IEICE TRANS. INF. & SYST., VOL.E92–D, NO.7 JULY 2009

LETTER

Adaptive Scanline Filling Algorithm for OpenVG 2D Vector Graphics Accelerator

Daewoong KIM[†,††a)], ***Student Member***, **Kilhyung CHA**[†], ***and*** **Soo-Ik CHAE**[†], ***Nonmembers***

SUMMARY We propose an optimized scanline filling algorithm for OpenVG two-dimensional vector graphics. For each scanline of a path, it adaptively selects a left or right scanning direction that minimizes the number of pixels visited during scanning. According to the experimental results, the proposed algorithm reduces the number of pixels visited by 6 to 37% relative to that with a constant scanning direction for all the scanlines.

key words: *OpenVG, two-dimensional vector graphics, adaptive, scanline filling, scanning direction, skew*

1. Introduction

Recently, applications using two-dimensional (2D) vector graphics such as SVG viewers, portable mapping applications, E-book readers, games, and scalable user interfaces have been widely accepted for mobile devices because the 2D vector graphics requires input data files with relatively smaller size, provides compression without artifacts, and offers easy scalability for various display panels sizes [1], [2]. OpenVG, which is the 2D vector graphics standard constituted by the Khronos Group., provides a royalty-free, cross-platform API for vector graphics libraries such as Flash and SVG [2]. Path, paint, and image are three basic components in OpenVG 2D vector graphics. All the geometric objects to be drawn are defined by one or more paths, each of which consists of a sequence of segment commands. Each segment command in the standard format may specify a move, a straight line segment, a quadratic or cubic Bézier segment, or an elliptical arc. A paint command defines a color and a transparent effect, which is called a filtered alpha value (FAV), for each pixel being drawn, and images are rectangular collections of pixel effects such as texturing.

In designing an efficient accelerator for OpenVG 2D vector graphics, optimization of its rendering part is important because of its computational complexity is substantially higher than that of its geometry part [3]–[6]. We found that the rendering part occupies more than 80% of the total computational complexity in most of the test images we used [6]. The rendering architecture can be classified into three types: vector, raster, and hybrid [4]–[6]. Among them, we focus on hybrid (or scanline-based) rendering architecture, which is more suitable for low-power and high-performance mobile applications because it requires only a scanline-sized buffer and removes unnecessary memory accesses associated with the sorting procedure [5], [6]. This architecture also consists of a geometry part and a rendering part, as shown in Fig. 1. After processing geometry operations for each path, three processing steps for rendering such as active edge generation, scanline filling, and pixel processing are pipelined for each scanline in the path. The scanline filling step, which calculates the FAV for each pixel, is the most complex among the three steps [6] and its complexity for each scanline is proportional to the number of active edges as well as the resolution of display panels (i.e. the number of pixels).

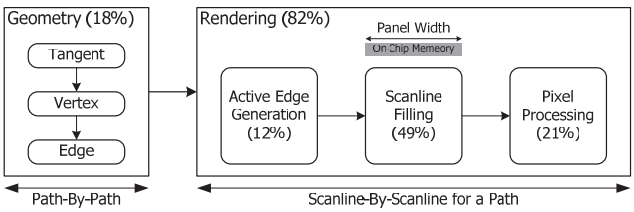

Fig. 1 A scanline-based rendering architecture.

In this paper, we propose an optimized scanline filling algorithm to alleviate a performance bottleneck of the rendering algorithm.

2. Adaptive Scanline Filling Algorithm

2.1 Motivation

To obtain the FAV of a pixel, the partial winding value contributed from each active edge is accumulated according to the crossing direction of the active edge. Figure 2 illustrates two examples for filling a triangle path: one scanned with a constant direction and the other with an adaptively selected direction for each scanline, together with another example for filling a two-polygon path with adaptively selected direction for each scanline. When the scanning direction is right, the partial winding value of each pixel on its right side is decremented by 1 if an active edge is downward; otherwise, incremented by 1, as shown in Fig. 2 (A). When the scanning direction is left, the partial winding value of each pixel on its left side is incremented by 1 if an active edge is downward; otherwise, decremented by 1.

Note that the partial winding values of a pixel

Manuscript received November 28, 2008.
Manuscript revised February 6, 2009.

