Infinite Precision Document Formats

Author: Samuel Moore[1]
Partners: David Gow[2]
Supervisor: Prof Tim French

THE UNIVERSITY OF
WESTERN AUSTRALIA
Achieve International Excellence

May 2, 2014

Abstract

At the fundamental level, a document is a means to convey information. The limitations on a digital document format therefore restrict the types and quality of information that can be communicated. Whilst modern document formats are now able to include increasingly complex dynamic content, they still suffer from early views of a document as a static page; to be viewed at a fixed scale and position. In this report, we focus on the limitations of modern document formats (including PDF, PostScript, SVG) with regards to the level of detail, or precision at which primatives can be drawn. We propose a research project to investigate whether it is possible to obtain an "infinite precision" document format, capable of including primitives created at an arbitrary level of zoom.

Move to introduction? But it discusses the Introduction :S

In Chapter 1 we give an overview of the current state of the research in document formats, and the motivation for implementing "infinite precision" in a document format. We will outline our approach to research in collaboration with David Gow[]. In Chapter 2 we provide more detailed background examining the literature related to rendering, interpreting, and creating document formats, as well as possible techniques for increased and possibly infinite precision. In Chapter **??** gives the current state of our research and the progress towards the goals outlined in Chapter 1. In Chapter 4 we will conclude with a summary of our findings and goals.

Keywords: *document formats, precision, floating point, graphics, OpenGL, VHDL, PostScript, PDF, bootstraps*

TODO: Make document smaller; currently 16 pages with almost no content; limit is 20 with actual content

Contents

1 Introduction **1**

1.1 Aim . . . 2

1.2 Methods . . . 2

1.3 Software and Hardware Requirements . . . 4

1.4 Timeline . . . 5

2 Literature Review **6**

2.1 Vector Graphics vs Raster Graphics . . . 6

2.2 Primitives in Vector Graphics Formats (and how they are Rendered) . . . 9

2.2.1 Bezier Curves . . . 9

2.2.2 Text . . . 9

2.2.3 Shapes . . . 9

2.2.4 Other Things . . . 9

2.3 Document Representations . . . 9

2.3.1 Interpreted Model . . . 9

2.3.2 Crippled Interpreted Model . . . 10

2.3.3 Document Object Model . . . 10

2.3.4 Blurring the Line — Javascript . . . 11

2.3.5 Why do we still use static PDFs . . . 11

2.4 Precision in Modern Document Formats . . . 12

2.5 Representation of Numbers . . . 12

2.5.1 The IEEE Standard . . . 13

2.5.2 Floating Point Number Representations . . . 13

2.5.3 Limitations Imposed By CPU . . . 13

2.5.4 Limitations Imposed By Graphics APIs and/or GPUs . . . 13

2.5.5 Examples of Precision Related Errors in Floating Point Arithmetic . . . 14

2.5.6 Relate This to the Sorts of Maths Done By Document Formats . . . 14

2.5.7 Techniques for Arbitrary Precision Arithmetic . . . 14

2.5.8 Alternate Number Representations 14

3 Progress Report **15**

3.1 Development of Testbed Software 15

3.2 Design and Implementation of “Tests” 15

3.3 Document Format 15

3.4 Floating Point Number Representations 16

3.5 Virtual FPU 16

3.6 Version Control 16

4 Conclusion **17**

4.1 Acheived Milestones 17

4.2 Areas of further work 17

4.3 Witty Conclusion Goes Here 17

References **19**

Chapter 1

Introduction

Most of this chapter is copy pasted from the project proposal
⟨http://szmoore.net/ipdf/documents/ProjectProposalSam.pdf⟩

Early electronic document formats such as PostScript were motivated by a need to print documents onto a paper medium. In the PostScript standard, this lead to a model of the document as a program; a series of instructions to be executed by an interpreter which would result in "ink" being placed on "pages" of a fixed size[3]. The ubiquitous Portable Document Format (PDF) standard provides many enhancements to PostScript taking into account desktop publishing requirements[4], but it is still fundamentally based on the same imaging model[5]. This idea of a document as a static "page" has lead to limited precision in these and other traditional document formats.

