Editor: Peiya Liu
Siemens Corporate Research

Scalable Vector Graphics

Antoine Quint
Fuchsia Design

The Scalable Vector Graphics standard describes graphics that are 2D, interactive, and animated. SVG, an XML grammar, grew out of an effort of the World Wide Web Consortium. Specifically, SVG emerged from the W3C's Document Formats group that formed in 1998 mainly on the basis of two submissions: the Vector Markup Language (VML) and the Precision Graphics Markup Language (PGML). A broad group of industry leaders—including Adobe, Corel, Apple, Macromedia, Microsoft, Sun, Hewlett-Packard, Canon, Kodak, and ILOG—joined the SVG Working Group and began work on SVG 1.0, which eventually became a full-blown specification in September 2001. Today, SVG remains a patent-free, public, and ongoing project. Quite frankly, it's such a versatile format that it's difficult to categorize in one simple acronym.

The main idea motivating SVG was simple: to create a generic document-oriented solution for graphics that can be adapted to modern media. The W3C is an ideal place to foster such an endeavor. In 1998, the Web was evolving quickly. Content creators demanded more than what HTML could provide to cope with new goals in multimedia communication. At that time, raster graphics technologies were mostly proprietary and used in unlikely ways to support HTML, which led to poor design patterns. Vector graphics, on the other hand, were gaining momentum through Macromedia's proprietary Flash technology. Over the past few years, the W3C working group has adapted the SVG design according to these trends, making sure to bring the technology to the largest audience possible as an open standard developed and approved by both the industry and the Web community.

The SVG specification isn't just a format for fancy graphics. It's a serious application designed by experts to match the most recent advancements in 2D graphics. Consider the SVG specification to be a rich 2D graphics application programming interface along the lines of Quartz, Java2D, or GDI+. Where SVG differs from these other APIs is in its declarative, XML-based, portable, and document-oriented architecture. SVG lets you come up with rich graphics compositions and applications without the usual tradeoffs. The format can help you achieve accessibility, semantic value, and seamless integration with the ever-expanding XML world.

Graphics features

First and foremost, SVG can be seen as a drawing solution and, as such, features all the basic vector graphics primitives typically found in drawing systems. These include lines, polylines, polygons, rectangles, circles, and ellipses. On top of these basic shapes, SVG also supports paths, including cubic and quadratic Bezier curves and elliptical arcs. These graphical objects each have different characteristics but they all share transformations and styling features. In SVG, you specify transformations on objects either as an additive instruction (scale, translate, rotate, or skew) or as a global transformation matrix. And you can do styling in SVG through the use of Cascading Style Sheets (CSS).

You can also group all graphical objects together in SVG and organize them hierarchically. As with XML, SVG lets you organize according to a tree structure. Through inheritance, the grouping and structuring capability offers great flexibility for both transformations and styling. For example, you could scale a group of three distinct graphical objects with one single transformation instruction. Similarly, you could fill all objects with a particular color by using a single styling instruction at the group level. You can also give all SVG elements individual XML IDs so you can reference them easily in other parts of the document. That way, by using one simple reference, you can define symbols and reusable components that can be drawn in as many situations as needed in the rest of the document.

SVG also offers advanced graphics capabilities

1070-986X/03/$17.00 © 2003 IEEE

Published by the IEEE Computer Society

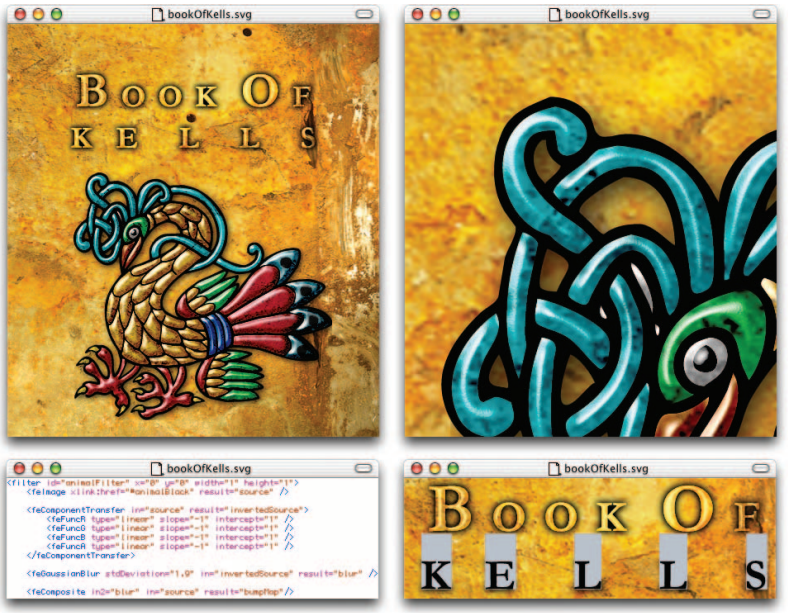

Figure 1. Example of SVG's ability to scale, transform, and filter graphics.

