IEICE Electronics Express, Vol.8, No.11, 788–794

Scanline-based rendering of 2D vector graphics

Sang-Woo Seo[1], **Yong-Luo Shen**[1,2], **Kwan-Young Kim**[3], **and Hyeong-Cheol Oh**[4a)]

[1] *Dept. of Elec. & Info. Eng., Graduate School, Korea Univ., Seoul 136–701, Korea*

[2] *Fac. of Mech. & Elec. Info., China Univ. of Geosciences, Wuhan 430–074, China*

[3] *R&D Center, Advanced Digital Chips, Seoul 135–280, Korea*

[4] *College of Sci. & Tech, Korea Univ. at SeJong, ChungNam 339–700, Korea*

a) *ohyeong@korea.ac.kr*

Abstract: External memory access exacts considerable timing and energy burdens from portable devices. However, most hardware accelerators for rendering two-dimensional (2D) vector graphics draw images in a path-based (path-by-path) manner, which frequently causes excessive external memory traffic. This paper proposes a scanline-based method for rendering 2D vector graphics in portable devices. The proposed method processes all paths spanning a scanline at a time, enabling the use of a scanline-sized internal frame buffer (FB). Using the internal FB, the accelerator can avoid repeated accesses to the external FB and reduce external memory access considerably for images in which many objects overlap with one another.

Keywords: vector graphics, rendering, hardware accelerator, memory access

Classification: Electron devices, circuits, and systems

References

[1] S. H. Kim, Y. Oh, K. Park, and W. W. Ro, "Hardware implementation of a tessellation accelerator for the OpenVG standard," *IEICE Electron. Express*, vol. 7, no. 6, pp. 440–446, 2010.

[2] D. Kim, K. Cha, and S. Chae, "A High-Performance OpenVG Accelerator with Dual-Scanline Filling Rendering," *IEEE Trans. Consum. Electron.*, vol. 54, no. 3, pp. 1303–1311, 2008.

[3] S.-W. Seo, Y.-L. Shen, S.-C. Lee, J.-S. Lee, and H.-C. Oh, "An Accelerator for Rendering 2D Vector Graphics," *Proc. Int. Conf. Computer Graphics and Virtual Reality*, pp. 88–93, 2010.

[4] I. Antochi, B. Juurlink, S. Vassiliadis, and P. Liuha, "Memory Bandwidth Requirements of Tile-Based Rendering," *Lecture Notes in Computer Science*, vol. 3113, pp. 323–332, 2004.

[5] D. Nehab and H. Hoppe, "Random Access Rendering of General Vector Graphics," *ACM Trans. Graphics*, vol. 27, no. 5, Article 135, 2008.

[6] D. F. Rogers, *Procedural Elements for Computer Graphics*, WCB/McGraw-Hill, Boston, MA, 1998.

[7] Khronos Group, OpenVG 1.0 Sample Implementation, [Online] http://www.khronos.org/developers/resources/openvg

DOI: 10.1587/elex.8.788
Received March 01, 2011
Accepted April 28, 2011
Published June 10, 2011

[8] Huayue Tech., OpenVG Release, [Online] http://www.hygraphics.com/English/prod01.htm

1 Introduction

Today's portable devices require hardware acceleration in rendering two-dimensional (2D) vector graphic images, because software-only solutions frequently fail to satisfy the real-time requirements of the devices. Some systems use three-dimensional (3D) hardware engines to render 2D vector graphics, but these 3D engines are still too expensive for many low-end portable devices, including low-cost toys and smart tools. In addition, a number of problems must be solved before portable devices can effectively use 3D engines in rendering 2D vector graphics [1].

Most hardware accelerators, including those in [2, 3], for rendering 2D vector graphics draw images in a path-based (path-by-path) manner. However, these path-based accelerators suffer from excessive external memory traffic for some input images. External memory access exacts considerable timing and energy burdens from a portable device; thus, reducing external memory access is an important issue in designing 2D vector graphics accelerators.

For 3D graphics, techniques such as tile-based 3D rendering methods [4] can be used to address this memory traffic problem. However, no successful technique has been reported for 2D rendering. The cell-based approach [5] can be a candidate, but its implementation may result in additional external memory accesses because of the boundary edges generated for the polygons crossing cell boundaries. It may also increase the time and memory space required for managing the winding values (WVs) [6].

This paper proposes a scanline-based method for rendering 2D vector graphics in portable devices. The proposed method can be considered a 2D version of tile-based rendering, which uses scanline-shaped tiles. Compared with the path-based method [2, 3][1], which processes one path at a time, the proposed method simultaneously processes all paths spanning the current tile.