The emergence of the internet, web browsers, XML/HTML, JavaScript and related technologies has seen a revolution in the ways in which information can be presented digitally, and the PDF standard itself has begun to move beyond static text and figures[6, 7]. However, the popular document formats are still designed with the intention of showing information at either a single, fixed level of detail, or a small range of levels.

As most digital display devices are smaller than physical paper medium, all useful viewers are able to "zoom" to a subset of the document. Vector graphics formats including PostScript and PDF support rasterisation at different zoom levels[3, 5], but the use of fixed precision floating point numbers causes problems due to imprecision either far from the origin, or at a high level of detail[8, 9].

We are now seeing a widespread use of mobile computing devices with touch screens, where the display size is typically much smaller than paper pages and traditional computer monitors; it seems that there is much to be gained by breaking free of the restricted precision of traditional document formats.

1.1 Aim

In this project, we will explore the state of the art of current document formats including PDF, PostScript, SVG, HTML, and the limitations of each in terms of precision. We will consider designs for a document format allowing graphics primitives at an arbitrary level of zoom with no loss of detail. We will refer to such a document format as "infinite precision". A viewer and editor will be implemented as a proof of concept; we adopt a low level, ground up approach to designing this viewer so as to not become restricted by any single existing document format.

New bit

It is necessary to clarify what we mean by "infinite" precision. We do not propose to be able to contain an infinite amount of information within a document. The goal is to be able to render a primitive at the same level of detail it is specified by a document format. For example, the precision of coordinates of primitives drawn in a graphical editor will always be limited by the resolution of the display on which they are drawn, but not by the viewer.

There are many possible applications for documents in which precision is unlimited. Several areas of use include: visualisation of extremely large or infinite data sets; visualisation of high precision numerical computations; digital artwork; computer aided design; and maps.

1.2 Methods

Initial research and software development is being conducted in collaboration with David Gow[2]. Once a simple testbed application has been developed, we will individually explore approaches for introducing arbitrary levels of precision; these approaches will be implemented as alternate versions of the same software. The focus will be on drawing simple primitives (lines, polygons, circles). However, if time permits we will explore adding more complicated primitives (font glyphs, bezier curves, embedded bitmaps).

At this stage we have identified two possible areas for individual research:

1. **Arbitrary Precision real valued numbers** — Sam Moore

   We plan to investigate the representation of real values to a high or arbitary degree of precision. Such representations would allow for a document to be implemented using a single global coordinate system. However, we would expect a decrease in performance with increased complexity of the data structure used to represent a real value. Both software and hardware techniques will be explored. We will also consider the limitations imposed by performing calculations on the GPU or CPU.

Starting points for research in this area are Priest's 1991 paper, "Algorithms for Arbitrary Precision Floating Point Arithmetic"[10], and Goldberg's 1992 paper "The design of floating point data types"[9]. A more recent and comprehensive text book, "Handbook of Floating Point Arithmetic"[11], published in 2010, has also been identified as highly relevant.

2. **Local coordinate systems** — David Gow [2]

   An alternative approach involves segmenting the document into different regions using fixed precision floats to define primitives within each region. A quadtree or similar data structure could be employed to identify and render those regions currently visible in the document viewer. Say more here?

We aim to compare these and any additional implementations considered using the following metrics:

1. **Performance vs Number of Primitives**

   As it is clearly desirable to include more objects in a document, this is a natural metric for the usefulness of an implementation. We will compare the performance of rendering different implementations, using several "standard" test documents.

2. **Performance vs Visible Primitives**

   There will inevitably be an overhead to all primitives in the document, whether drawn or not. As the structure of the document format and rendering algorithms may be designed independently, we will repeat the above tests considering only the number of visible primitives.

3. **Performance vs Zoom Level**

   We will also consider the performance of rendering at zoom levels that include primitives on both small and large scales, since these are the cases under which floating point precision causes problems in the PostScript and PDF standards.

