International Journal of Advanced Science and Technology
Vol.117 (2018), pp.29-40
http://dx.doi.org/10.14257/ijast.2018.117.03

# Design of 2D Vector and 3D Graphics Integrated Structures and Parallel Pipeline Optimization for Mobile Devices

Youngsik Kim

*Department of Game and Multimedia Engineering,*
*Korea Polytechnic University*
*kys@kpu.ac.kr*

***Abstract***

*Most mobile devices support both 2D vector / 3D graphics using 3D graphics hardware, but this method is burdened by the CPU having to process curve tessellation with a large amount of calculation. This paper analyzes the graphics pipeline stages to design and validate a high performance / low-cost efficient hardware pipeline sharing structure that is suitable both for 2D vector graphics and 3D graphics applications. Simulate with real-world applications using 2D vector graphics APIs (OpenVG, Flash, SVG) and 3D graphics API (OpenGL ES) as inputs. Then, this paper designs and validates efficient 2D vector / 3D graphics core shared blocks such as low-cost anti-aliasing and numerical error approximation algorithms based on numerical calculation coverage, and gradients using matrix transformation and texture mapping. In this paper, a function call graph is extracted by profiling a function. This paper also analyzes the frequency, complexity, and association of functions in the full function call graph. This paper presents the optimization parameters such as shared block and bottleneck section analysis of the 2D vector/3D graphics pipeline stage, step delay prediction, etc., and drive parameters such as cache power mode switching criteria, prefetch decision and clock gating criteria. In order to maximize the performance, pipelines are arranged in parallel and pipeline bottleneck analysis is performed to maximize the pipeline throughput, which determines the number of pipeline stages and the step delay time.*

***Keywords:*** *Vector Graphics, 3D Graphics, Parallel Pipeline, Optimization, Mobile Devices*

## 1. Introduction

Hardware that efficiently supports 2D vector and 3D graphics services such as high-quality 2D / 3D user interface, GIS map, game, avatar, and animation in various mobile platforms (Smart Phone, MID, Software sharing structure is urgently needed. 2D vector graphics technology provides high-quality images using vectors such as straight lines, Bezier curves, and elliptical arcs, as opposed to bitmap graphics, which are low quality and can not scale without loss. Adobe Flash, World Wide Web Consortium (SVG), Scalable Vector Graphics (SVG), and Khronous Group's OpenVG API are core technologies that can be used without quality loss when enlarging the screen. Unlike 2D graphics, 3D graphics technology is a core technology that provides realistic graphics processing by processing depth shading algorithms, texture mapping, etc. using 3D depth information, and it is the OpenGL of Khronous group and Direct3D API of Microsoft. 2D vector graphics and 3D graphic technology are being used simultaneously in recent smartphone user interfaces [3,4,5,7,8,10,11].

Received (March 15, 2018), Review Result (June 5, 2018), Accepted (June 7, 2018)

ISSN: 2005-4238 IJAST

As shown in Figure 1 and Table 1, 2D vector graphics technology can be roughly classified into curve tessellation, geometry processing, rasterization, image & gradient, and pixel processing similar to the 3D graphics processing method. 2D vector graphics can share a portion of the 3D graphics hardware pipeline. However, in 2D vector graphics, unlike 3D graphical triangle primitives, it allows unformatted polygons (Bezier curve, concave, convex, *etc.*,) primitives, so it is different from 3D graphics technology in curve tessellation and rasterization inside polygons. 3D graphics are primitives triangulated, so the detail rasterizer steps are possible with triangle setup, edge setup, and span processing. However, the 2D vector graphics can be used to perform active edge setup after curve tessellation and coverage calculation for anti-aliasing, and then use a winding counter (not shown) to support odd-even or non-zero filling rules in unformed complex shape polygons span processing is performed by dividing one scanline into several spans using a winding counter [5,6,7].

In the case of pixel processing, texture mapping of 3D graphics and image processing of 2D vector graphics are similar except for u and v texel mapping of 3D graphics. However, 3D graphics require a depth (z) value test that does not exist in a 2D vector graphic to give a three-dimensional effect, and gradient processing that is not in 3D graphics in a 2D vector graphic is required. Clipping, masking, scissoring, blending, anti-aliasing, stencil, *etc.*, can then be processed similarly. However, 2D vector graphics should support various image filtering such as Gaussian blur [10,11].

