Stylish and scaleable – vector graphics for all on the web

Simon Ruffle and Paul Richens
Digital Studio, Department of Architecture
University of Cambridge
1 Benet Place
Lensfield Road
Cambridge, CB2 1EL, UK.
sjr56@cam.ac.uk
pnr12@cam.ac.uk

Abstract

Raster graphics are ubiquitous on the web, but many architectural and engineering applications would be better served by vector techniques. Scalable Vector Graphics (SVG) is an emerging XML based vector graphics standard from the World Wide Web Consortium, but is not yet implemented in mainstream browsers. We describe a way of using the widely distributed Macromedia Flash Player to display SVG files. The resulting drawings are easy to rescale and restyle within the browser, offer superior printing, and many possibilities for advanced interaction and animation.

1. INTRODUCTION

Over the past few years we have been developing and maintaining a Planning and Building Web Site for the University of Cambridge [1], which provides over a hundred pages of information, news, maps, drawings, and photographs about the University's building and urban planning programme and gives an opportunity for consultation on-line about the University's proposals [2]. It is a public web site aimed primarily, but not exclusively, at members of the University and has been live since July 2000, serving about 10,000 pages per month. Conceived originally as a vehicle for research into web-based architectural communications, and funded by an EPSRC Research Grant [3], the site has proved so valuable that the University itself is funding its continuing maintenance and development.

A large part of the project involved taking models, drawings and images from architects' and engineers' offices and making them accessible to the public on the web. A problem for web site producers who are aiming at a general public audience is that they have no control over what equipment their readers are using in terms of computer, web browser software and internet connection bandwidth. If the "lowest common denominator" is chosen, then the web site will be restricted to little more than text. Experience shows it is safe to assume a "middle ground" equipment set-up. For the first two years of the project we assumed our

readers would be able to display in-line raster images in GIF and JPG formats within web pages programmed with Dynamic HTML and JavaScript, but assumed no browser plug-ins were available. We have had very few complaints and the web site has served out over two million files to date. Recently, however, we have begun to assume that our “middle ground” set-up will have a stock of the more common plug-ins such as Adobe Reader and Macromedia Flash. This has led to some reconsideration of the best way to handle architectural drawings.

There are two main techniques for processing graphics in a computer. *Vector* graphics are stored as point coordinates joined by lines and curves, while *raster* graphics are stored as a regular grid of pixel colour values. Among popular applications, Adobe Illustrator works predominately with vector graphics, while Photoshop uses rasters. Computer-aided Design (CAD) and Geographic Information (GIS) systems used by architects, engineers, planners, archaeologists and geographers use vector graphics. On the other hand, raster is ubiquitous on the World Wide Web. Raster graphics are essential for images such as photographs, but vector graphics have a number of advantages for line drawings; the files are smaller, they retain structural information on what is connected to what, they can be displayed at any scale without loss of quality, and the display styles (such as colour and thickness of line) can be separated from the graphics and readily altered. Though vectors can be converted to raster fairly easily the process freezes the scale and style, and destroys the structural information. Originally we thought this did not matter, but the more we have worked with drawings and maps on web pages the more we have realised the need to display vector graphics on web pages using a vector technique.

2. PUBLISHING DRAWINGS ON THE WEB

When consultants provided us with information we found the most common source format was the AutoCAD DXF (Drawing Exchange Format) file [4]. Other formats occasionally encountered were DWG (AutoCAD native), PDF (Adobe Portable Document) and EPS (Encapsulated Postscript) files. These are all predominantly vector formats. Adobe Illustrator [5] proved to be an important piece of software because it can import all these formats, perform a vector to raster conversion whilst retaining layering information, and export into GIF format, though a better result is usually achieved if the drawing is exported to Adobe

Photoshop [6], which can be used to “finish off” the drawing and export it to a wider range of raster formats (figure 1 and [7]).

There is wide support across web browsers for the JPG and GIF raster formats. These formats are compressed, in that the file size has been reduced to improve download speeds. Due to differences in compression algorithms, in general GIF is preferable for diagrams, maps and drawings, and JPG is preferable for photographs. GIF has an additional advantage in that images can contain transparent regions. Using this technique in conjunction with the HTML tag, layering of graphics on the web page is possible. We have used this to great effect on the Planning and Building Web Site, producing drawings where layering on the web pages matches the layering of the original CAD drawing, and we can offer the reader the ability to switch layers on and off to enhance their understanding of the drawing [8].