4. **Performance whilst translation and scaling**

   Whilst changing the view, it is ideal that the document be re-rendered as efficiently as possible, to avoid disorienting and confusing the user. We will therefore compare the speed of rendering as the standard documents are translated or scaled at a constant rate.

5. **Artifacts and Limitations on Precision**

   As we are unlikely to achieve truly "infinite" precision, qualitative comparisons of the accuracy of rendering under different implementations should be made.

1.3 Software and Hardware Requirements

Due to the relative immaturity and inconsistency of graphics drivers on mobile devices, our proof of concept will be developed for a conventional GNU/Linux desktop or laptop computer using OpenGL. However, the techniques explored could easily be extended to other platforms and libraries.

1.4 Timeline

Deadlines enforced by the faculty of Engineering Computing and Mathematics are *italicised.*[1].

Date	**Milestone**
17th April	Draft Literature Review completed. This sort of didn't happen...
1st May	Testbed Software (basic document format and viewer) completed and approaches for extending to allow infinite precision identified.
26th May	*Progress Report and Revised Literature Review due.*
9th June	Demonstrations of limitations of floating point precision in the Testbed software.
1st July	At least one implementation of infinite precision for basic primitives (lines, polygons, curves) completed. Other implementations, advanced features, and areas for more detailed research identified.
1st August	Experiments and comparison of various infinite precision implementations completed.
1st September	Advanced features implemented and tested, work underway on Final Report.
TBA	*Conference Abstract and Presentation due.*
10th October	*Draft of Final Report due.*
27th October	*Final Report due.*

[1] David Gow is being assessed under the 2014 rules for a BEng (Software) Final Year Project, whilst the author is being assessed under the 2014 rules for a BEng (Mechatronics) Final Year Project; deadlines and requirements as shown in Gow's proposal[2] may differ

Chapter 2

Literature Review

This chapter will review the literature. It will also include some figures created by us from our test programs to aid with conceptual understanding of the literature. A paper by paper summary of the literature is also available at:
⟨http://szmoore.net/ipdf/documents/LiteratureNotes.pdf⟩.

TODO: If I want to link to the Paper by Paper summary it will need a bit of rewriting.

TODO: Actually (re)write this entire chapter.

TODO: Un dot point ify

TODO: Citations

TODO: Make less terrible

TODO: Reconsider sections (do I really want to make this go down to `\subsubsection`?)

TODO: :-(

2.1 Vector Graphics vs Raster Graphics

TODO: Distinguish between Raster Formats and the Rasterisation of an image (which may or may not be in a raster format)

Raster Graphics

- Bitmap — array of colour information for pixels
- Exact pixels in a similar format to how they would appear on a (modern) display device.

  - Also similar to how they would be stored by a camera or scanner
  - Is it misleading to say 2D array? Pixels are actually stored in a 1D array, but conceptually it's nicer to say 2D
  - For that matter, should it described as 3D (3rd dimension = colour)?

- Lowest level representation of a document
- Issues with scaling; values of extra pixels must be calculated
- Not convenient to edit; ill suited to text

Vector Graphics

- Stores relative position of primitives - scales better
- In particular, *edges* of lines can be zoomed without becomming jagged; sometimes (somewhat misleadingly) described as "infinitely sharp"
- Vector Graphics must be rasterised before being drawn on most display devices.
- Still can't scale forever due to use of fixed size floats

Resolution and Raster Graphics

- DPI = dots (pixels) per inch differs per display device - a rastered image looks different on different display devices
- PostScript/PDF use 72 points per inch; this means a rasterised image will look the same in all pdf viewers regardless of the display.
- Tex uses 72.27 points per inch (?)
- The vector image was rastered at 96 points per inch
- Hence, have to scale by 72.27/96 = 0.7528125 to get the vector and rastered version to look exactly the same in the pdf