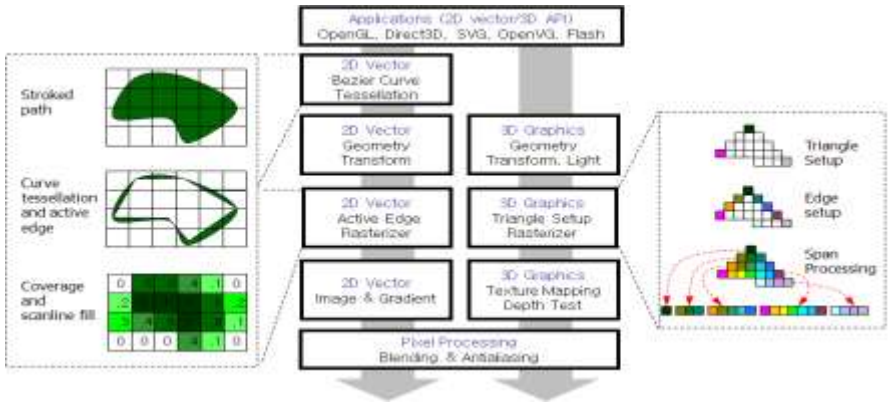



**Figure 1. Features of 2D Vector/3D Graphics Pipeline Structure and Rasterizer**

**Table 1. Comparison of 2D Vector Graphics and 3D Graphics Pipeline**

| | 2D Vector Graphics Pipeline | 3D Graphics Pipeline |
|---|---|---|
| Preprocessing | curve tessellation | no |
| Geometry Processing | transform | transform, lighting |
| Rasterizer (primitive setup and span processing) | edge table setup<br>active edge table setup<br>winding counting fill rule<br>scanline filling (several spans) | triangle setup<br>edge walk<br>span processing |
| Pixel processing | image | texture mapping |
| | gradient(line, radial) | no |
| | no | z test |
| | clipping, masking, scissoring, blending, antialiasing, stencil, image filtering | clipping, masking, scissoring, blending, antialiasing, stencil |

Most mobile devices support 2D vector / 3D graphics using 3D graphics hardware, but this method has a burden of processing the computation-intensive curve tessellation in the CPU [1,2,5]. Therefore, there is an urgent need for a high-performance/low-cost efficient hardware pipeline sharing structure suitable for both 2D vector graphics and 3D graphics applications .by analyzing 2D vector graphics and 3D graphics detail pipeline stages. In particular, the unstructured 2D vector graphics primitive rasterizer and the spanned portion of the 3D graphics triangle rasterizer are consistent, so this previous span setup step must be configured for 2D and 3D graphics processing.

To analyze each graphics pipeline and design an efficient 2D vector/3D graphics accelerator sharing structure, this paper builds a simulation environment that uses real-world applications and APIs. It is necessary to analyze the processing accuracy of the functional block, the bottleneck section of the pipeline stage, and the delay time of the pipeline stage to be reflected in the optimal pipeline shared structure design. Because the 2D/3D graphics pipeline shared structure is based on the amount and complexity (linear, curved, elliptic arc, polygon, *etc*.,) of input primitives (2D vector path, 3D triangle), interpolation level, antialiasing quality, rendering quality, Demand performance and cost grade. This is because it dynamically determines pipeline bottleneck points, operating mode change thresholds, and so on. It is very important to determine the high-performance pipeline design and the low-cost pipeline design parameters according to the required performance such as the frame rate per second of the mobile device and the required cost such as power consumption/gate count. Variable pipeline stages, variable step delay times, variable operating frequencies and operating voltages, variable operating mode decision thresholds, and memory latency hiding for high-performance / low-cost pipeline optimization. Moreover, design parameters such as the crystal must be optimized by simulation analysis.

This paper designs a structure to efficiently share 2D vector / 3D graphics pipeline for 3D stereoscopic images, 2D vector sprites, background, and animation. This paper presents an antialiasing algorithm based on numerical computation which is not supersampling based but low cost. The proposed numerical computation based anti-aliasing technique includes a method of approximating coverage against extreme numerical errors. This paper proposes an efficient rasterization technique using internal memory. This paper also proposes a new algorithm that performs 2D vector graphics gradient functions by reusing matrix transformation and 3D texture mapping blocks.

## 2. Related Works

There are two ways to emulate 2D vector graphics using 3D graphics hardware, depending on how the CPU is processed [1,2]. The first method [1] changes the stroked path of a 2D vector graphic to a triangular primitive through a tessellation such as a plane sweep algorithm in the CPU and then processes the rendering in the 3D graphics pipeline. The second method [2] is changed to a concave polygon through a path subdivision in the CPU and then processed by filling in the path repeatedly using the stencil buffer algorithm in the 3D graphics pipeline. Thus, the emulation method using 3D graphics hardware requires excessive CPU usage. Especially, the first pass tessellation method can cause the CPU software driver's job to become overworked and the rendering frame rate to be interrupted due to tessellation in the middle. The second pass segmentation method reduces CPU computation compared to pass tessellation, but instead uses stencil buffer rendering to consume a large amount of memory bandwidth. In addition, a method using 3D

texture mapping for rendering 2D curves [3, 4] has been proposed, but its functions are limited.