[†]The authors are with the School of Electrical Engineering and Computer Sciences, Seoul National University, 151–742 Korea.
[††]The author is with Media Development Team, Samsung Electronics Ltd., 446–711 Korea.
a) E-mail: dwkim316@sdgroup.snu.ac.kr

DOI: 10.1587/transinf.E92.D.1500

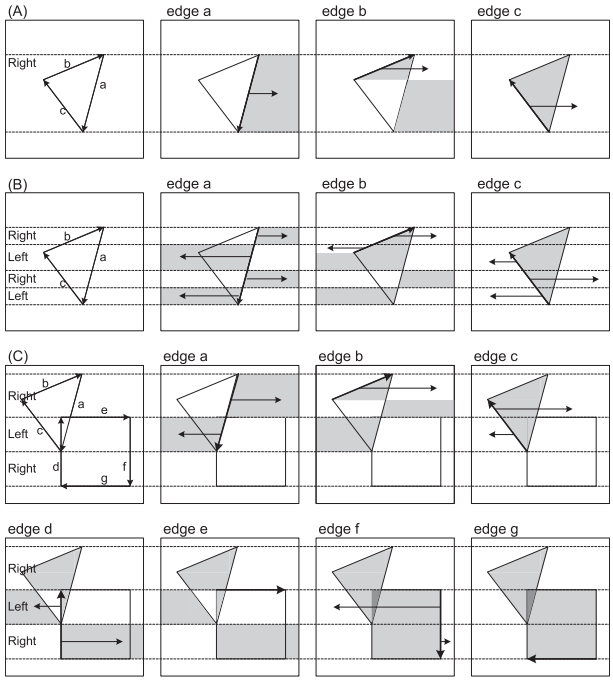

Fig. 2 Filling a triangle path: (A) scanned with a constant direction and (B) scanned with both directions, and (C) filling a path with two polygons scanned with both directions.

contributed from two opposite directional active edges are cancelled. According to these rules, we can obtain correct filling even though we adaptively select the scanning direction, as shown in Fig. 2 (B) and 2 (C), as long as we use a constant scanning direction for each scanline. The dark grey region in Fig. 2 (C) should be filled if a non-zero fill rule is applied or not filled if an even/odd fill rule is applied. Note that filling the overlapped region depends on which fill rule is selected [2]. We can fill a path correctly either with constant direction scanning or with adaptive one. With adaptive scanning, therefore, we can further optimize the scanline filling step by selecting a scanning direction that requires fewer operations.

The scanline filling algorithm first evaluates the edge functions for the edge pixels that intersect with the active edge as well as the first non-edge pixel in the scanning direction. Then it copies the partial winding value of the first non-edge pixel to those of all the remaining pixels in the scanning direction. In Fig. 3, each edge pixel is represented as a grey-color rectangle and two candidates of the first non-edge pixel are represented with a circle. Note that the first non-edge pixel is selected from the two candidates according to the scanning direction. For example, three active edges exist in the scanline shown in Fig. 3. If the scanning direction is right, fourteen copy operations are required for the pixels annotated with a rectangle; otherwise, ten copy operations for the pixels with a triangle. In this example, therefore we should scan to the left to reduce the number of copy operations, or the number of pixels visited.

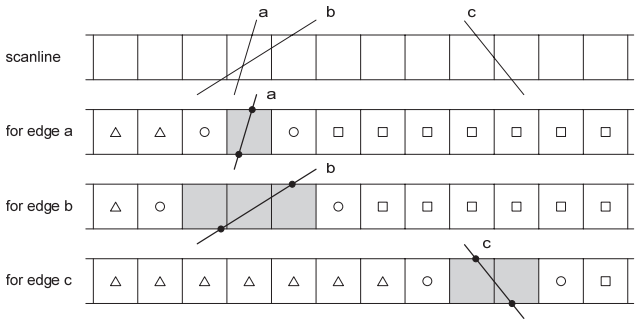

Fig. 3 Winding value calculations of each pixel for active edges in a scanline.

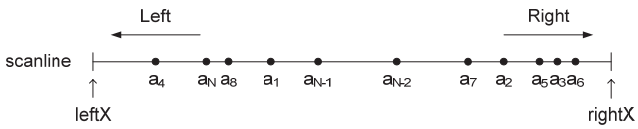

Fig. 4 A scanline with N active edges.

2.2 Criterion for Selecting the Scanning Direction

Now we will derive a simple criterion to select for each scanline a scanning direction that minimizes the number of pixels visited while evaluating their partial winding values.

Let a_i be the pixel coordinate of the midpoint for an active edge i, and N be the total number of the active edges in a scanline. We calculate the leftmost and rightmost crossover points of a polygon with the scanline in the active edge generation step, which will be referred to as *leftX* and *rightX* respectively, as illustrated in Fig. 4.

Because it is efficient to confine the copy operation only for the pixels within the interval of the leftmost and rightmost crossover points, the number of copy operations can be approximated as follows:

$$\begin{aligned} C_{left} &= (a_1 - leftX) + \cdots + (a_N - leftX) \\ &= \sum_1^N a_k - N \cdot leftX \end{aligned} \tag{1}$$

$$\begin{aligned} C_{right} &= (rightX - a_1) + \cdots + (rightX - a_N) \\ &= N \cdot rightX - \sum_1^N a_k, \end{aligned} \tag{2}$$

where C_{left} and C_{right} represent the number of copy operations for the left, or right, scanning direction, respectively. Here, we define the skew of a scanline as $(C_{left} - C_{right})/2N$, which is equal to the mean pixel coordinate of the midpoints for all active edges minus $(rightX + leftX)/2$. We use it as a selection measure for deciding the scanning direction for the scanline like the following. If a scanline has a positive skew, scan it to the right; otherwise, to the left. Consequently, the number of the pixels visited during scanning is reduced.