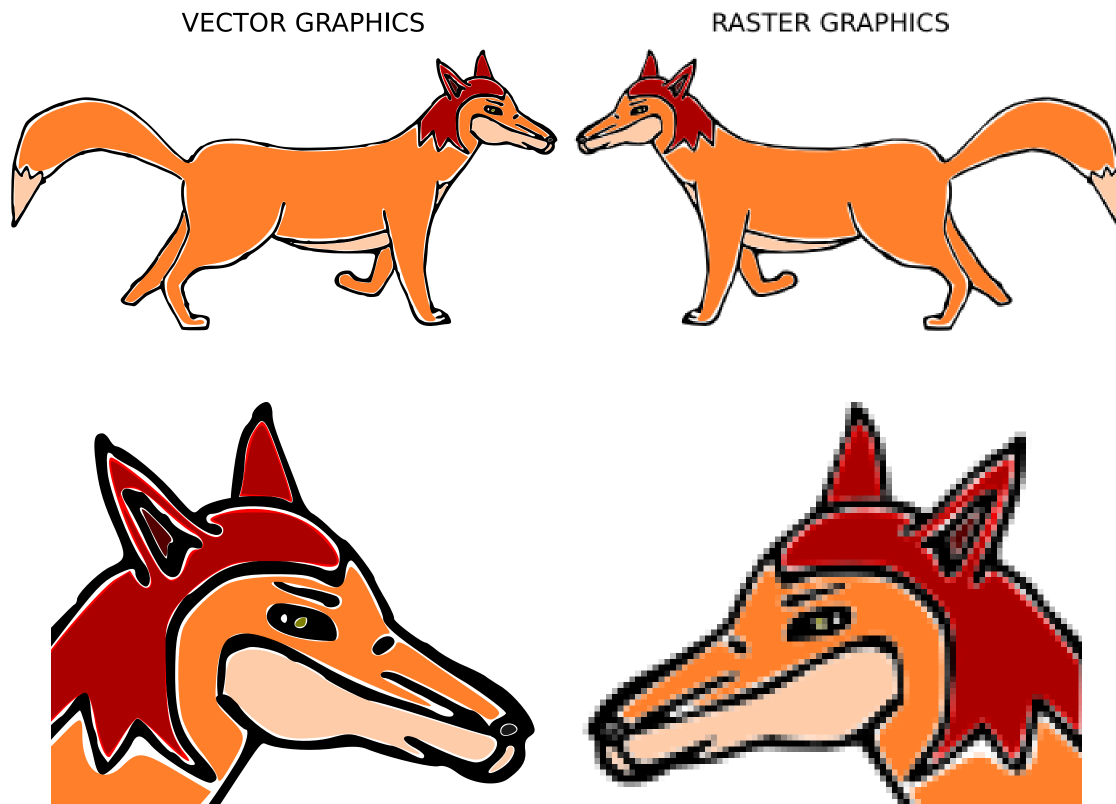

Figure 2.1: Scaling of vector and Raster Graphics

Figure 2.1 shows a vector image (left) which has been rasterised (right). At the original scale the two foxes should appear to be mirror images[1]. When the scale is increased, the edges of the vector image remain sharp, whilst the raster image begins to appear jagged. PDF viewers will typically use antialiasing to smooth the edges of a scaled bitmap, causing the image to appear blurred.[2].

Various ways to end this section:

1. It should be obvious that documents containing text must use the vector graphics format, and so the remainder of this chapter will concentrate on the latter.
2. As can be seen in Figure **??**, if we were to decide to pursue "infinite precision" in raster graphics we would be shooting ourselves in both feet and then the face before we even started. The rest of this chapter will concentrate on vector graphics.
3. You can't have infinite precision in raster graphics by definition, therefore we no longer care about them in this report.
4. This report being in a vector format is a clue that we only care about vector formats.

[1] If I've worked out the scaling to account for dpi differences between inkscape and latex/pdf correctly

[2] In the Atril Document Viewer 1.6.0 this image will only be antialiased at zoom levels $\leq 125\%$

2.2 Primitives in Vector Graphics Formats (and how they are Rendered)

2.2.1 Bezier Curves

I did an ipython notebook on this in February, but I forgot all of it

2.2.2 Text

Text is just Bezier Curves

2.2.3 Shapes

Shapes are just bezier curves joined together.

2.2.4 Other Things

We don't really care about other things (ie: Colour gradients etc) in this report.