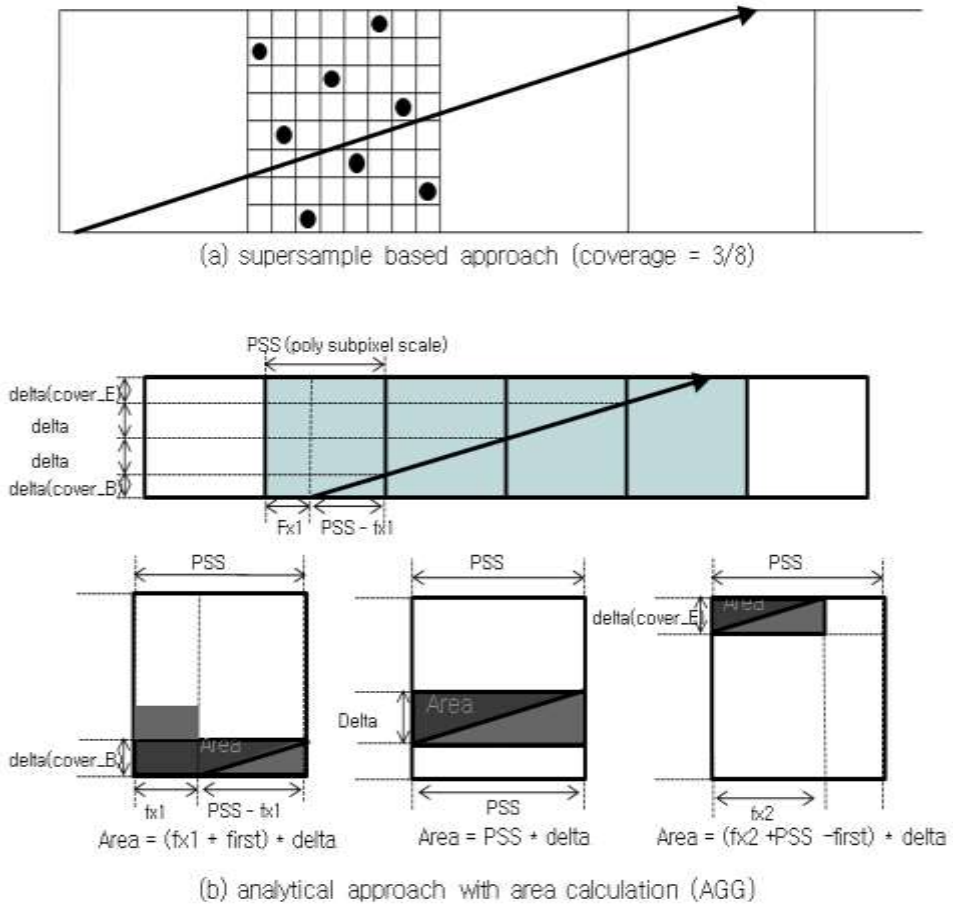



**Figure 2. The Method to Calculate Coverage for Anti-aliasing in AGG Library**

Generally, in order to perform span processing, a path is subdivided into an edge, and then an edge table is configured and an active edge table is configured for each scan line [6, 7, 9]. The anti-grain geometry (AGG) library [13] shows how to create spans using an alignment in the outline table in the open-source 2D vector graphics library. [6] and [7] proposed a scan line-based rendering method that generates spans without alignment in the edge table. However, this method requires a curve subdivision for untyped polygons to be processed by the CPU. There is a drawback that computation requires a lot of memory bandwidth.

Anti-aliasing is a very important technology in both 2D vector graphics and 3D vector graphics. In the case of 3D rendering, however, the texture data is anisotropic (trilinear) beforehand, so that the anti-aliasing focuses on the polygon edge. On the other hand, the high quality of anti-aliasing is even more important because high contrast is often seen in 2D vector rendering. To deal with anti-aliasing, calculate the coverage representing the partial area occupied by the polygon in the pixel. The pixel coverage calculation method has a sample-based method (supersampling) and an analytical approach as shown in Figure 2. The sample-based method [6, 7, 9] computes coverage by segmenting and sampling one pixel into several subpixels with increasing resolution as shown in Figure 2 (a). Increasing the sampling resolution is a robust way to improve anti-aliasing quality. However, the sample-based method requires a large amount of memory bandwidth and low utilization of

the memory. In some cases, artifacts such as a moire pattern or ringing shape may occur even if the sampling resolution is increased. In addition, polygon partitioning with high computational complexity is generally required for self-intersecting polygons [9].

On the other hand, the numerical calculation method as shown in Figure 2 (b) has no sampling artifact and guarantees high anti-aliasing quality. However, you have to numerically calculate the area at the pixel level. Also, in the case of a self-intercepting polygon in which a polygon edge is superimposed inside a pixel, an error may occur in the numerical calculation. The method of Figure 2 (b) is a method of calculating the coverage used in the open source AGG library. In the active scan line, the pixels of the first left pixel and the pixel of the rightmost pixel among the pixels that meet the polygon edge vary according to the X coordinate value and the slope, and the partial areas of the remaining intermediate pixels have a constant value according to the slope of the edge.