However, there are problems with raster drawings. To fit a drawing on a web page without it having to be scrolled, it can be no wider than about 1000 pixels. In practice, given that the web page will also contain menus and explanatory text, the drawing may be only 600 pixels wide. This low resolution makes it difficult to read fine detail and text and is particularly noticeable when the drawing is printed – a modern laser printer can resolve 1200 pixels per inch. Clicking on the drawing, for example to obtain more information about a particular building drawn on a site plan, is difficult to achieve because the structural information about the perimeter of the building has been lost in the vector to raster conversion. The usual way round this is to employ an HTML technique called an image map where a separate vector description is supplied via the <map> tag, which creates clickable shapes on the drawing that act as links. We have found image maps cumbersome to make and maintain.

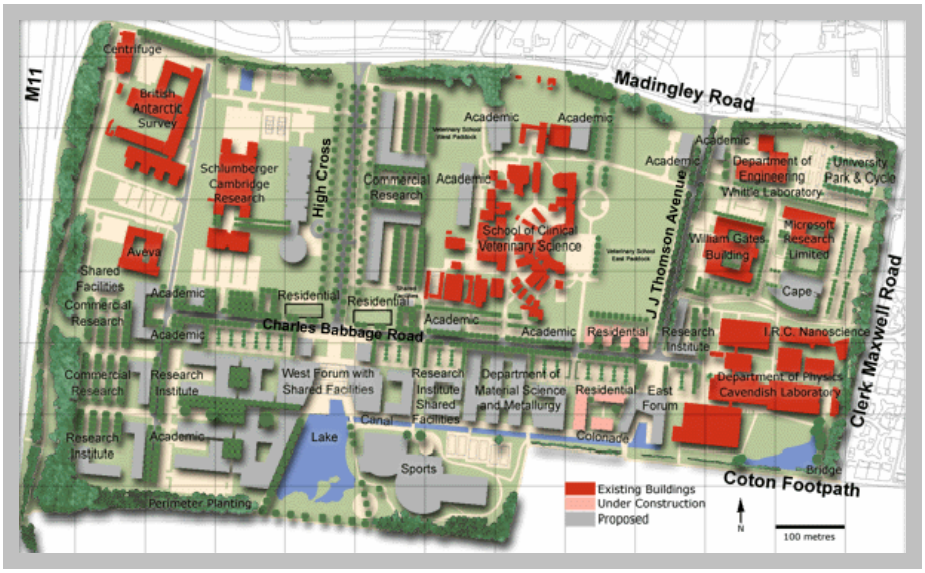

Figure 1: West Cambridge Master Plan as Raster Image on Web Page

As the Planning and Building web site project progressed we began to appreciate the problem of responding to change. We would publish a drawing, only to find that three months later the original had been revised by the consultants. The problem with raster representation of a vector source is that any change to the source material, however small, requires re-running the conversion process in its entirety. As we were making stylistic changes, such as adding drop shadows and colour filling polygons by hand, this became a considerable task. What was required was a technique akin to the mark-up and cascading style sheets of the normal HTML web page, where content and style are separated allowing them to be changed independently

This implies that the source material should be held in vector format, particularly so when graphics are served from a database and there is no opportunity for manual intervention. The conversion from vector to raster could in principle be done either in the server, or in the browser. Conversion in the browser will often reduce network traffic, and offers better zooming and printing behaviour. Because the structural information is made available to the client, advanced interactive behaviours such as roll-overs, visibility switching, dynamic styling and animated effects all become conceivable.