2.3 Document Representations

The file format can be either human readable[3] or binary[4]. Can also be compressed or not. Here we are interested in how the document is interpreted or traversed in order to produce graphics output.

2.3.1 Interpreted Model

Did I just invent that terminology or did I read it in a paper? Is there actually existing terminology for this that sounds similar enough to "Document Object Model" for me to compare them side by side?

- This model treats a document as the source code program which produces graphics
- Arose from the desire to produce printed documents using computers (which were still limited to text only displays).
- Typed by hand or (later) generated by a GUI program

[3] For some definition of human and some definition of readable
[4] So, our viewer is basically a DOM style but stored in a binary format

- PostScript — largely supersceded by PDF on the desktop but still used by printers[5]
- TEX— Predates PostScript! LATEX is being used to create this very document and until now I didn't even have it here!
  - I don't really want to go down the path of investigating the billion steps involved in getting LATEXinto an actually viewable format
  - There are interpreters (usually WYSIWYG editors) for LATEXthough
  - Maybe if LATEXwere more popular there would be desktop viewers that converted LATEXdirectly into graphics
- Potential for dynamic content, interactivity; dynamic PostScript, enhanced Postscript
- Scientific Computing — Mathematica, Matlab, IPython Notebook — The document and the code that produces it are stored together
- Problems with security — Turing complete, can be exploited easily

2.3.2 Crippled Interpreted Model

I'm pretty sure I made that one up

- PDF is PostScript but without the Turing Completeness
- Solves security issues, more efficient

2.3.3 Document Object Model

- DOM = Tree of nodes; node may have attributes, children, data
- XML (SGML) is the standard language used to represent documents in the DOM
- XML is plain text
- SVG is a standard for a vector graphics language conforming to XML (ie: a DOM format)
- CSS style sheets represent more complicated styling on objects in the DOM

[5]Desktop pdf viewers can still cope with PS, but I wonder if a smartphone pdf viewer would implement it?

2.3.4 Blurring the Line — Javascript

- The document is expressed in DOM format using XML/HTML/SVG
- A Javascript program is run which can modify the DOM
- At a high level this may be simply changing attributes of elements dynamically
- For low level control there is canvas2D and even WebGL which gives direct access to OpenGL functions
- Javascript can be used to make a HTML/SVG interactive
  - Overlooking the fact that the SVG standard already allows for interactive elements...
- Javascript is now becoming used even in desktop environments and programs (Windows 8, GNOME 3, Cinnamon, Game Maker Studio) (**shudder**)
- There are also a range of papers about including Javascript in PDF "Pixels or Perish" being the only one we have actually read[6]
  - I have no idea how this works; PDF is based on PostScript... it seems very circular to be using a programming language to modify a document that is modelled on being a (non turing complete) program
  - This is yet more proof that people will converge towards solutions that "work" rather than those that are optimal or elegant
  - I guess it's too much effort to make HTML look like PDF (or vice versa) so we could phase one out

2.3.5 Why do we still use static PDFs

Despite their limitations, we still use static, boring old PDFs. Particularly in scientific communication.

- They are portable; you can write an amazing document in Mathematica/Matlab but it
- Scientific journals would need to adapt to other formats and this is not worth the effort
- No network connection is required to view a PDF (although DRM might change this?)
- All rescources are stored in a single file; a website is stored accross many seperate files (call this a "distributed" document format?)

- You can create PDFs easily using desktop processing WYSIWYG editors; WYSIWYG editors for web based documents are worthless due to the more complex content
- Until Javascript becomes part of the PDF standard, including Javascript in PDF documents will not become widespread
- Once you complicate a PDF by adding Javascript, it becomes more complicated to create; it is simply easier to use a series of static figures than to embed a shader in your document. Even for people that know WebGL.

2.4 Precision in Modern Document Formats

All this is very interesting and provides important context, but it is not actually directly related to the problem of infinite precision which we are going to try and solve.