The gradient feature is a unique feature of 2D vector graphics that 3D graphics do not provide [10]. A gradient is divided into a linear gradient and a radial gradient. Both require a lot of hardware costs because they involve floating point multiplication, division, and Sqrt operations. Then, when determining the actual pixel color, it is a common method to read the value from a pre-configured LUT (look up table) according to the ratio of the distance obtained by the floating-point operation [13].

The 2D vector / 3D graphics pipeline structure is based on the amount of input data (*i.e.*, triangle and stroked path) and complexity (straight line, curve, elliptical arc, *etc.*), interpolation level, presence of antialiasing, *etc.*, which determine the pipeline bottleneck point, threshold of operation mode change, and so on, it is very important to construct an accurate simulation environment and extract hardware design parameters. Qsilver [15] is an integrated simulation environment study for graphic hardware design parameter extraction. The Chromium [16] tool used by Qsilver intercepts the OpenGL API call stream and annotates the trace to provide the simulation. We also consider an analytical power model, Wattch [17], for structural discovery related to power consumption of graphics hardware. In addition, the software energy profiling technique (PowerSpy [18]) for mobile devices, the task of various application programs, profiling using compiler technique, and the technique of extracting parameters of operating voltage adjustment [19, 20] Respectively.

In this paper, we will construct a simulation environment that uses actual application programs using 2D / 3D graphics API as input to software simulator. We will use the open 3D game engine (Ogre engine [14]) and 2D / 3D graphics library (Vincent 3D [12], and AGG [13]), and will use API function output format change, function call graph, and output value trace as inputs to the simulator. Simulates input with actual graphic API. Then we will analyze span generation rate, rendering speed, internal and external memory size and bandwidth, antialiasing speed and quality, gradient processing speed, texture cache performance and power of existing methods and methods proposed in this study.

## 3. Design of 2D Vector and 3D Graphics Integrated Structures

### 3.1. Coverage Calculation Hardware

This paper uses a numerical computation based coverage generation method that requires significantly less memory bandwidth than the sample-based method. However, when the polygon edge of the self-intersecting path is overlapped within one pixel, the computation of the coverage value of the AGG analyzed by the present inventor results in a calculation error that can not accurately calculate the partial area as shown in Figure 3 (b). Figure 3 (a) shows the median value of the pixel color according to the coverage because the intersection of the self-intercepting path is not inside the pixel. However, if

the polygon edge intersects the pixel exactly symmetrically, Is offset by +, -, coverage is 0, and pixel color does not appear. In order to solve this problem, in the case of a self-intercepting pass, a pixel having a polygon edge overlap must be accurately calculated by dividing the partial area into two parts. However, in order to divide the polygon, the intersection of the edges must be found and the computation becomes very complicated. Therefore, in this paper, we will use a coverage approximation solution which selects one of the intersections of polygon edge in one pixel and accumulation of +, - in the same path. As shown in Figure 3 (c), when pixel coverage is canceled due to +, - accumulation in one pass, the visual artifact is reduced by keeping only one coverage without accumulation.