2 Path-based accelerator

Fig. 1 (a) depicts an exemplary architecture of path-based accelerators. The accelerator processes one path at a time. The Tessellator engine reads the vertices for a path from the external memory, generates the edges for the path, and stores them into the internal edge buffer (EB) unless the internal EB is overflowed. Subsequently, the Rasterizer engine reads the edges and calculates the WVs, as in [2, 3].

The PixelPipe engine reads the WVs, destination and source colors, and texture images (if necessary), and then calculates the new colors to be stored

[1] In [2, 3], the path-based method was called the scanline-based rendering, in the sense that the method processed one scanline at a time while it was rendering a path.

DOI: 10.1587/elex.8.788
Received March 01, 2011
Accepted April 28, 2011
Published June 10, 2011

into the Frame Buffer (FB).

In Fig. 1, the part consisting of Rasterizer and PixelPipe implements the rendering methods, which are of interest in this paper. This part is denoted as *renderer*. Unlike 3D renderers, 2D renderers do not suffer from the computational burden for hidden surface removal because the order of the paths, from back- to front-most, has already been sorted in the application stage.

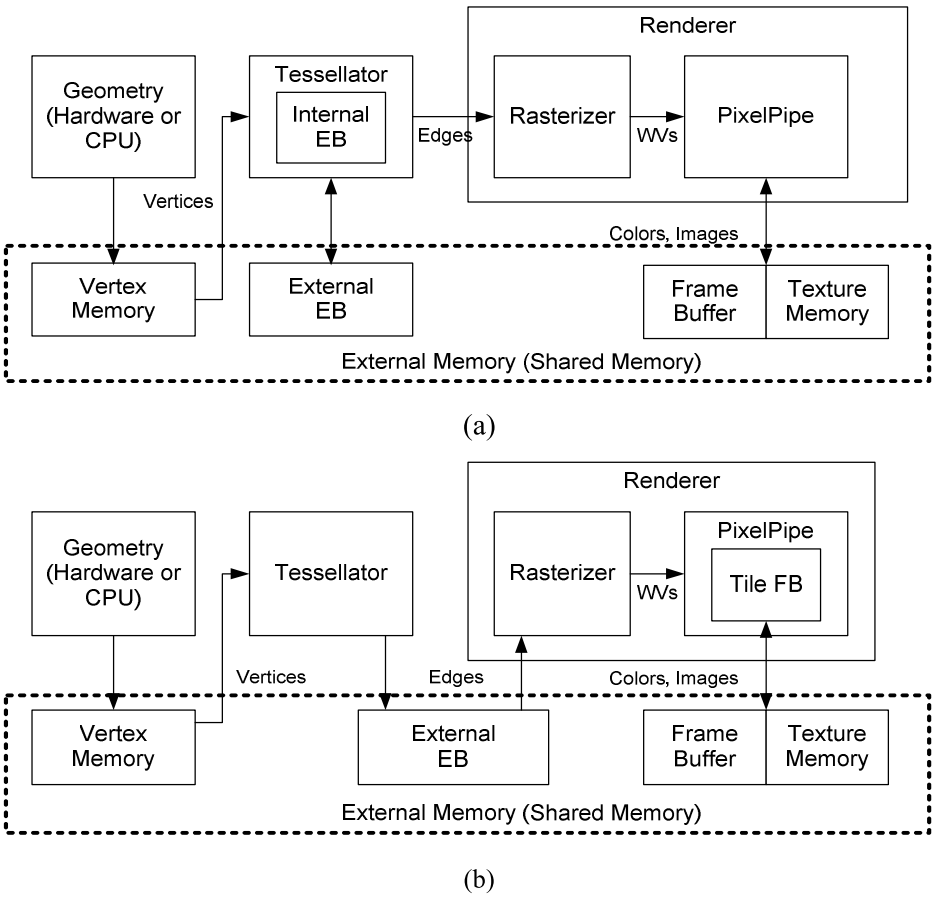

Fig. 1. *Top*: Path-based accelerator. *Bottom*: Proposed accelerator.

3 Proposed (Scanline-based) accelerator

The proposed accelerator processes one scanline, or all paths spanning a scanline, at a time. Fig. 1 (b) depicts the architecture of the proposed accelerator. Tessellator does not use the internal EB. It is because the proposed accelerator has to store the edges for all the paths spanning a scanline, so the amount of edge data to be stored can be too large for the on-chip memories in portable devices. The edges are stored into the regions predefined for their corresponding path, in the external EB.

After the Tessellator engine completes writing all the edges for one image frame, Rasterizer processes the edges in a scanline-by-scanline manner. It reads all the edges spanning the current scanline, calculates the WVs, and

DOI: 10.1587/elex.8.788
Received March 01, 2011
Accepted April 28, 2011
Published June 10, 2011

sends them to PixelPipe. Since the WVs are calculated for one path at a time and then reset for the next scanline, the proposed method does not complicate much the calculation of the WVs.