- Implementations of PostScript and PDF must by definition restrict themselves to IEEE binary32 "single precision"[6] floating point number representations in order to conform to the standards[3, 5].
- Implementations of SVG are by definition required to use IEEE binary32 as a **minimum**. "High Quality" SVG viewers are required to use at least IEEE binary64.[12]
- Numerical computation packages such as Mathematica and Maple use arbitrary precision floats
  - Mathematica is not open source which is an issue when publishing scientific research (because people who do not fork out money for Mathematica cannot verify results)
  - What about Maple? [11] and [13] both mention it being buggy.
  - Octave and Matlab use fixed precision doubles

The use of IEEE binary32 floats in the PostScript and PDF standards is not surprising if we consider that these documents are oriented towards representing static pages. They don't actually need higher precision to do this; 32 bits is more than sufficient for A4 paper.

2.5 Representation of Numbers

Although this project has been motivated by a desire for more flexible document formats, the fundamental source of limited precision in vector document formats is the restriction to IEEE floating point numbers for representation of coordinates.

[6] The original IEEE-754 defined single, double and extended precisions; in the revision these were renamed to binary32, binary64 and binary128 to explicitly state the base and number of bits

Whilst David Gow will be focusing on structures and the use of multiple coordinate systems to represent a document so as to avoid or reduce these limitations[2], the focus of our own research will be increased precision in the representation of real numbers so as to get away with using a single global coordinate system.

2.5.1 The IEEE Standard

2.5.2 Floating Point Number Representations

$$x = (-1)^s \times m \times B^e$$

$B = 2$, although IEEE also defines decimal representations for $B = 10$ — these are useful in financial software[14].

Aside: Are decimal representations for a document format eg: CAD also useful because you can then use metric coordinate systems?

Precision

The floats map an infinite set of real numbers onto a discrete set of representations.

Figure: 8 bit "minifloats" (all 255 of them) clearly showing the "precision vs range" issue

The most a result can be rounded in conversion to a floating point number is the units in last place; $m_N \times B^e$.

Even though that paper that claims double is the best you will ever need because the error can be as much as the size of a bacterium relative to the distance to the moon[] there are many cases where increased number of bits will not save you.[11]

2.5.3 Limitations Imposed By CPU

CPU's are restricted in their representation of floating point numbers by the IEEE standard.

2.5.4 Limitations Imposed By Graphics APIs and/or GPUs

Traditionally algorithms for drawing vector graphics are performed on the CPU; the image is rasterised and then sent to the GPU for rendering[]. Recently there has been a great deal of literature relating to implementation of algorithms such as bezier curve rendering[]

or shading[] on the GPU. As it seems the trend is to move towards GPU

- Internal representations are GPU dependent and may not match IEEE[15]
- OpenGL standards specify: binary16, binary32, binary64
- OpenVG aims to become a standard API for SVG viewers but the API only uses binary32 and hardware implementations may use less than this internally[16]
- It seems that IEEE has not been entirely successful; although all modern CPUs and GPUs are able to read and write IEEE floating point types, many do not conform to the IEEE standard in how they represent floating point numbers internally.

AND THUS WE FINALLY GET TO THE POINT

2.5.5 Examples of Precision Related Errors in Floating Point Arithmetic

2.5.6 Relate This to the Sorts of Maths Done By Document Formats

2.5.7 Techniques for Arbitrary Precision Arithmetic

- Fast2SUM for summation (and multiplication).
- Guard digits.
- Other techniques
- Hardware techniques that improve speed (which may be beneficial because you can get away with higher precision in hardware)
- Anything you can do in hardware you can do in software but it will be slower and have more segmentation faults

2.5.8 Alternate Number Representations

They exist[11].

Chapter 3

Progress Report

This chapter outlines the current state of our research in relation to the aims outlined in Chapter 1. It will serve as an explanation for where the Figures in Chapter 2 came from. It will just be a short summary of the implementation details.

3.1 Development of Testbed Software

We wrote a very simple OpenGL 1.1 program to experiment with, and then David Gow converted it to OpenGL 3.1 and I have no idea how it works anymore.

3.2 Design and Implementation of "Tests"

- Compile by swapping out `main()` for a tester
- There are tests for doing some of the things in Chapter 1 but most still aren't written yet.