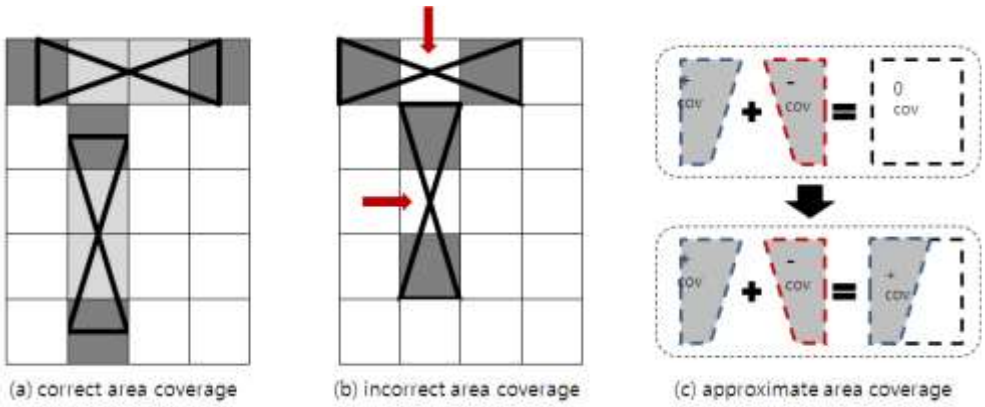



**Figure 3. How to Calculate Numerical Calculation Errors and How to Calculate Approximate Coverage**

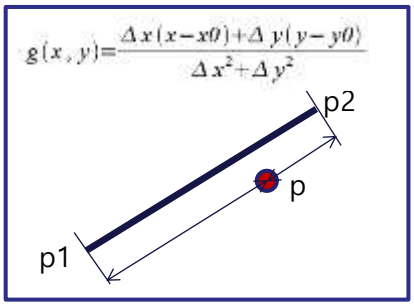
### 3.2. Linear Gradient Hardware

This paper proposes a texture mapping technique which recycles the texture memory used in the 3D graphics pipeline, not the structure that reads the color value from the gradient map into the LUT table, instead of using the floating point arithmetic instead of the floating point arithmetic in the linear gradient. Even in the case of radial gradients, the ratio of distances must be floating point, but the gradient map utilizes the multitexture mapping feature of 3D graphics, not the LUT table. When processing a linear gradient as described in Figure 4, the transformation of P1 to the origin is performed and the rotation is performed so that the P1P2 segment is on the X-axis. Then, scaling transformation is performed so that the length of the P1P2 segment is 1. Then, we can obtain the distance ratio of P in P1P2 segment. Instead of reading the LUT table of K value size, we can obtain color through texture mapping in Kx1 texture image instead. Using this method, expensive floating-point operations can be replaced by matrix transformations used in existing geometric transformations, and texture memory and texture mapping hardware in 3D graphics can be reused in LUT table mapping requiring additional memory.

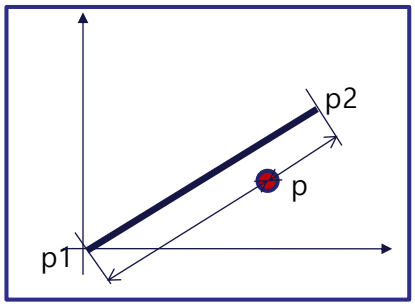
### 3.3. 2D vector and 3D Graphics Pipeline Integrated Structure

Figure 5 schematically shows the hardware parameter extraction and hardware/software comparison verification method by profiling the block diagram and application program of the high performance low cost 2D vector / 3D graphics accelerator shared structure proposed in this study. It simulates 2D vector graphics APIs (OpenVG, Flash, SVG) and 3D graphics API (OpenGL ES) as actual applications that use either or both inputs to provide low cost anti-aliasing and numerical error approximation algorithms based on numerical calculation coverage, Design and validate efficient 2D vector / 3D graphics core shared blocks such as gradients using texture mapping.

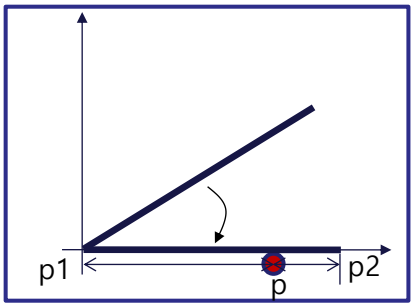
 



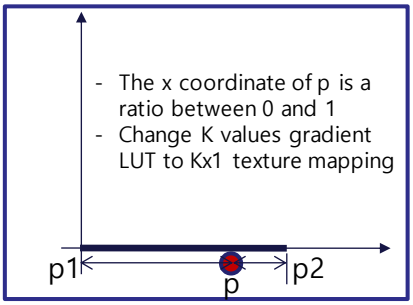

**step0.** linear gradient




**step1.** translation from p1 to origin (p1.x = 0, p2.y =0)




**step2.** rotate p2 so that it overlaps the x-axis (p2.y = 0)




**step3.** scaling to make p2 equal to (1,0) (p2.x = 1)

**Figure 4. Texture based Linear Gradient with Matrix Transform**

It extracts the function call graph and analyzes the frequency, complexity, and association of the functions shown in the whole function call graph. It analyzes the optimized design parameters and the cache power mode switching criterion using the bottleneck section analysis of graph pipeline stage, And the driving parameter such as the clock gating criterion are presented. Depending on the decision parameters of application profiling, pipeline stage integration and threshold determination determine the best trade-off between performance and cost that balances the pipeline load. Among the right parallel pipelines in Figure 5, the grayed pipeline represents a pipeline that is not driven by the drive control of API profiling. Non-driven pixel pipelines use clock gating to achieve optimal performance and power consumption. Driven pipelines that are not grayed also maximize pipeline throughput by varying pipeline stage delay time/clock frequency / operating voltage by pipelining and splitting through pipeline bottleneck analysis of API profiling. For example, if the number of primitives is large and the fragment processing operation of each primitive is small, the bottleneck of the geometry processing occurs. In this case, the pipeline operating frequency of the geometry processing section is maximized to minimize the step delay time. On the other hand, since the fragment processing stage is not part of a relatively critical timing path, some pipeline driving of the fragment processing stage is not performed within the limit that does not affect the overall pipeline throughput, minimize power consumption. In this case, it is a waste of hardware to separately provide a PLL (phased locked loop) for generating a required variable clock frequency, which may be a performance constraint because the stabilization time of the newly generated clock frequency is very long. Therefore, the required clock frequency is generated by dividing the global clock frequency generated by the global clock generator.