Using the provided information (WVs, source and destination colors, etc.) for each path, PixelPipe calculates the new colors and stores them into the scanline-sized internal FB, called *tile FB*. After PixelPipe completes all the paths spanning the current scanline, the contents of the tile FB are copied to the (external) FB.

4 Analysis

This paper focuses only on the rendering methods, or the renderers in the hardware. In analyzing the renderers, this paper focuses on the amount of external memory accesses because external memory access is a major source of power consumption, as well as a primary cause of the performance degradation of portable devices.

The three places where external memory accesses occur are (1) between the Geometry engine and Tessellator, (2) between Tessellator and Rasterizer, and (3) at PixelPipe.

The amount of data transferred between Geometry and Tessellator depends on the number of vertices and is decided by the geometry stage. Therefore, this data transfer does not depend on the type of renderers and is not considered in this paper.

The amount of data transferred between Tessellator and Rasterizer, denoted as *M_edge*, depends on the number of edges. The edges are placed in the internal EB for the path-based renderer, unless they are spilled over to the external memory. For the scanline-based renderer, the edges are placed in the external memory.

The external data accessed at PixelPipe mainly consist of the data transferred between PixelPipe and graphics memories (external FB and texture memory). Thus, their quantity, denoted as *M_pixel*, strongly depends on the number of pixels. For the path-based renderer, every color read/write operation causes an external memory access. Conversely, for the scanline-based renderer, managing the tile FB in an internal memory is feasible. Using the scanline-sized tile FB, the renderer can access the external FB only once per painted pixel. For the path-based renderer to use the tile FB, a screen-sized buffer is required.

5 Experiments

An edge is represented in a two-word reduced format. The path-based renderer internally has a 2 KB next active edge list (NAEL) for storing the active edges for the next scanline [3] and uses a 2 KB internal EB in Tessellator. The proposed scanline-based renderer internally has a 640-word buffer as the tile FB and uses a larger (3 KB) NAEL to handle more paths simultaneously.

Four benchmark images, shown in Fig. 2 (a), are used: (1) Tiger [7] (for which stroke operations dominate), (2) Subway [8] (which has many letters),

DOI: 10.1587/elex.8.788
Received March 01, 2011
Accepted April 28, 2011
Published June 10, 2011

and (3) Manga [8] and (4) Clock [8] (for which fill operations dominate).

Both path-based and scanline-based renderers were modeled in Verilog and simulated using Cadence NC-Sim. Xilinx ISE 12.2 suite with the CORE generator was used to generate and simulate memory models such as SRAMs and register files.

The designs were verified by implementing the accelerators shown in Fig. 1 on an FPGA kit with Xilinx xc5vsx95t, as shown in Fig. 2 (b). The system-on-a-chip platform presented in [3] was used in the verification. Even though the proposed scanline-based renderer can be generally adopted for various 2D vector graphics standards, the accelerators used in the verification were designed for the OpenVG standard [7]. The implemented scanline-based accelerator operating at 50 MHz can render a Tiger image of size 640×480 pixels on the screen of size 640×480 pixels, at the speed of about 27 frames per second.

Fig. 2 (c) summarizes the *M_edge* and *M_pixel* required by the renderers. The size of the images, that is, the number of painted pixels affects the