3. EXISTING TECHNIQUES

A number of techniques for displaying vector graphics on web pages have emerged over the last few years:

- Industry web file formats, such as DWF [9] and WebCGM [10], require viewers which are not very well supported by browsers, and are unlikely to be installed on a reader's computer. They may offer the best solution to professional users, but our interest here is in the general public.
- VML (Vector Mark-up Language) is an XML based proposal for a vector graphics standard [11] and is supported from Internet Explorer 5 onwards. It is not widely used at present, possibly because it is closely associated with Microsoft.
- SVG (Scaleable Vector Graphics) is an XML based proposal for a vector graphics standard for the web and mobile computing devices, which has the potential to become the standard for web delivered vector graphics. SVG 1.1 [12] is the current W3C Recommendation. It requires a plug-in viewer (one is available from Adobe [13]).
- Macromedia Flash uses a proprietary, yet published, binary format (SWF). Flash files are displayed by a plug-in viewer - the "Flash Player" [14].
- Adobe Portable Document Format uses a proprietary binary format (PDF). PDF files are displayed by a plug-in viewer – the "Adobe Reader" [15].

To assist in our evaluation of these alternatives, two main criteria were applied. First, the readers' "browsing experience" should not be interrupted by the inconvenience of having to wrestle with installing a plug-in before the vector drawing can be seen. If a plug-in is required, it should be a widely used one, so that it is likely that it is already installed, and be available on a wide range of browsers and platforms, including at a minimum the latest versions of Internet Explorer and Netscape on Windows and Macintosh. Second, it was essential that the technique should allow content and style to be separated allowing them to be changed independently.

Unfortunately none of the techniques above meet with these criteria. Only VML and SVG allow content and style to be separated, but neither of these is widely supported on browsers: VML is available in Internet Explorer only, and Adobe's SVG viewer is not available on the recent versions of Netscape [16]. We were attracted to SVG, nevertheless, given its status as a potential standard for vector graphics on the web and mobile computing devices. To us, the ideal would be a combination of the open SVG standard and the high level of animation, interactivity and presentational quality implemented in the ubiquitous Macromedia

Flash Player. So we set about developing a proof of concept to see if this was possible within the application domain of our Planning and Building web site.

There are two possible approaches to displaying an SVG file in the Flash Player. A server process could convert the file to the Flash SWF binary format, which is published by Macromedia [16]. Alternatively the SVG file could be processed directly in the browser, by the Flash plug-in running an ActionScript program[18]. The second approach has great advantages in integrating SVG graphics with the rest of the Flash MX Development Environment. Experienced Flash developers, of which there is a sizable community, would be able to add SVG vector graphics to their collection of building blocks.

4. CURRENT RESEARCH

Triolo [19] has developed techniques for displaying SVG graphics from within ActionScript. Although she has only implemented a few SVG elements her work shows how SVG graphics can be integrated into the philosophy of Flash. For example, her animated toucan (figure 2, and [20]), loaded from an SVG file, can slowly draw itself in an attractive effect. She shows how another Flash movie, of a butterfly flapping its wings, can "fly" along the vector paths of the SVG graphics. She has conceived a programming model where the SVG drawing once imported becomes a "MovieClip" object, the basic animated building block of the ActionScript language. She uses the built-in XML commands of ActionScript to load and parse the SVG file.

Figure 2: The SVG Toucan.

Triolo publishes sample code [20]. For application to the kind of “typical” drawings we wanted to display on the Planning and Building Web Site, her code suggested a number of opportunities for further research and development:

- Only the <svg>, <g> and <path> elements from the SVG standard are implemented.
- Style information is derived from in-line attributes, not an external style file.
- No functions are provided to pan and zoom the image.
- Processing speed is too slow for our typical drawings. This is partly due to two in-memory data structures being stored – one the parsed XML representation, the other a display list of drawing commands. The double processing increases drawing time and additionally risks problems with shortage of memory. Slow processing in ActionScript presents a particularly unfortunate problem in that after a (non-programmable) time executing a script body the Flash Player displays a timeout message asking if the user wants to abort the script.

5. SVGCLASS

We developed an ActionScript class called “SVGclass”. It allows Flash developers to instantiate an object, by dragging from the Library or using the attachMovie command, which will display an SVG vector graphic file on the Flash stage. Various properties can be set from within ActionScript, including source URL, size on screen, default scale and style file. Methods include zoom, pan and print. Individual layers of the drawing are programmatically accessible so, for example, they can be switched on and off.