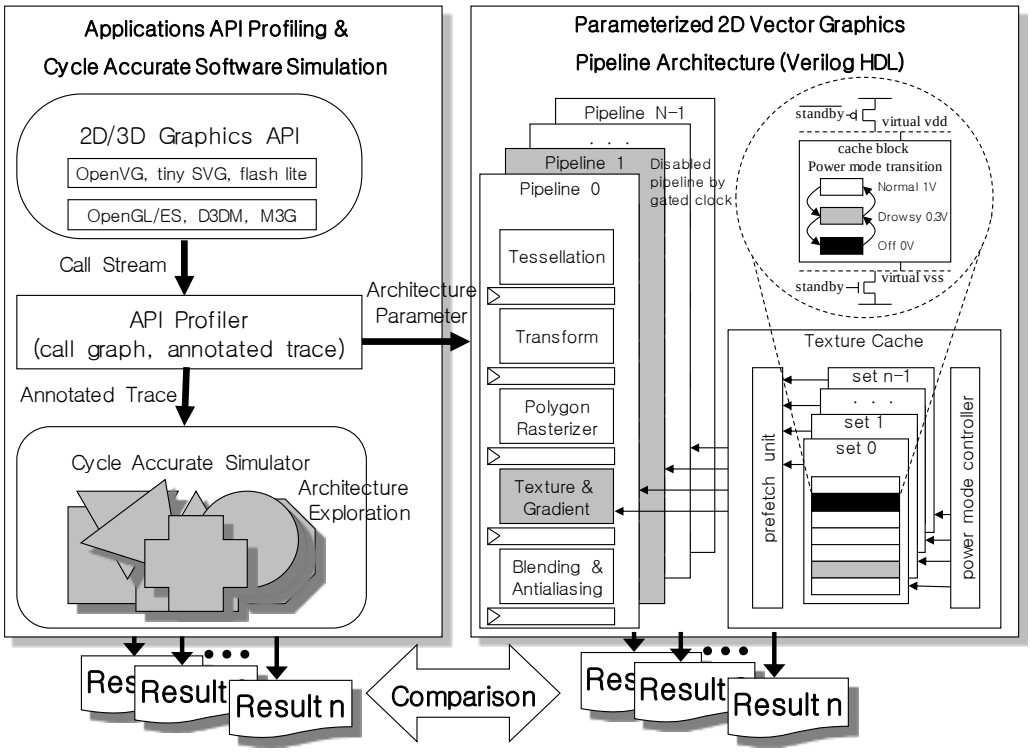



**Figure 5. Proposed 2D Vector/3D Graphics Pipeline Sharing Structure and Verification Method**

## 4. Performance Evaluation

In this paper, we propose a low power texture and pixel cache architecture with shared power mode conversion criteria that are shared by parallel pipelines and reduce memory latency and vary with profiling parameter driven values. To maintain consistency of cache, it has a single cache structure and has a prefetch buffer and a crossbar switch to solve the access bottleneck. The texture cache of this study introduces the variable power mode technique of drowsy cache [21] and snug cache [22]. The white, gray, and black cache blocks in Figure 5 represent the power modes normal mode (1V), drowsy mode (0.3V), and off mode (0V), respectively. However, both the drowsy cache and the snug cache assume a power mode switching penalty of one clock cycle, but this is a very unreasonable assumption. This is because many circuits start driving at once to re-supply the blocked power, which may cause problems in signal integrity such as noise and ground bouncing, and at least several clock cycles are required to prevent this. Also, the drowsy cache and the snug cache have a drawback in which the wait time threshold value of the cache block, which is the power mode switching reference, must be set to a constant value. In Figure 6, we can see that the texture cache hit rate and the average waiting time of the replacement block are significantly different according to the filtering. Comparing Figure 6 (c) and (d) show that switching to the drowsy mode and off mode in bilinear filtering does not affect performance and power consumption can be reduced. In this paper, we design a structure with variable power mode switching threshold according to API profiling parameters. We also develop a prefetch structure and a power mode switching prediction algorithm that predicts the texture coordinates in the power mode of the cache block, which is scheduled to be prefetched according to the control signal, in the normal mode in advance.

