Rendering 2D vector graphics on mobile GPU Devices

Harish Kumar[1], Anmol Sud[1]

[1]{harkumar, sud}@adobe.com

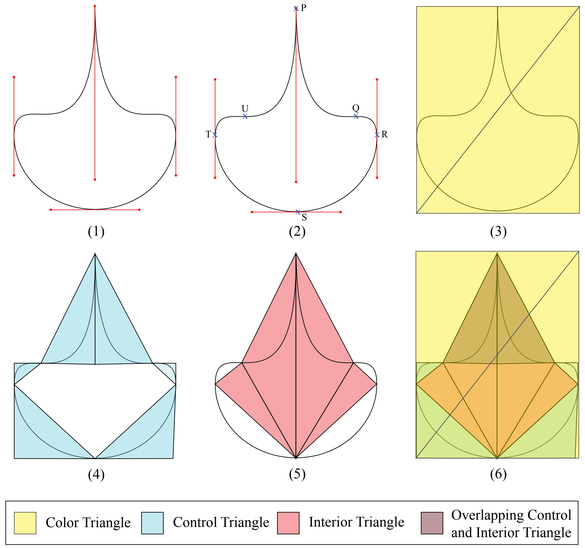
Introduction

Designers and artists world-wide rely on vector graphics to design and edit 2D artwork, illustrations and typographic content.There is a recent trend of vector graphic applications moving to mobile platforms. These vector applicationsare not read only but also requires real time vector editing experience.

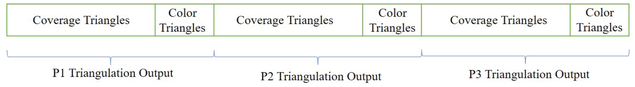
Our solution builds upon standard stencil then cover paradigm and develops an algorithm targeted for GPUs based on tile based deferred rendering architecture. Our technique provides an efficient way to use signed distance based anti aliasing techniques with 'stencil then cover' paradigm. Our solution builds over the stencil then cover method but solves the perfomance issues due to multisampling, two render passes for each vector objects and also allows batching of vector objects for optimal performance. Mobile GPUs (tile-based GPUs) allows reading current pixel memory without any performance penalty (texture barriers or different draw calls). This technique exploits this fundamental property of mobile GPUs.

Method

CPU Triangulation

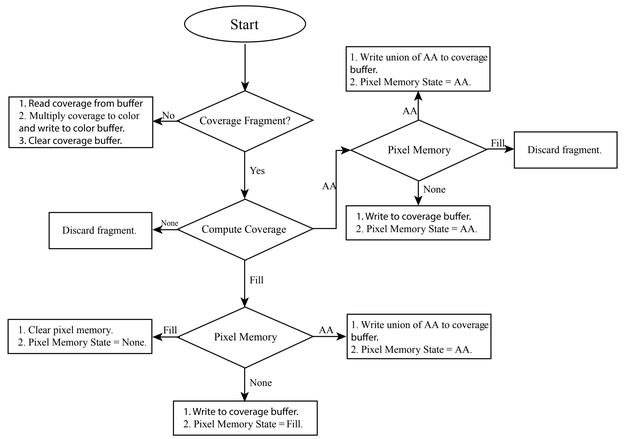


1. Input vector shape 2. Cubic Beziers approximated by Quadratic Beziers 3. Color Triangles 4. Quadratic Bezier Control Triangles 5. Interior Polygon Triangulation 6. Total Coverage triangles and color triangles



Vertex Buffer Layout for drawing multiple paths in a single draw call

Fragment Shader State Machine



Fragment shader reads current values in frame buffer (pixel memory) to decide the state changes in pixel memory for current fragment. On mobile devices, reading curent pixel memory from framebuffer is free of performance cost using 'Framebuffer Fetch'. Our tecnique uses a single render pass to write coverage and color values and clears the coverage buffer in same render pass. This allows to batch and render multiple vector objects n a single render call leading to significant performance gains.

Results

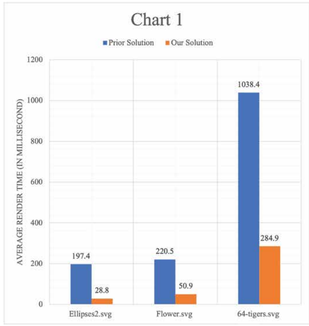


Chart 1: First frame draw comparison (w/o cache)

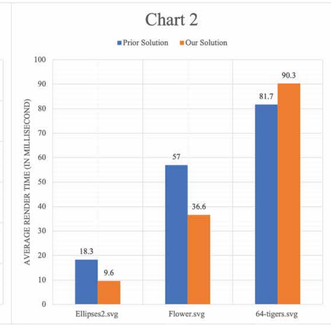


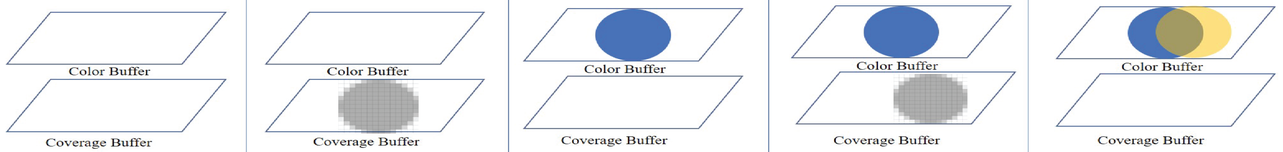
Chart 2 : Frame redraw comparison (with cache)

Artwork quality rendered using our technique

Related Work

Conventionally, vector objects are rendered with either stencil then cover techniques or tessellation based methods. Stencil then cover methods rely on multisampling for antialiasing but multisampling increases memory usage and impacts performance due to per sample shading with transparency and blend modes applied over artwork. Tessellation based methods on the other hand generate tight non overlapping triangles that confines the path geometry and generate spread around the control triangles to generate extra pixels for antialiasing. This tessllalation process runs on CPU and performance intensive leading to slow editing performance.

GPU Framebuffer States in Render Pass



Frame Buffer state at beginning of draw call

Frame Buffer state after fragment shader execution of coverage fragments of Path P1

Frame Buffer state after fragment shader execution of color fragments of path P1.

Frame Buffer state after fragment shader execution of coverage fragments of path P2.

Frame Buffer state after fragment shader execution of color fragments of path P2.

References

[KSST] KOKOJIMA Y., SUGITA K., SAITO T., TAKEMOTO T.: Resolution independent rendering of deformable vector objects using graphics hardware. SIGGRAPH '06. doi:10.1145/1179849.1179997. 1

[NH] NEHAB D., HOPPE H.: Random-access rendering of general vector graphics. SIGGRAPH '06. doi:10.1145/1409060.1409088. 2

[Ope] URL: http://www.glprogramming.com/red/chapter14.html. 2