with support for stroking, solid fills, gradient fills, alpha transparency, clipping, masking, and filters. You can fill objects either with a single color (specified in any desired color space) or a gradient (both linear and radial). In SVG, strokes and fills each support independent alpha transparency. You can turn any SVG graphics into a mask or a clipping path, which you can then apply to any other graphics in the document. SVG also comes with a broad set of built-in filter effects. The usual suspects are present here, including blurring, shading, blending, turbulence, lighting, color-matrix tweaking, displacement maps, and many other effects. In fact, you can composite filter effects together to create more complex effects that you can then apply to any SVG graphics. SVG draws all these filter effects independent of the client's resolution, which means that all these effects scale perfectly.

Despite its name, SVG isn't just a vector-drawing format. It also supports raster images and text. The specification requires compliant implementations to support at least the patent-free JPEG and PNG formats, and it defines a generic way to include raster graphics in a document either as external or inline files. In SVG, you can lay out text precisely in any direction and even along a path. SVG gives you the ability to make text searchable and selectable at all times. SVG considers both text and raster graphics to be regular citizens, which means they can be scaled, transformed, and filtered (see Figure 1).

These features provide a perfect match to what software vendors offer in graphics packages such as Quartz, Java2D, or GDI+. However, SVG offers an alternative to regular programming. SVG remains a platform-neutral and device-independent mechanism to define rich graphics that leverage XML's power and flexibility. In that respect, SVG is a high-level cross-platform graphics layer, and it wouldn't be surpising to see SVG included in future operating systems.

Dynamic features

Although SVG already provides advanced 2D graphics capabilities, it also goes beyond static graphics to provide extensive dynamic capabilities. SVG's dynamism revolves around two main axes: animation and scripting.

Animation

A major use for vector graphics on the Web today is animation. SVG supplies the tools of the trade here, too. Concurrently with SVG's development, the W3C was facilitating work on the Synchronized Multimedia Integration Language (SMIL). In its second version, SMIL introduced an animation module designed to be reused and adapted to any other XML language that would need this kind of functionality. SVG is a host language for the SMIL animation module and includes all of the functionalities defined by the SMIL animation specification as well as all the other SMIL modules that it relies on.

SMIL-based SVG animation, a straightforward approach, is time-based and relies on a single timeline for the whole document. The way you would go about creating an animation in SVG is fairly simple. First, you would target an SVG element, specify what XML attributes or CSS properties you want to animate, and then specify the interpolation values and timing parameters, such as when the animation starts and how long it lasts. Even in simple cases like these, there are two ways to specify your target element: either by having your animation element as a child of your target element or by using an XML linking language to point to your target. You can specify interpolation values using different paradigms: a *from-to* syntax for a simple two-value animation, a *by* method to specify an increment from an original value, or a list of values applied to your element's selected attribute or property.

SVG also makes timing very flexible. You can specify beginning and ending times, duration, or no specific times at all. You can make time values absolute, relative to the loading time of the document, or based on keywords (such as

"mouseover" to specify a mouse interaction as the beginning or end value). The real beauty of timing in SVG animation is that, from its SMIL base, it includes synchronization capabilities too. Using the synchronization mechanism, you can easily start an animation when another is over, halfway through, or has looped any number of times. This capability adds great flexibility to animation creation and to the potential semantics value of your documents.

On top of these basic SMIL features, SVG adds several specific features related to animating transformations, color spaces, and shapes along paths. SVG lets you do shape-morphing using path-data animation. Animation control in SVG is also advanced with looping supported by either time duration or absolute loop counts. By default, SVG executes animations linearly with an equal interpolation between each specified value (like keyframes). But you can also ask for discrete, paced, or even spline-based animation. With spline-based animation, you can specify a Bezier spline as a time function for each interval in your animation.

SVG animation provides for the same kind of animation capabilities that Flash designers have been used to. However, because SVG isn't a frame-based format, it guarantees that you can preserve timing. Unlike Flash, a 1-second animation will last exactly 1 second regardless of the client's processing power. In addition, SVG animation is more flexible and offers greater control over the life of an animation than the Flash format. SVG animation is also semantically rich. In the end, SVG's core difference with Flash is that it's a live document with the most important information preserved in a reworkable form. SVG isn't binary soup. In this respect, SVG is a much better fit for Web-based designs because it fosters information sharing instead of merely offering slick presentation capabilities.