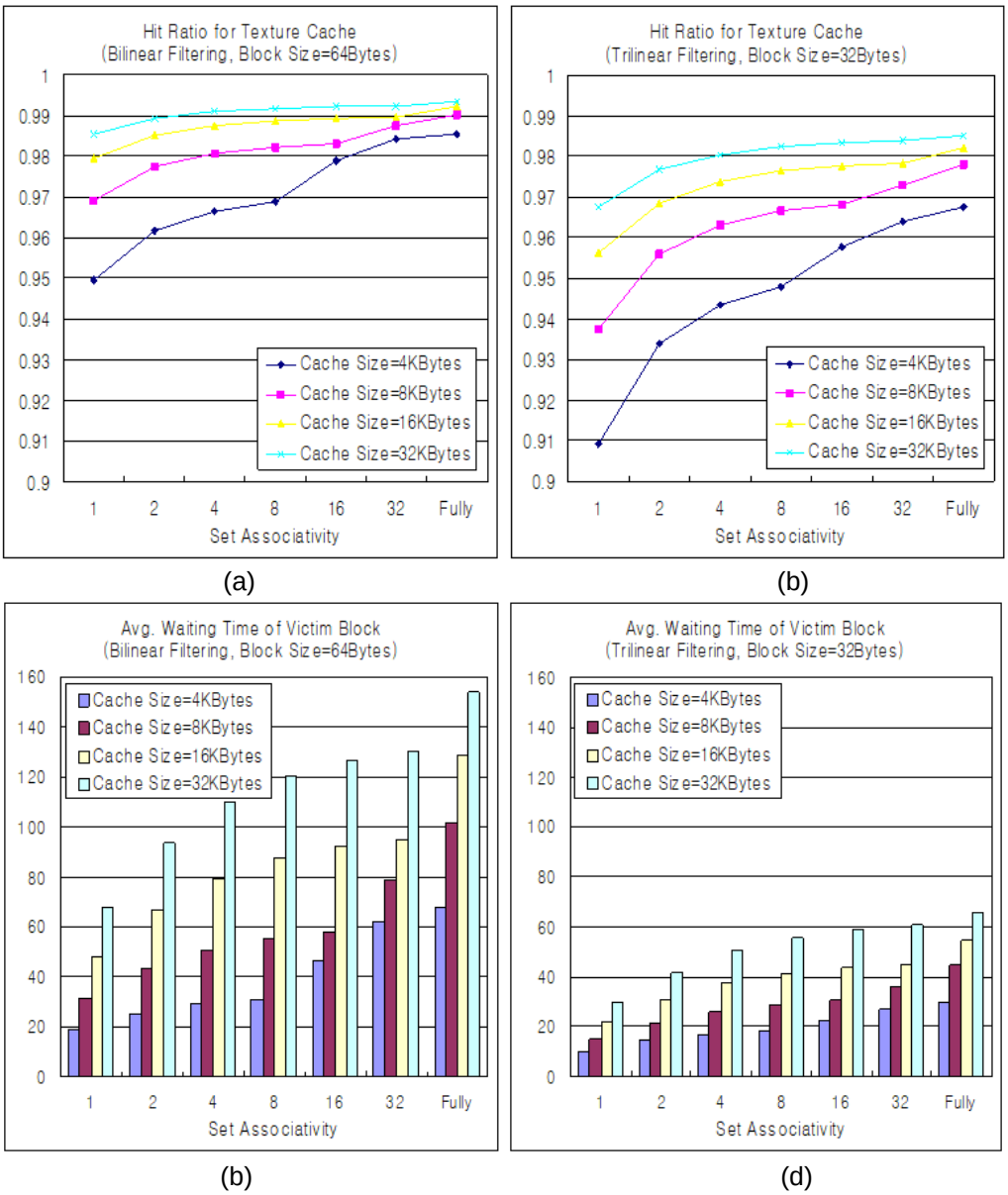



**Figure 6. Texture Cache Experimental Results**

To verify the proposed 2D vector / 3D graphics hardware shared structure, we modify the open source Vincent 3D and AGG library to create a cycle accuracy simulator and perform performance evaluation. API function output format changes, function calls, and output values using the published 3D game engine (Ogre engine [14]) and real applications using 2D vector APIs (flash, SVG, OpenVG) and 3D graphics API Trace is used as a software simulator input. The software simulator's performance evaluation metrics include span creation rate, rendering speed, antialiasing quality and coverage numeric error range, gradient processing performance, the average delay time of each pipeline stage, memory bandwidth, normalized power consumption, total fragment count, the number of texels, the cache miss rate, and the like. The values are then used to design graphics pipelines and design efficient shared structures.

## 5. Conclusion

In order to maximize performance, the proposed architecture performs parallel pipeline pipelining and maximizes pipeline throughput by determining the number of pipeline stages and the delay time of each pipeline by analyzing pipeline bottlenecks. Also, the texture and pixel cache integrated with the parallel pipeline introduces a variable power mode switching technique and proposes a new algorithm that is prefetched according to coordinates previously passed in profiling to reduce memory latency. In order to minimize the cost, the operating voltage and frequency are varied according to the profiling decision parameters. Inoperative pipelines gain low power gain through clock gating technique. It also balances pipeline load with pipeline stage integration and threshold determination.

This paper constructs and profiles a simulation environment using 2D vector graphics API (OpenVG, Flash, SVG), 3D graphics API (OpenGL ES), and real application program as input to extract function call graph. This paper analyzes the frequency, complexity, and association of the functions in the full function call graph and analyzes the optimized design parameters and cache power mode switching criterion using the shared block and bottleneck section analysis of the 2D vector / 3D graphics pipeline stage. We present the drive parameters such as pre-fetch decision and clock gating criteria. To verify the performance and operation of the proposed architecture, a core block is implemented with Verilog HDL RTL to perform hardware/software comparison verification.

