Literature Review

David Gow

May 19, 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.

2 Document Formats

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.

Ultimately, there are two fundamental stages by which all documents — digital or otherwise — are produced and displayed: *layout* and *display.* The *layout* stage is where the positions and sizes of text and other graphics are determined, while the *display* stage actually produces the final output, whether as ink on paper or pixels on a computer monitor.

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.

3 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].

Indeed, there are several implementations of entire vector graphics systems using OpenGL: OpenVG[15] on top of OpenGL ES[16]; the Cairo[17] library, based around the PostScript/PDF rendering model, has the "Glitz" OpenGL backend[18] and the SVG/PostScript GPU renderer by nVidia[19] as an OpenGL extension[20].

4 Floating-Point Precision

On modern computer architectures, there are two basic number formats supported: fixed-width integers and *floating-point* numbers. Typically, computers natively support integers of up to 64 bits, capable of representing all integers between 0 and $2^{64} - 1$[1].

Floating-point numbers[21] are the binary equivalent of scientific notation: each number consisting of an exponent (e) and a mantissa (m) such that a number is given by

$$n = 2^e \times m \qquad (1)$$

---

[1] Most machines also support *signed* integers, which have the same cardinality as their *unsigned* counterparts, but which represent integers between $-(2^{63})$ and $2^{63} - 1$

The IEEE 754 standard[22] defines several floating-point data types which are used[2] by most computer systems. The standard defines 32-bit (8-bit exponent, 23-bit mantissa) and 64-bit (11-bit exponent, 53-bit mantissa) formats[3]

How floating-point works and what its behaviour is w/r/t range and precision [21] [24]

Arb. precision exists

Higher precision numeric types can be implemented or used on the GPU, but are slow. [25]

5 Quadtrees

When viewing or processing a small part of a large document, it may be helpful to only processs — or *cull* — parts of the document which are not on-screen.

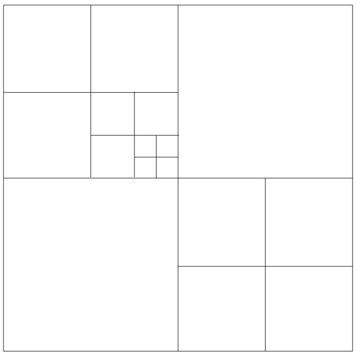

Figure 1: A simple quadtree.

The quadtree[26]is a data structure — one of a family of *spatial* data structures — which recursively breaks down space into smaller subregions which can be processed independently. Points (or other objects) are added to a single node, which if certain criteria are met — typically the number of points in a node exceeding a maximum, though in our case likely the level of precision required exceeding that supported by the data type in use — is split into four equal-sized subregions, and points attached to the region which contains them.

In this project, we will be experimenting with a form of quadtree in which each node has its own independent coordinate system, allowing us to store some spatial information[4] within the quadtree structure, eliminating redundancy in the coordinates of nearby objects.

Other spatial data structures exist, such as the KD-tree[27], which partitions the space on any axis-aligned line; or the BSP tree[28], which splits along an arbitrary line which need not be axis aligned. We believe, however, that the simpler conversion from binary coordinates to the quadtree's binary split make it a better avenue for initial research to explore.

---

[2] Many systems' implement the IEEE 754 standard's storage formats, but do not implement arithmetic operations in accordance with this standard.

[3] The 2008 revision to this standard[23] adds some additional formats, but is less widely supported in hardware.

[4] One bit per-coordinate, per-level of the quadtree

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] 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.

[16] 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.

[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] Mark J Kilgard. Programming with NV path rendering: An annex to the SIGGRAPH paper GPU-accelerated path rendering. *heart*, 300:300.

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

[22] IEEE standard for binary floating-point arithmetic. *ANSI/IEEE Std 754-1985*, 1985.

[23] IEEE standard for floating-point arithmetic. *IEEE Std 754-2008*, pages 1–70, Aug 2008.

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

[25] 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.

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

[27] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975.

[28] Henry Fuchs, Zvi M. Kedem, and Bruce F. Naylor. On visible surface generation by a priori tree structures. In *Proceedings of the 7th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH ’80, pages 124–133, New York, NY, USA, 1980. ACM.