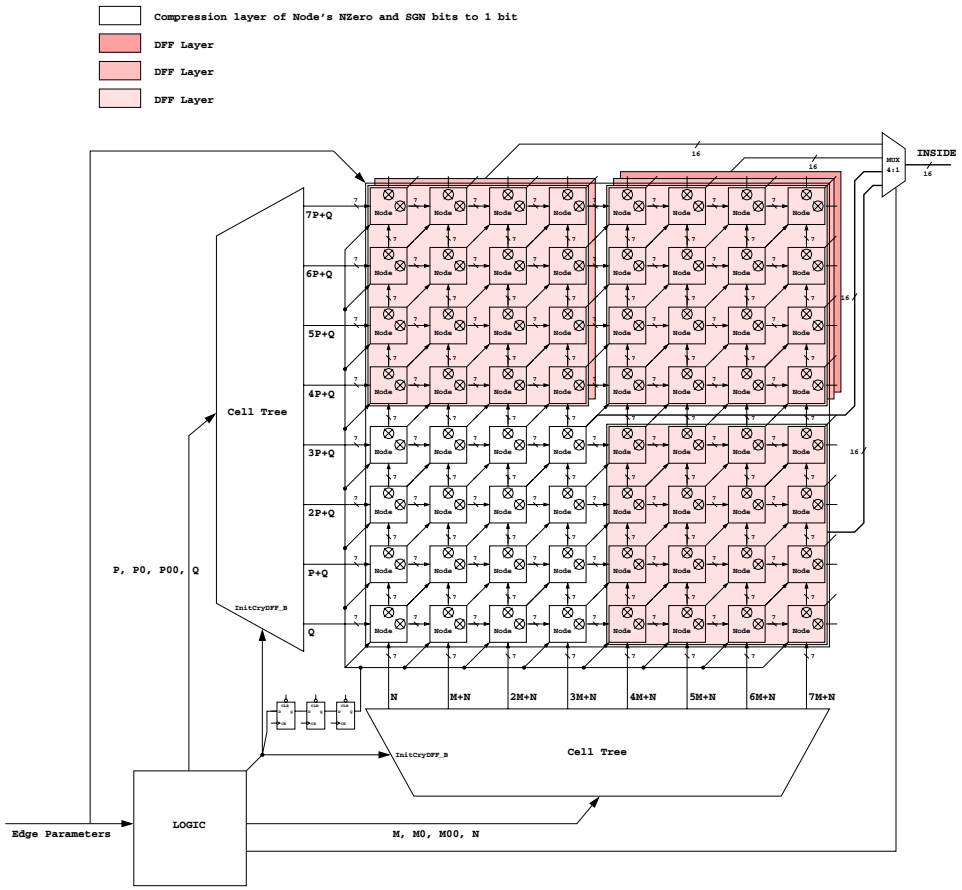

Fig. 9. Systolic computation of the edge function for an 8×8 pixel window.

IV. Conclusions

An efficient systolic primitive scan-conversion subsystem to accelerate primitive rasterization in 3D graphics tile-based rasterizers was presented. The system solves all the complications brought by the tile-based rasterization paradigm. The main features are no overhead in finding the first hit position inside the primitives, efficient handling of "ghost" primitives, and a space-filling rasterization order that simplifies the pixel processing stage hardware design for high throughput.

As the pixel filling-rate performance of a 3D graphics tile-based hardware rasterizer is affected by texturing, in the near future we plan to conduct research on various texture cache architectures to maximize the hit ratio for the proposed pixel rasterization order to sustain the theoretical fill rate figure estimated in Section III.

References

[1] T. Akenine-Möller and E. Haines, *Real-Time Rendering*. A K Peters, Ltd., 2002.

[2] M. Waller, J. Ewins, M. White, and P. Lister, "Efficient primitive traversal using adaptive linear edge function algorithms," *Computer & Graphics*, vol. 23, pp. 365–375, 1999.

[3] J. McCormack and R. McNamara, "Tiled Polygon Traversal Using Half-Plane Edge Functions," in *Proceedings of the 2000 SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, 2000, pp. 15–21.

[4] J. Pineda, "A Parallel Algorithm for Polygon Rasterization," *Computer Graphics (ACM SIGGRAPH '88 Conference Proceedings)*, vol. 22, no. 4, pp. 17–20, 1988.

[5] Z. S. Hakura and A. Gupta, "The Design and Analysis of a Cache Architecture for Texture Mapping," in *Proceedings of the 24th International Symposium on Computer Architecture*. ACM Press, 1997, pp. 108–120.

[6] M. Cox, N. Bhandari, and M. Shantz, "Multi-Level Texture Caching for 3D Graphics Hardware," in *Proceedings of the 25th Annual International Symposium on Computer Architecture*. IEEE Press, 1998, pp. 86–97.

[7] H. Igehy, M. Eldridge, and K. Proudfoot, "Prefetching in a Texture Cache Architecture," in *Proceedings of the 1998 EUROGRAPHICS/SIGGRAPH Workshop on Graphics Hardware*. ACM Press, 1998, pp. 133–142.

[8] D. Crisu, S. Cotofana, S. Vassiliadis, and P. Liuha, "Logic-Enhanced Memory for 3D Graphics Tile-Based Rasterizers," in *Proceedings of the 2004 IEEE International Midwest Symposium on Circuits and Systems (MWSCAS 2004)*, July 2004, pp. II–237–II–240.

[9] A. Schilling, "A New Simple and Efficient Antialiasing with Subpixel Masks," *Computer Graphics (ACM SIGGRAPH '91 Conference Proceedings)*, vol. 25, no. 4, pp. 133–141, 1991.

[10] D. Crisu, S. Cotofana, and S. Vassiliadis, "A Proposal of a Tile-Based OpenGL-Compliant Rasterization Engine," Computer Engineering Laboratory, Delft University of Technology, Deliverable no. (2002)–02, 2002, Tech. Rep.

[11] I. Antochi, B. Juurlink, S. Vassiliadis, and P. Liuha, "GraalBench: A 3D Graphics Benchmark Suite for Mobile Phones," in *Proceedings of ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'04)*, June 2004.

[12] "ARM MBX HR-S 3D Graphics Core — Technical Overview," ARM Ltd. and Imagination Technologies Ltd., 2003.