## Acknowledgments

This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (No. 2016-0-00204, Development of mobile GPU hardware for photo-realistic real time virtual reality).

## References

[1] J. B. Hobby, “Practical segment intersection with finite precision output”, Computational Geometry, vol. 13, no. 4, **(1999)**, pp. 199-214.
[2] D. Filled, “Concave Polygons Using the Stencil Buffer”, OpenGL Programming Guide (The Red Book, Chapter 13.
[3] Z. Qin, Michael D. McCool and C. S. Kaplan, “Real-time texture-mapped vector glyphs”, Proceedings of the 2006 symposium on Interactive 3D graphics and games**, (2006)**, pp. 125-132.
[4] C. Loop and J. Blinn, “Resolution Independent Curve Rendering using Programmable Graphics Hardware”, ACM Transactions on Graphics, vol. 24, no. 3, **(2005)**, pp. 1000-1009.
[5] AMD Vector Graphics Processor note, www.khronos.org/developers/library/2007_siggraph_bof_openvg/OpenVG%20BOF%204%20-%20AMD.pdf.
[6] K. Kallio, “Scanline edge-flag algorithm for antialiasing”, Theory and Practice of Computer Graphics Conference, **(2007)** June, pp. 81-88.
[7] D. Kim, K. Cha and S. I. Chae, “A high-performance OpenVG accelerator with dual-scanline filling rendering”, IEEE Transactions on Consumer Electronics, vol. 54, no. 3, **(2008)** August, pp. 1303-1311.
[8] Y. Kato, M. Hamada and Y. Inoue, “Multithread Shader for 3D and Vector Graphics”, ACM SIGGRAPH, Research posters, **(2006)**.
[9] K. Doan, “Antialiased rendering of self-intersecting polygons using polygon decomposition”, In Computer Graphics and Applications, PG 2004. Proceedings. 12th Pacific Conference on IEEE, **(2004)**, pp. 383-391.
[10] OpenVG, http://www.khronos.org/openvg/.
[11] OpenGL ES, http://www.khronos.org/opengles/.
[12] Vincent Rendering Library- OpenGL ES 1.1 Open Source implementation http://www.vincent3d.com/Vincent3D/index.html.
[13] Antigrain geometry graphics library http://www.antigrain.com.
[14] Ogre 3D game engine, http://www.ogre3d.org/.

[15] J. W. Sheaffer, D. Luebke and K. Skadron, "A flexible simulation framework for graphics architectures", In Proceedings of SIGGRAPH / Eurographics Workshop on Graphics Hardware, **(2004)**, pp. 85-94.
[16] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner and J. T. Klosowski, "Chromium: A stream processing framework for interactive graphics on clusters of workstations", ACM Trans. on Graphics, vol. 21, no. 3, **(2002)** July, pp. 693-702.
[17] D. Brooks, V. Tiwari and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations", In Proceedings of Int. Symp. Computer Architecture, **(2000)** June, pp. 83-94.
[18] K. S. Banerjee and E. Agu, "PowerSpy: fine-grained software energy profiling for mobile devices", IEEE Int'l Conf. on Wireless Networks Comm. and Mobile Computing, **(2005)**, pp. 1136-1141.
[19] D. Rakhmatov, S. Vrudhula and C. Chakrabarti, "Battery-conscious task sequencing for portable devices including voltage/clock scaling", Proceedings of the 39th annual Design Automation Conference, ACM, **(2002)**, pp. 189-194.
[20] W. Zhang, J. S. Hu, V. Degalahal, M. Kandemir, N. Vijaykrishnan and M. J. Irwin, "Compiler-directed instruction cache leakage optimization", In Microarchitecture, (MICRO-35), Proceedings. 35th Annual IEEE/ACM International Symposium on IEEE, **(2002)**, pp. 208-218.
[21] N. S. Kim, K. Flautner, D. Blaauw and T. Mudge, "Drowsy instruction caches. leakage power reduction using dynamic voltage scaling and cache sub-bank prediction", In Microarchitecture, (MICRO-35), Proceedings, 35th Annual IEEE/ACM International Symposium on IEEE, **(2002)**, pp. 219-230.
[22] J.-J. Li and Y.-S. Hwang, "Snug set-associative caches. Reducing leakage power while improving performance", Low Power Electronics and Design, ISLPED'05. Proceedings of the 2005 International Symposium on IEEE, **(2005)**, pp. 345-350.

## Author

**Youngsik Kim** received the B.S., M.S., and Ph.D degree in Dept. Computer Science from the Yonsei University, Korea, in 1993, 1995, and 1999 respectively. He had worked for System LSI, Samsung Electronics Co. Ltd from Aug. 1999 to Feb. 2005 as a senior engineer. Since March 2005 he has been working for Dept. of Game & Multimedia Engineering in Korea Polytechnic University. His research interests are in 3D Graphics and Multimedia Architectures, Game Programming, and SOC designs.