Literature Review

David Gow

May 3, 2014

1 Introduction

Since mankind first climbed down from the trees, it is our ability to communicate that has made us unique. Once ideas could be passed from person to person, it made sense to have a permanent record of them; one which could be passed on from person to person without them ever meeting.

And thus the document was born.

Traditionally, documents have been static: just marks on paper, but with the advent of computers many more possibilities open up. Most existing document formats — such as the venerable PostScript and PDF — are, however, designed to imitate existing paper documents, largely to allow for easy printing. In order to truly take advantage of the possibilities operating in the digital domain opens up to us, we must look to new formats.

Formats such as `HTML` allow for a greater scope of interactivity and for a more data-driven model, allowing the content of the document to be explored in ways that perhaps the author had not anticipated.[1] However, these data-driven formats typically do not support fixed layouts, and the display differs from renderer to renderer.

Existing document formats, due to being designed to model paper, have limited precision (8 decimal digits for PostScript[2], 5 decimal digits for PDF[3]). This matches the limited resolution of printers and ink, but is limited when compared to what aught to be possible with "zoom" functionality, which is prevent from working beyond a limited scale factor, lest artefacts appear due to issues with numeric precision.

2 Rendering

Computer graphics comes in two forms: bit-mapped (or raster) graphics, which is defined by an array of pixel colours, and *vector* graphics, defined by mathematical descriptions of objects. Bit-mapped graphics are well suited to photographs and are match how cameras, printers and monitors work. However, bitmap devices do not handle zooming beyond their "native" resolution — the resolution where one document pixel maps to one display pixel —, exhibiting an artefact called pixelation where the pixel structure becomes evident. Attempts to use interpolation to hide this effect are never entirely successful, and sharp edges, such as those found in text and diagrams, are particularly effected.

Vector graphics lack many of these problems: the representation is independent of the output resolution, and rather an abstract description of what it is

being rendered, typically as a combination of simple geometric shapes like lines, arcs and "Bézier curves". As existing displays (and printers) are bit-mapped devices, vector documents must be *rasterized* into a bitmap at a given resolution. This bitmap is then displayed or printed. The resulting bitmap is then an approximation of the vector image at that resolution.

This project will be based around vector graphics, as these properties make it more suited to experimenting with zoom quality.

The rasterization process typically operates on an individual "object" or "shape" at a time: there are special algorithms for rendering lines[4], triangles[5], polygons[6] and Bézier Curves[7]. Typically, these are rasterized independently and composited in the bitmap domain using Porter-Duff compositing[8] into a single image. This allows complex images to be formed from many simple pieces, as well as allowing for layered translucent objects, which would otherwise require the solution of some very complex constructive geometry problems.

While traditionally, rasterization was done entirely in software, modern computers and mobile devices have hardware support for rasterizing some basic primitives — typically lines and triangles —, designed for use rendering 3D scenes. This hardware is usually programmed with an API like `OpenGL`[9].

More complex shapes like Bézier curves can be rendered by combining the use of bitmapped textures (possibly using signed-distance fields[10][11][12]) with polygons approximating the curve's shape[13][14].

GPU Rendering[13], OpenVG implementation on GLES: [15], [16]

Existing implementations of document format rendering

2.1 Xr: Cross-device Rendering for Vector Graphics[17]

Xr (now known as Cairo) is an implementation of the PDF v1.4 rendering model, independent of the PDF or PostScript file formats, and is now widely used as a rendering API. In this paper, Worth and Packard describe the PDF v1.4 rendering model, and their PostScript-derived API for it.

The PDF v1.4 rendering model is based on the original PostScript model, based around a set of *paths* (and other objects, such as raster images) each made up of lines and Bézier curves, which are transformed by the "Current Transformation Matrix." Paths can be *filled* in a number of ways, allowing for different handling of self-intersecting paths, or can have their outlines *stroked*. Furthermore, paths can be painted with RGB colours and/or patterns derived from either previously rendered objects or external raster images. PDF v1.4 extends this to provide, amongst other features, support for layering paths and objects using Porter-Duff compositing[8], giving each painted path the option of having an α value and a choice of any of the Porter-Duff compositing methods.

The Cairo library approximates the rendering of some objects (particularly curved objects such as splines) with a set of polygons. An `XrSetTolerance` function allows the user of the library to set an upper bound on the approximation error in fractions of device pixels, providing a trade-off between rendering quality and performance. The library developers found that setting the tolerance to greater than 0.1 device pixels resulted in errors visible to the user.

2.2 Glitz: Hardware Accelerated Image Compositing using OpenGL[18]

This paper describes the implementation of an `OpenGL` based rendering backend for the `Cairo` library.