The main design issue was the choice of a subset of SVG to implement. The SVG standard as a whole is far too complex for a proof-of-concept research project, yet we wanted a subset that was adequate for the kind of drawings found on our Planning and Building Web Site. W3C publish two profiles (subsets) of SVG – SVG Tiny and SVG Basic [21] – however Basic is still too large and Tiny does not implement style sheets. Adobe Illustrator is our key tool for the manipulation of vector graphics, with its ability to import a range of alternative vector standards and an SVG export facility. We decided to use Illustrator as our definitive source of SVG files, and implement compatibility with the current version, Illustrator CS, whilst maintaining compatibility with the two earlier versions Illustrator 9 and 10 (which employ a markedly different subset of SVG). We additionally implemented simple support for <a> (clickable link) and

<image> elements. The elements and attributes implemented in SVGClass are shown in Table 1, though there are restrictions in the scope of implementation (for details see notes in the sample code).

Table 1: SVG Elements implemented in SVGClass. Elements marked * support in addition style, class, fill, stroke, stroke-width, font-family, font-style, font-size, font-weight, and opacity attributes.

Elements	Supported Attributes
<svg>	width, height
<g>	id, display
<line>*	x1, y1, x2, y2
<polyline>*	points
<polygon>*	points
<rectangle>*	x, y, height, width
<circle>*	cx, cy, r
<ellipse>*	cx, cy, rx, ry
<path>*	d
<text>*	x, y, transform
<tspan>*	x, y
<a>	xlink:href, xlink:target
<image>	xlink:href, x, y

SVGClass was coded in ActionScript, using the Flash MX Development Environment. It inherits from a base class we developed called “xmlLoaderClass” that itself inherits from the MovieClip class, which is the basic animated building block of ActionScript. xmlLoaderClass provides basic functionality to do with loading an XML file, and could be used as a basis for a variety of XML driven Flash effects. It creates a clipped frame on the “stage” in which XML-driven graphics are drawn, it loads the XML file and optional style file asynchronously, provides error reporting and shows a loading progress bar.

SVGClass processes the elements of the SVG file. It processes top level <g> elements in a timer-driven loop, and all lower elements recursively. The timer-driven loop yields to the operating system between each iteration, allowing processing of other mouse and keyboard events. It also restarts the timer that the Flash player uses to display its script timeout message. As <g> elements correspond to layers in Illustrator, the limitation of size becomes one on the layer, rather than the drawing itself; if a drawing starts timing out, the size of layers can be reduced by splitting them into smaller layers. Further advantages of using the timer-driven loop are that the progress bar will increment and each layer will draw on the screen as soon as it is processed. However, it means all graphics must be inside a layer. SVGClass only stores one version of the SVG file in memory – the parsed XML representation.

As each element is processed, the corresponding ActionScript drawing commands are executed. As Flash maintains its own internal display list, effects such as zooming and panning are possible just by changing the scale and origin of the graphic movie clips, and are very rapid. A LIFO stack of style definitions is maintained allowing style attributes to be attached to any element and inherited by child elements. To draw text, the font must have been loaded into the library, and if it is not, the text simply does not appear. ActionScript provides no way of testing whether a given font is in the library.

Although our code is written from scratch we have employed some of Triolo's ActionScript code techniques including her use of Groleau's Bezier handling [22], and Neff's sample code for SAX-style recursive event-based processing of XML nodes [23]. Sample code is available [24]. The properties and methods of SVGClass are shown in Tables 2 and 3. Most properties, if they are to be supplied, can only be set at initialisation time, i.e. when attachMovie is called. Others are get-only at any time; these are marked with a *.

Table 2: Methods of SVGClass

Methods	
panDisplayX (amount)	Shifts the display in X by an amount in SVG file coordinates

panDisplayY (amount)	Shifts the display in Y by an amount in SVG file coordinates
zoomDisplay (factor)	Zooms the display in or out by a factor
defaultDisplay ( )	Returns the display to the zoom and pan settings it had at initialisation.
print (level)	Prints, using the full available resolution of the printer. Note: prints the entire level, not just the SVG display object.