Table 1 Features of test images.

Test Image	Number of Paths	Number of Total Edges	Number of Total Active Edges for QVGA
Tiger	305	225,297	255,538
Dude	167	83,171	89,921
E-book	19	44,955	57,380
Picture	712	50,820	67,019
Basket	719	77,904	86,390
Bottle	899	46,565	57,311
Snow	481	16,922	22,945
Pelican	95	4,027	9,013

Table 2 Scanning direction and the number of pixels visited for the proposed algorithm.

Test Image	Scanning Direction		Number of Pixels Visited (x 10^6)			Reduction Ratio Compared to (%)	
	Right	Left	Right only	Left only	Proposed	Right only	Left only
Tiger	3022	2577	4.114	4.125	2.589	62.9	62.8
Dude	1477	1074	0.643	0.591	0.564	87.7	95.4
E-book	118	123	4.118	4.015	3.875	94.1	96.5
Picture	5474	2141	0.979	0.968	0.899	91.9	92.9
Basket	2647	1902	0.221	0.203	0.194	87.8	95.5
Bottle	4054	2182	0.299	0.285	0.276	92.4	97.1
Snow	2155	868	0.242	0.233	0.220	90.8	94.4
Pelican	1047	601	0.262	0.283	0.221	84.5	78.0

3. Experimental Results

We used eight test images summarized in Table 1 to verify the efficiency of our algorithm and their full color images are shown in Appendix. Among them, Tiger and Dude are representative test images released from Khronos Group [2], and the others are translated from SVG files by the authors. In the experiment we employed an equal-weight 8-Queen box filter for anti-aliasing filtering and assumed a QVGA display panel.

We compared the proposed algorithm to a conventional one. Table 2 shows the number of scanlines scanned in either direction in the proposed algorithm for each test image as well as the number of the pixels visited for three scanning schemes: right only, left only, or adaptive. The proposed algorithm always reduces the number of the pixels visited relative to that with a constant direction. Note that the proposed algorithm performs better especially for Tiger and Pelican because they include many scanlines with more positive, or negative, skew.

4. Conclusions

In the proposed scanline filling algorithm, the scanning direction for each scanline is adaptively selected according to its skew, which is equal to the mean pixel coordinate of the midpoints for all active edges minus $(rightX + leftX)/2$. Experimental results show that the proposed algorithm considerably improves the performance with minimal computational overhead. Presently, we are focusing our efforts on reducing the memory bandwidth of the OpenVG 2D vector graphics accelerator by optimizing its memory architecture.

Acknowledgments

This work was supported by "System IC2010" project of Korea Ministry of Knowledge Economy and Inter-university Semiconductor Research Center (ISRC) in Seoul National University.

References

[1] K. Pulli, "New APIs for mobile graphics," Proc. SPIE — The International Society for Optical Engineering, vol.6074, no.607401, 2006.
[2] Khronos Group Inc. OpenVG 1.0.1 Specification: http://www.khronos.org/openvg/
[3] G. He, B. Bai, Z. Pan, and X. Cheng, "Accelerated rendering of vector graphics on mobile devices," Lecture Notes in Computer Science, vol.4551, pp.298–305, 2007.
[4] S.-Y. Lee and B.-U. Choi, "Vector graphic reference implementation for embedded system," Lecture Notes in Computer Science, vol.4761, pp.243–252, 2007.
[5] K. Kallio, "Scanline edge-flag algorithm for antialiasing," Theory and Practice of Computer Graphics Conference, pp.81–88, June 2007.
[6] D.-W. Kim, K.-H. Cha, and S.-I. Chae, "A high-performance OpenVG accelerator with dual-scanline filling rendering," IEEE Trans. Consum. Electron., vol.54, no.3, pp.1303–1311, 2008.

Appendix

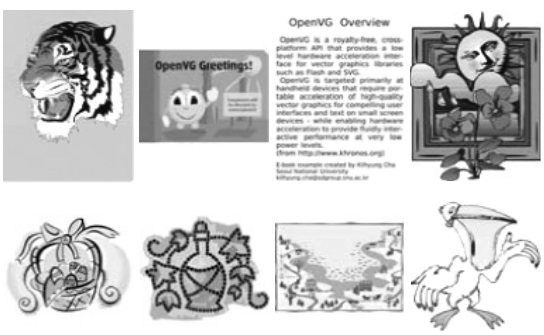

Fig. A·1 Test images: Tiger, Dude, E-book, Picture, Basket, Bottle, Snow, and Pelican (from left-to-right and top-to-bottom).