The paper describes how OpenGL's Porter-Duff compositing is easily suited to the Cairo/PDF v1.4 rendering model. Similarly, traditional OpenGL (pre-version 3.0 core) support a matrix stack of the same form as Cairo.

The "Glitz" backend will emulate support for tiled, non-power-of-two patterns/textures if the hardware does not support it.

Glitz can render both triangles and trapezoids (which are formed from pairs of triangles). However, it cannot guarantee that the rasterization is pixel-precise, as OpenGL does not proveide this consistently.

Glitz also supports multi-sample anti-aliasing, convolution filters for raster image reads (implemented with shaders).

Performance was much improved over the software rasterization and over XRender accellerated rendering on all except nVidia hardware. However, nVidia's XRender implementation did slow down significantly when some transformations were applied.

Also look at `NV_path_rendering` [19]

3 Floating-Point Precision

How floating-point works and what its behaviour is w/r/t range and precision [20] [21]

Arb. precision exists

Higher precision numeric types can be implemented or used on the GPU, but are slow. [22]

4 Quadtrees

The quadtree is a data structure which [23]

References

[1] Brian Hayes. Pixels or perish. *American Scientist*, 100(2):106 – 111, 2012.

[2] Adobe Systems Incorporated. *PostScript Language Reference*. Addison-Wesley Publishing Company, 3rd edition, 1985 - 1999.

[3] Adobe Systems Incorporated. *PDF Reference*. Adobe Systems Incorporated, 6th edition, 2006.

[4] Jack E Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems journal*, 4(1):25–30, 1965.

[5] Fabien Giesen. Triangle rasterization in practice. `http://fgiesen.wordpress.com/2013/02/08/triangle-rasterization-in-practice/`, 2013.

[6] Juan Pineda. A parallel algorthim for polygon rasterization. *ACM Computer Graphics*, 22(4):17–20, 1988.

[7] Ron Goldman. The fractal nature of bezier curves. The de Casteljau subdivision algorithm is used to show that Bezier curves are also attractors (ie: fractals). A new rendering algorithm is derived for Bezier curves.

[8] Thomas Porter and Tom Duff. Compositing digital images. In *ACM Siggraph Computer Graphics*, volume 18, pages 253–259. ACM, 1984.

[9] Mark Segal, Kurt Akely, and Jon Leech. *The OpenGL® Graphics System: A Specification*. The Kronos Group, Inc, 2014.

[10] F Leymarie and Martin D Levine. Fast raster scan distance propagation on the discrete rectangular lattice. *CVGIP: Image Understanding*, 55(1):84–94, 1992.

[11] Sarah F Frisken, Ronald N Perry, Alyn P Rockwood, and Thouis R Jones. Adaptively sampled distance fields: a general representation of shape for computer graphics. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 249–254. ACM Press/Addison-Wesley Publishing Co., 2000.

[12] Chris Green. Improved alpha-tested magnification for vector textures and special effects. In *ACM SIGGRAPH 2007 courses*, pages 9–18. ACM, 2007.

[13] Charles Loop and Jim Blinn. Resolution independent curve rendering using programmable graphics hardware. *ACM Transactions on Graphics (TOG)*, 24(3):1000–1009, 2005.

[14] Charles Loop and Jim Blinn. Rendering vector art on the gpu. *GPU gems*, 3:543–562, 2007.

[15] Aekyung Oh, Hyunchan Sung, Hwanyong Lee, Kujin Kim, and Nakhoon Baek. Implementation of openvg 1.0 using opengl es. In *Proceedings of the 9th international conference on Human computer interaction with mobile devices and services*, pages 326–328. ACM, 2007.

[16] Mathieu Robart. Openvg paint subsystem over opengl es shaders. In *Consumer Electronics, 2009. ICCE'09. Digest of Technical Papers International Conference on*, pages 1–2. IEEE, 2009.

[17] Carl Worth and Keith Packard. Xr: Cross-device rendering for vector graphics. In *Linux Symposium*, page 480, 2003.

[18] Peter Nilsson and David Reveman. Glitz: Hardware accelerated image compositing using OpenGL. In *USENIX Annual Technical Conference, FREENIX Track*, pages 29–40, 2004.

[19] Mark J Kilgard and Jeff Bolz. Gpu-accelerated path rendering. *ACM Transactions on Graphics (TOG)*, 31(6):172, 2012.

[20] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surv.*, 23(1):5–48, March 1991.

[21] David Goldberg. The design of floating-point data types. *ACM Lett. Program. Lang. Syst.*, 1(2):138–151, June 1992.

[22] Niall Emmart and Charles Weems. High precision integer multiplication with a graphics processing unit. In *2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*, pages 1–6. IEEE, 2010.

[23] Raphael A. Finkel and Jon Louis Bentley. Quad trees a data structure for retrieval on composite keys. *Acta informatica*, 4(1):1–9, 1974.