3.3 Document Format

Currently we effectively have a DOM format but with the following non-features:

- Binary file format (non standard; not XML)
- Only rectangles.

3.4 Floating Point Number Representations

I have[1] some figures that I would prefer to include in Chapter 2 when I am talking about the papers that inspired them. This section will probably briefly talk about how they were created and just refer back to them.

- `calculatepi.test`
- `typedef` of `Real`

3.5 Virtual FPU

Techniques for dealing with FP numbers can be implemented in software (CPU) or on dedicated hardware (FPU). We are able to run FP arithmetic on arbitrary simulations of FPUs created using VHDL. Hopefully explore this a bit in Chapter 2.

3.6 Version Control

Git is a distributed version control system widely used in the development of open source software[]. All rescources created for or used by this project have been placed in git repositories on several servers. The repositories are publically accessable at ⟨http://git.ucc.asn.au⟩

[1] Ok... "will have"

Chapter 4

Conclusion

This report has provided motivation for considering approaches to achieving an infinite level of zoom in a document.

4.1 Acheived Milestones

4.2 Areas of further work

- Continue looking for relevant literature
- Implement all those tests mentioned in Chapter 1
- Actually identify the techniques I will use **THIS ONE SHOULD BE DONE BEFORE I HAND IN THE LITERATURE REVIEW!**
- Possible Ultimate Goal: Implement (a subset) of SVG and then show an SVG document that we can render but a browser can't
  - This means extending our viewer to be able to read (a subset) SVG
  - Can already read XML, so this shouldn't actually be too bad
    - Emphasis on **subset**
    - (I've seen the SVG standard; I'm talking about implementing the 18 pages under "Basic Shapes". The other 818 pages can complain to someone who cares.)
  - Suggestion to David that he probably won't like (or read): Make his octree structure specifiable as an SVG extension

4.3 Witty Conclusion Goes Here

References

[1] Sam Moore. Infinite precision document formats (project proposal). ⟨http://szmoore.net/ipdf/documents/ProjectProposalSam.pdf⟩, 2014.

[2] David Gow. Infinite-precision document formats (project proposal). ⟨http://davidgow.net/stuff/ProjectProposal.pdf⟩, 2014.

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

[4] Michael A. Wan-Lee Cheng. Portable document format (pdf) – finally, a universal document exchange technology. *Journal of Technology Studies*, 28(1):59 – 63, 2002.

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

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

[7] David G. Barnes, Michail Vidiassov, Bernhard Ruthensteiner, Christopher J. Fluke, Michelle R. Quayle, and Colin R. McHenry. Embedding and publishing interactive, 3-dimensional, scientific figures in portable document format (pdf) files. *PLoS ONE*, 8(9):1 – 15, 2013.

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

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

[10] D.M. Priest. Algorithms for arbitrary precision floating point arithmetic. In *Computer Arithmetic, 1991. Proceedings., 10th IEEE Symposium on*, pages 132–143, Jun 1991.

[11] Jean-Michel Muller, Nicolas Brisebarre, Florent de Dinechin, Claude-Pierre Jeannerod, Vincent Lefèvre, Guillaume Melquiond, Nathalie Revol, Damien Stehlé, and Serge Torres. *Handbook of Floating-Point Arithmetic*. Birkhäuser Boston Inc., Cambridge, MA, USA, 2010.

[12] Erik Dahlstóm, Patric Dengler, Anthony Grasso, Chris Lilley, Cameron McCormack, Doug Schepers, Jonathon Watt, Jon Ferraiolo, Fujisawa Jun, and Dean Jackson. Scalable vector graphics (svg) 1.1 (second edition). *WC3 Recommendation*, August 2011.

[13] Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, and Paul Zimmermann. Mpfr: A multiple-precision binary floating-point library with correct rounding. *ACM Trans. Math. Softw.*, 33(2), June 2007.

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

[15] Karl E Hillesland and Anselmo Lastra. Gpu floating-point paranoia. *Proceedings of GP 2004*, 2004.

[16] Daniel Rice and RJ Simpson. Openvg specification, version 1.1. *Khronos Group*, 2008.