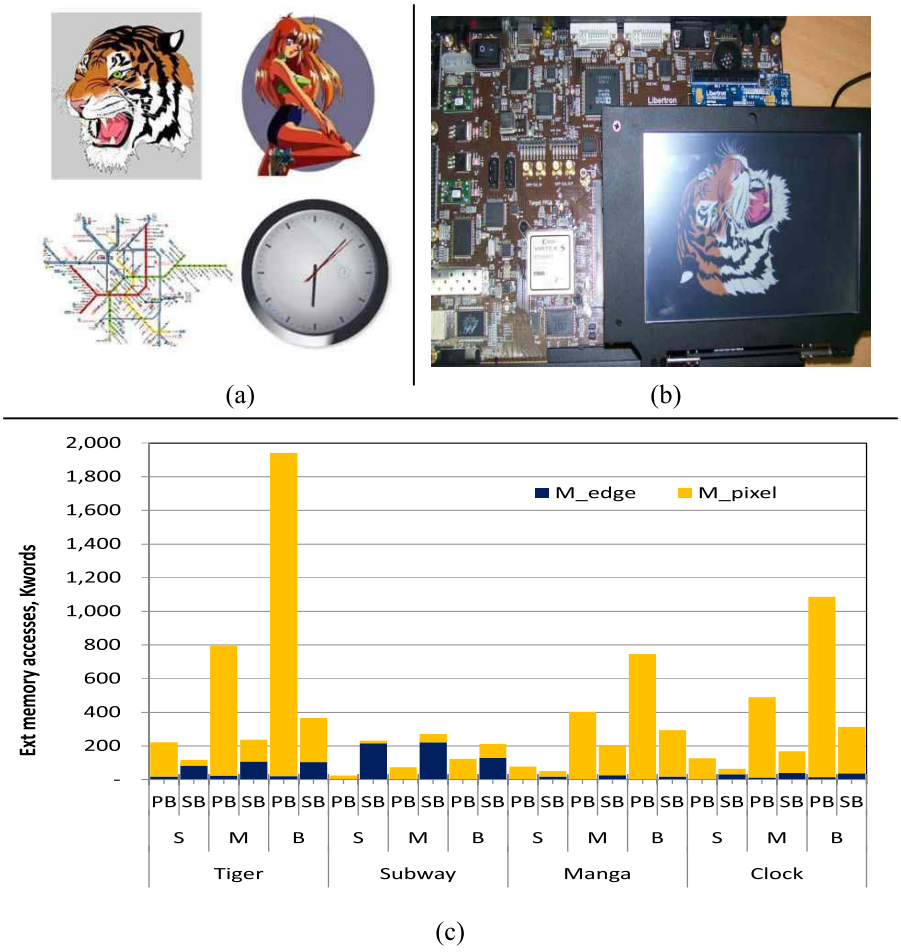

Fig. 2. *Top Left*: Benchmark images, Tiger, Manga, Clock, and Subway (clockwise, from top left). *Top Right*: Design verification using FPGA kit. *Bottom*: *M_edge* and *M_pixel* for the path-based (PB) and scanline-based (SB) renderers (in words).

DOI: 10.1587/elex.8.788
Received March 01, 2011
Accepted April 28, 2011
Published June 10, 2011

performance; thus, images with three different sizes were employed: big (B; 1280×960), medium (M; 640×480), and small (S; 320×240). The screen size is 640×480 pixels for all images. The properties of the benchmark images are summarized in Table I.

Table I. Specification of benchmark images.

Benchmark image	Size	# of vertices	# of edges	# of painted pixels
Tiger	Small (S)	11,100	18,421	32,453
	Medium (M)	14,142	23,144	128,435
	Big (B)	18,451	21,477	262,557
Subway	Small (S)	57,528	51,269	14,208
	Medium (M)	57,528	51,269	49,122
	Big (B)	57,528	28,531	84,320
Manga	Small (S)	4,041	3,733	32,318
	Medium (M)	5,130	5,117	180,016
	Big (B)	6,419	4,858	275,808
Clock	Small (S)	1,066	7,641	32,506
	Medium (M)	1,822	9,379	129,373
	Big (B)	3,370	8,264	275,808

The path-based renderer performs well for Subway because there are fewer objects overlapping with one another (that is, fewer repeated accesses of color values to the external FB). Furthermore, although Subway includes considerably more edges than do the other images (Table I), each path of Subway consists of a small number of edges. Thus, the path-based renderer can manage all the edges in the internal EB and NAEL.

On the other hand, except for Subway, the scanline-based renderer accesses much less pixel data and reduces external memory accesses by about 55% (harmonic mean) compared with the path-based renderer. A maximum reduction of 81% is observed for the image with many objects (paths) overlapping with one another, for which the repeated updates of pixels to the external FB can be reduced using the tile FB.

In the experiments, the scanline-based renderer uses about 1.5 KB more memory. However, the path-based renderer requires about 1.2 MB buffer to adopt the tile FB, for the screen size used in the present paper.

6 Conclusion

A scanline-based method for rendering 2D vector graphics has been proposed to reduce external memory access. The reduction comes mainly from decreasing the repeated accesses of the color values to the external FB. Thus, the proposed renderer is superior to the path-based renderer for images in which many objects overlap with one another. Such images usually cause bottlenecks in the real-time rendering operation of portable devices.

Even though the accelerators were designed based on the OpenVG standard in the experiments, the renderers were independent of the API standard of 2D vector graphics. Thus, the proposed rendering method is not confined to any specific API standard of 2D vector graphics.

DOI: 10.1587/elex.8.788
Received March 01, 2011
Accepted April 28, 2011
Published June 10, 2011

Acknowledgement

This work was supported financially in part by Korea University. The CAD tool was provided by IDEC, Korea.

DOI: 10.1587/elex.8.788
Received March 01, 2011
Accepted April 28, 2011
Published June 10, 2011