Table 3: Properties of SVGClass (*=get-only)

Properties	
src	URL for the SVG file to display. This is the only mandatory property at initialisation time.
scale	The default display scale.
panX	The default X offset in SVG file coordinates.
panY	The default Y offset in SVG file coordinates.
x	Sets x ordinate of left of the viewbox of the SVG display on the Flash stage.
y	Sets y ordinate of top of the viewbox of the SVG display on the Flash stage.

width	Sets width of the viewbox of the SVG display on the Flash stage
height	Sets height of the viewbox of the SVG display on the Flash stage.
backgroundColor	Sets background color of viewbox. If not provided, background will be transparent and pan cursor will not be visible.
onDisplayDone	Pointer to function to call once the rendering of the SVG graphics is complete. Because of the asynchronous nature of the loading operations, it is important that any functionality that follows on uses this function callback.
styleSheet	URL of style sheet
status*	Status of the SVG display object after creation. If zero, SVG file was drawn successfully
error*	Contains error message if status not equal to 0.
topDepth*	The base depth of the object is provided as a parameter to attachMovie. This property returns the highest drawing priority used by the object and its parts.
layers*	Object containing all the layers of the displayed drawing. For example, a top level layer called “fred” in Illustrator can be accessed programmatically as a child object called <svg object>.layers.fred. As each child object inherits from MovieClip, a number of useful properties are inherited including _alpha and _visible.
numLayers*	The number of layers in the display

6. RESULTS

The outcomes of our research are:

- Vector maps and drawings in SVG files exported from Adobe Illustrator versions 9, 10 and CS can be drawn within the Flash player (see figure 3 [25]).
- Zooming and panning work more quickly, and zooming maintains quality, compared to equivalent raster techniques (see figures 4 [25] and 6 [26]).
- Various animated effects can be applied to the vector graphics (see figure 5 [26]).
- Any graphic element – line, polyline, polygon, text etc. - or combination of elements, can be a made into a clickable link with the hotspot clipped correctly to the graphic's shape. It is much easier to make a graphic clickable in SVG than equivalent raster techniques.
- Printing is far higher quality than equivalent raster techniques, because lines and text are re-rendered at printer resolution.
- Styling can be applied either from inline attributes or from an external style file.
- Visibility and even variable transparency, of layers can be controlled through single lines of ActionScript code – easier than creating separate GIF file layers.
- The SVGClass component can be dragged from the Flash MX Library and at minimum requires just one parameter – the source URL of the SVG file. This makes it easy for Flash developers to get started.
- Methods and properties are accessible to the advanced ActionScript programmer.
- There is potentially much more programmatic control possible over layers and individual graphic elements, compared to equivalent raster techniques. For example Illustrator can attach a name to any graphic element which is saved as the id attribute in the SVG file and could be used to drive ActionScript features.
- Due to the inherent slowness of the ActionScript code, there is a limit of around 200Kb on the size of the SVG file that can be processed in a reasonable timeframe. A number of optimisations are possible, and are noted in the sample code, such as removal of all inline style definitions from the SVG file and relying totally on the style file. This removes verbose XML from the SVG file and reduces processing time.

7. CONCLUSION

Developers producing graphics for web pages from regularly updated vector sources, particularly dynamic sources like database queries, require the separation of style and content, akin to the cascading style sheets of HTML. This implies a vector display technique as opposed to the more common raster technique. Our conceptual ideal is a combination of the Macromedia Flash Player and the open, XML based W3C SVG vector graphics standard. We have implemented proof of concept code that displays a realistic subset of the SVG standard in the Flash Player using the ActionScript scripting language. Extensive tests, using source vector drawings from our Planning and Building Web site, demonstrate many practical advantages for displaying drawings and maps. Beyond that, we find that by integrating this vector technique with the rest of Flash, we can develop many refinements in dynamic presentation and user interaction.

8. FUTURE DEVELOPMENT

Our implementation in ActionScript is satisfactory as a proof of concept but inherently inefficient, being interpreted at run time. This will always limit the size of drawing (in terms of SVG file size) that can be viewed in a reasonable time frame because the processing speed is too slow. Thus the technique as currently implemented works best for fairly simple drawings and. We would like to display SVG files of several megabytes but this would require much of our implementation to be re-coded as part of the Flash Player itself. It is an interesting question as to whether sometime in the future Macromedia might themselves add an SVG-driven MovieClip component to ActionScript.