Scripting

SVG animation offers simple declarative interactivity, but its lack of logical control can often limit more advanced interactivity. SVG graphics can react to just about any type of interaction through scripting. Most commonly, SVG implementations offer an ECMAScript interpreter that lets you access, read, write, and tweak your document programmatically.

Because SVG is expressed as XML, all of an SVG document's content could technically be scripted with string operations. However, this would be a tiresome method to use. To avoid this kind of tedium, SVG integrates, relies on, and extends another XML technology: the Document Object Model. DOM is an API designed to access and manipulate parsed XML content. It's an object-oriented model that saves you the difficult task of continuous parsing. SVG 1.0 requires implementation of a selected set of DOM Level 2 modules.

First and foremost, the DOM Core forms the basis for document manipulation. This API gives you the most basic abilities, such as navigating through the SVG tree, accessing particular elements, and changing the value of an attribute. Two other important DOM modules are supported by compliant SVG implementations: events and CSS. These two modules will let you programmatically handle events (a must-have for user interfaces) and tweak CSS data as easily as DOM Core allows XML manipulation.

While technically DOM Core and its friends allow complete access to your SVG document, some tasks will still remain daunting. Because SVG is a graphics format, content creators will want to do things like automated matrix computations. To content developers with these needs, SVG 1.0 extends the ground modules to provide a more flexible user experience. For example, you can get your rectangle's position and size, taking into account all inherited transformations, with a single line of code.

It's important to note that DOM's goal is not simply to enable client-side interaction with the document. It's a complete document creation solution. Developers get to use the same API on the server side to create SVG documents from scratch. Being able to do so on the server also means that you can create SVG contents on the client side, thus providing for a rich user experience and lowering network traffic in client-server applications.

There's a lot more to SVG and DOM than I can explore here. It's important to note that you can use DOM to extend SVG's feature set. DOM makes it easy to create reusable components for new higher level graphic objects to share with customers or fellow developers.

Latest progress

While SVG 1.0 was a big achievement for the SVG working group and the graphics industry in general, work on SVG is still evolving. Indeed, the SVG 1.0 specification is a large document at about 800 printed pages. After finalizing the first version in September 2001, the working group started

Further Reading

For more information on SVG, visit these Web sites:
http://www.w3.org/TR/SVG/
http://www.w3.org/TR/SVG11/
http://www.w3.org/TR/SVGMobile/
http://www.w3.org/TR/SVG12/

work on SVG 1.1 with two goals: modularization and profiles for mobile devices. While offering the same feature set as SVG 1.0, version 1.1 will split the specification into different modules that each cover a different set of functionalities.

On the basis of this modularization, we could come up with lighter weight profiles for specific uses. For example, the SVG working group has worked on SVG Mobile, a set of two SVG 1.1 profiles aimed at mobile devices: SVG Basic and SVG Tiny. SVG Basic targets mobile devices (such as PDAs) with sufficient processing power to cope with some of SVG's most advanced features. SVG Tiny is a much more restrictive profile targeted at less powerful devices (such as mobile phones). Both profiles are poised to have dramatic impact on the mobile industry with, for instance, SVG Tiny being mandated by the 3rd Generation Partnership Project (3GPP) for the next generation of mobile messaging service. SVG 1.1 and SVG Mobile were approved as W3C recommendations in January 2003 and many implementations are already available.

Modularizing was a time-consuming but crucial step. It paved the way for more enhancements to the SVG standard. Most recently, the W3C SVG working group has been working on SVG 1.2, which introduces new features with a focus on providing what the developer community is requesting most. The working group has already published three drafts. Noteworthy new features include the addition of native state-of-the-art text wrapping and flowing, SMIL integration enhancements (including audio and video), an extension framework meant to simplify the use of foreign XML grammars for use cases like components and behaviors, and miscellaneous DOM enhancements.

Conclusion

Hopefully, this article has introduced you to the SVG world by highlighting two main SVG themes: graphics capabilities and dynamism. See the "Further Reading" sidebar for leads to lots of other information on the subject. Hopefully, SVG can fill your needs for distributed, lightweight, graphics-centered applications. Obviously, SVG is not the only viable candidate available, but its integration with other rock-solid XML technologies make it an ideal fit for ever-expanding XML workflows. And its open nature will help you immerse yourself in the technology and rapidly leverage its capabilities. **MM**

Antoine Quint is an independent SVG consultant and Invited Expert at the W3C SVG Working Group. Readers may contact him at antoine.quint@fuchsia-design.com.

Contact Standards editor Peiya Liu at Siemens Corporate Research, 755 College Rd. East, Princeton, NJ 08540; pliu@scr.siemens.com.