Figure 3: West Cambridge Master Plan as SVG Vector File rendered by Flash Player

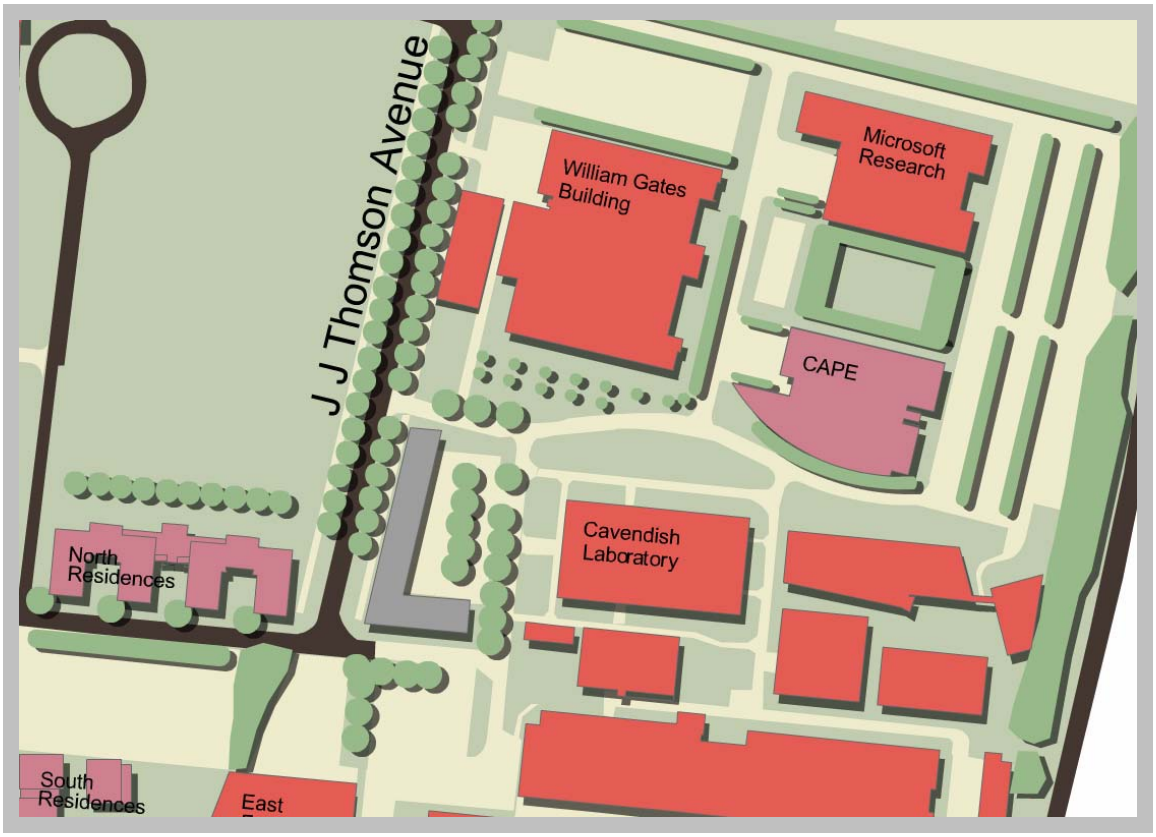

Figure 4: Zoomed in Detail rendered by Flash Player

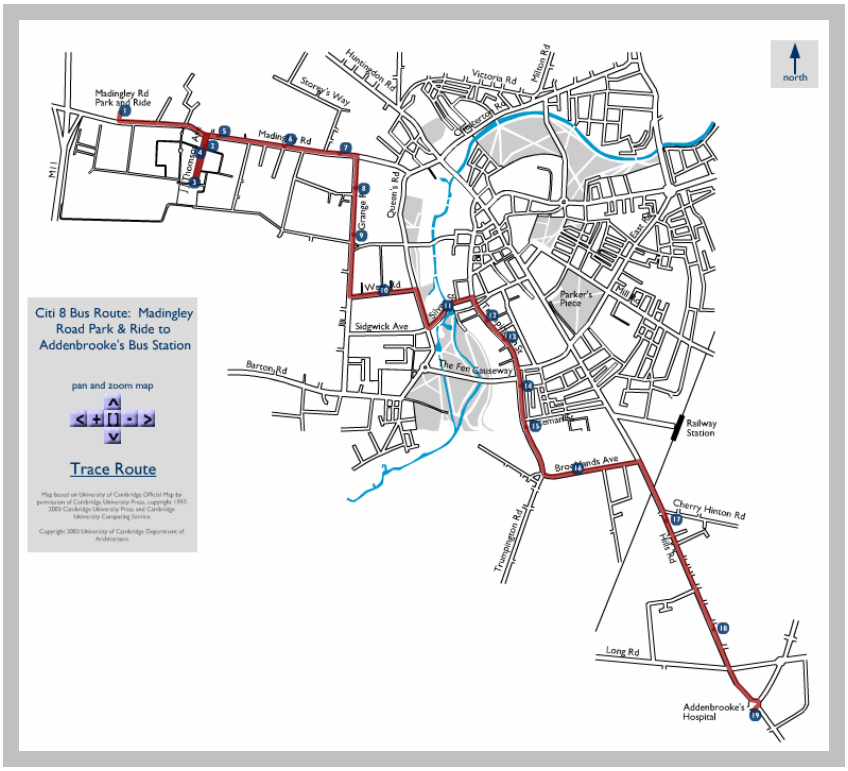

Figure 5: Map of Cambridge as SVG Vector File rendered by Flash Player

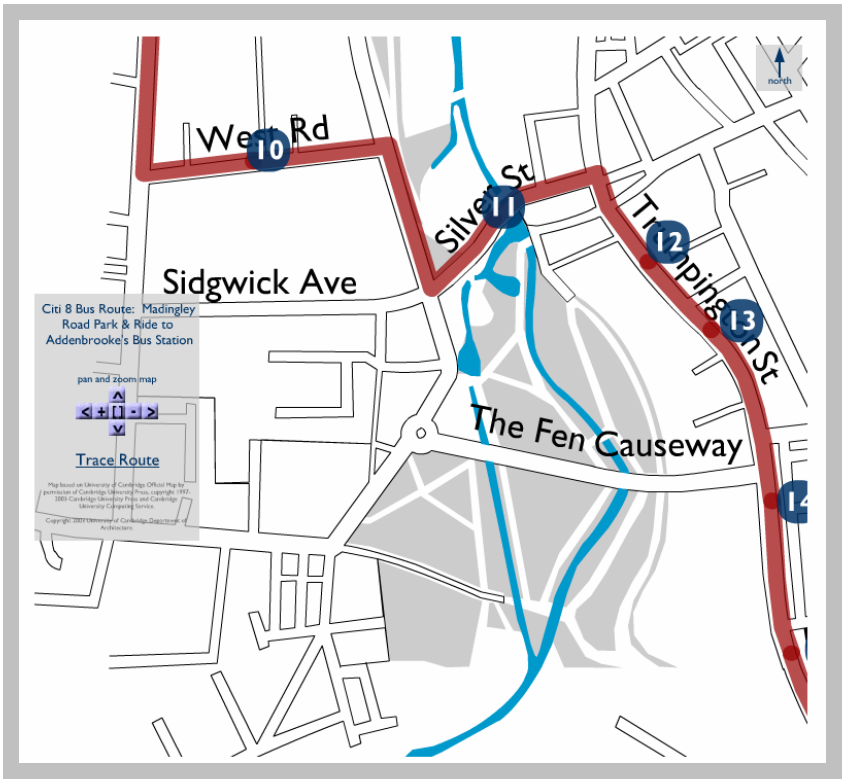

Figure 6: Zoomed in Detail rendered by Flash Player

References

1. University of Cambridge Planning and Building Web Site, Home Page. http://www.cam.ac.uk/building/ [23-01-2004]

2. Ruffle, S. J., Richens, P. N., Web Based Consultation for Cambridge University's Building Program, in: Jabi, W., ed., Reinventing the Discourse: Proceedings of the Twenty First Annual Conference of the Association for Computer-Aided Design In Architecture, Association for Computer-Aided Design in Architecture, 2001, pp366-371.

3. Grant reference GR/N27798/01. http://www.epsrc.ac.uk [23-01-2004]

4. AutoDesk, Inc., General DXF File Structure. http://www.autodesk.com/techpubs/autocad/acad2000/dxf/general_dxf_file_structure_dxf_aa.htm [23-01-2004]

5. Adobe, Inc., Adobe Illustrator CS. http://www.adobe.com/products/illustrator/main.html [23-01-2004]

6. Adobe, Inc., Adobe PhotoShop CS. http://www.adobe.com/products/photoshop/main.html [23-01-2004]

7. University of Cambridge Planning and Building Web Site, West Cambridge Master Plan 2003. http://www-building.arct.cam.ac.uk/westc/masterplan2003/masterplan2003.html [23-01-2004]

8. University of Cambridge Planning and Building Web Site, Sidgwick Master Plan, Buildings, Clickable Site Plan. http://www-building.arct.cam.ac.uk/sidgwick/schemes/buildings.html [23-01-2004]

9. AutoDesk, Inc., Your design—share it, protect it, DWFit. http://usa.autodesk.com/adsk/servlet/index?siteID=123112&id=3232971 [23-01-2004]

10. W3C, WebCGM Profile. http://www.w3.org/Graphics/WebCGM/ [23-01-2004]

11. Microsoft, Inc., Introduction to Vector Markup Language (VML). http://msdn.microsoft.com/library/default.asp?url=/workshop/author/vml/Default.asp [23-01-2004]

12. W3C, Scalable Vector Graphics (SVG) 1.1 Specification. W3C Recommendation 14 January 2003. http://www.w3.org/TR/SVG11/ [23-01-2004]

13. Adobe, Inc., SVG Zone. http://www.adobe.com/svg/main.html [23-01-2004]

14. Macromedia Inc., Flash Player http://www.macromedia.com/shockwave/download/download.cgi?P1_Prod_Version=ShockwaveFlash [23-01-2004]

15. Adobe, Inc., Adobe Reader. http://www.adobe.com/products/acrobat/readermain.html [23-01-2004]

16. Adobe, Inc., Adobe SVG Viewer for Windows, Release Notes, Version 3.0 (Build 76), System Requirements. Via, http://www.adobe.com/svg/viewer/install/main.html [23-01-2004]

17. Macromedia, Inc., Macromedia Flash (SWF) File Format Specification. 2003. http://download.macromedia.com/pub/flash/flash_file_format_specification.pdf [23-01-2004]

18. Macromedia, Inc., Macromedia Flash Support Center: Using ActionScript. http://www.macromedia.com/support/flash/action_scripts.html [23-01-2004]

19. Triolo, H., Notes on Parsing SVG Path Tags and Rendering in Flash MX, 2003. http://actionscript-toolbox.com/svgnotes.php [23-01-2004]

20. Triolo, H., Shape-defined Movieclips in Flash MX, 2003. http://actionscript-toolbox.com/shapeDefinedMC.php [23-01-2004]

21. W3C, Mobile SVG Profiles: SVG Tiny and SVG Basic, W3C Recommendation 14 January 2003. http://www.w3.org/TR/SVGMobile/ [23-01-2004]

22. Groleau, T., Approximating Cubic Bezier Curves in Flash MX, 2002. http://www.timotheegroleau.com/Flash/articles/cubic_bezier_in_flash.htm [23-01-2004]

23. Neff, S., SAX Processor -- Flash Example. http://www.blinex.com/users/sam/ [23-01-2004]

24. Ruffle, S. J., Richens, P. N., A Technique for Drawing SVG Vector Graphics on the Web: On Line Resources. http://www-digitalstudios.arct.cam.ac.uk/websites/svgflashsamplecode/ [23-01-2004]

25. West Cambridge Master Plan: Vector Format. http://www-building.arct.cam.ac.uk/westc/svgflash/masterplan2003.html [23-01-2004]

26. University of Cambridge Planning and Building Web Site, Citi 8 Bus Route. http://www-building.arct.cam.ac.uk/westc/infrastructure/citi8a.html [23-01-2004]