EG UK Computer Graphics & Visual Computing (2015)
Rita Borgo, Cagatay Turkay (Editors)

GKS-94 to SVG: Some Reflections on the Evolution of Standards for 2D Graphics

D. A. Duce[1] and F.R.A. Hopgood[2]

[1]Department of Computing and Communication Technologies, Oxford Brookes University, UK
[2]Retired, UK

Abstract

Activities to define international standards for computer graphics, in particular through ISO/IEC, started in the 1970s. The advent of the World Wide Web has brought new requirements and opportunities for standardization and now a variety of bodies including ISO/IEC and the World Wide Web Consortium (W3C) promulgate standards in this space. This paper takes a historical look at one of the early ISO/IEC standards for 2D graphics, the Graphical Kernel System (GKS) and compares key concepts and approaches in this standard (as revised in 1994) with concepts and approaches in the W3C Recommendation for Scalable Vector Graphics (SVG). The paper reflects on successes as well as lost opportunities.

Categories and Subject Descriptors (according to ACM CCS): I.3.6 [Computer Graphics]: Methodology and Techniques—Standards

1. Introduction

The Graphical Kernel System (GKS) was the first ISO/IEC international standard for computer graphics and was published in 1985 [GKS85]. This was followed by other standards including the Computer Graphics Metafile (CGM), the Computer Graphics Interface (CGI), a 3D extension of GKS (GKS-3D) and the Programmers' Hierarchical Graphics System (PHIGS). PHIGS was a 3D system providing a modelling capability as well as a viewing capability and was aimed at environments requiring rapid modification of graphical data describing geometrically related objects. At a more conceptual level, the Computer Graphics Reference Model was published in 1992. For an overview of graphics standards of this period see [AD90]. Following ISO/IEC procedures GKS was subject to periodic review and it was revised resulting in the publication of the second edition in November 1994 [GKS94].

Meanwhile, the World Wide Web was launched in 1989/90. Native vector graphics support was unavailable in the early days and plugins only worked on a subset of browsers, so images were often used instead. This changed somewhat with the advent of the Virtual Reality Markup Language in November 1994 (published as an ISO/IEC standard after extensive revision in 1997 [VRM97]). Also, work started on a WebCGM profile as early as 1995 and became a W3C Recommendation early in 1999. The current (at the time of writing) revision was published in 2010 [web10].

It was clear from the early days that a vector graphics format specifically for the web would be a useful addition to the then-available set of markup languages. Chris Lilley at the World Wide Web Consortium (W3C) wrote a requirements document for a scalable graphics language in 1996 [svga] and drawing on ideas from a number of input documents the W3C Scalable Vector Graphics Working Group was set up and produced a first draft document on 11 February 1999 [svgb]. Historically the main influences were PGML from Adobe, IBM, Netscape and Sun Microsystems (based on the imaging model of PostScript and PDF) and VML from Autodesk, Inc., Hewlett-Packard Company, Macromedia, Inc., Microsoft Corporation and Visio Corporation. Key features of the first draft of SVG were the use of XML, integration with style sheets and use of the DOM (Document Object Model) to provide a scripting interface to manipulate document content. SVG then went through an extensive revision process until W3C published the first SVG standard (a W3C Recommendation in their terminology) on 4 September 2001 [svg01a] Since that time, SVG has been further revised. The current W3C Recommendation is version 1.1. and work on SVG2 is in progress. A comparison of SVG1.0 and WebCGM appeared in [DHH02].

DOI: 10.2312/cgvc.20151238

Both ISO/IEC and W3C develop standards through consensus building, the former amongst national standards making bodies, the latter through their membership. Software and hardware vendors were represented in both the GKS and SVG committees, though it is fair to say that user organisations were better represented in the former than the latter. Participation by academics was low in both cases.

Although GKS-94 and SVG have very different origins, and indeed purposes, this paper explores some aspects of the functionality of each and how ideas have evolved over this period. It offers an historical perspective, but also contains some reflections on both strengths and weaknesses of SVG when seen in the context of GKS-94. The primary motivation for this paper is to see how ideas in 2D vector graphics have evolved from GKS to SVG, what new insights have been gained and also what has been lost. A secondary motivation is a plea to not loose sight of the early standardization efforts in the field and to encourage the digital preservation of appropriate documents.

The next section gives some context about the standardization processes of ISO/IEC and W3C. The paper then sets out some historical context then gives brief overviews of GKS-94 and SVG. The paper concludes with reflections on key aspects of these.

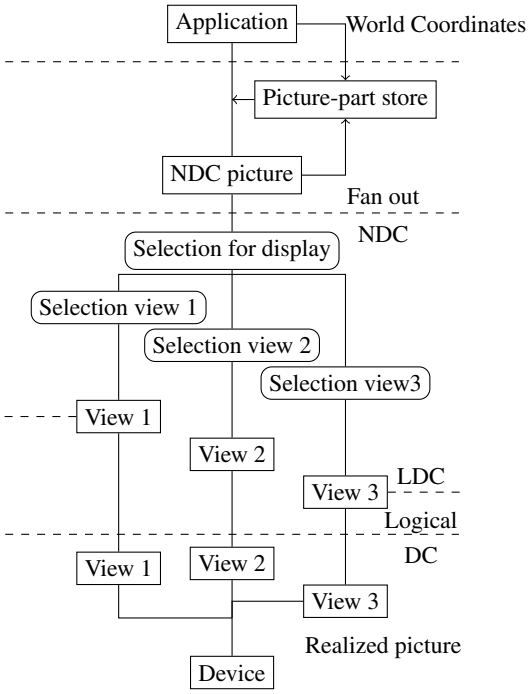

Figure 1: *GKS-94 Architecture.*

2. Historical Context

The first edition of GKS was published in 1985 though development of the computer graphics standards for both 2D and 3D graphics had started in the mid 1970s. In those days it was commonplace for applications to use multiple displays for a single task, for displays to have a wide range of different characteristics and a wide variety of input devices were in use (for example light pens and tracker balls). Vector displays were widespread, raster displays were both rare and expensive. Storage tube devices (such as the Tektronix 4010 and 4014) were popular graphical output devices, and for the time were relatively low cost. Accumulated images could only be moved by flashing and redrawing the whole screen. Continuously refreshed vector displays, such as the Imlac PDS1 were much richer in capabilities, but much more expensive.

It was not unusual for a new graphics package to be developed, or an existing package to be extended, whenever a new device or new hardware was purchased in order to provide convenient access to device features. Providing abstractions to support device independence was an important issue that the early graphics standards sought to address. The workstation concept in GKS was developed with this in mind, a way of representing a device with certain characteristics to which output could be tailored. It is perhaps true that engineering and scientific applications of computer graphics were dominant.

During the late 1980s and 1990s this position changed considerably. Computing evolved from mainframes with attached peripherals to high performance individual workstations with raster graphics capability and thence to laptops and mobile devices in the 2000s along with high-end large scale displays and immersive environments. Operating systems evolved to support multi-window displays through window management systems. In the 1990s it was commonplace for a single display to be used per task and for the mouse to be the most common input device.

3. An Overview of GKS-94

An early event in the revision of the 1985 edition of the GKS standard was a GKS Review Workshop held in September 1987 [gks87]. This explored the need to revise GKS in the light of subsequent developments in other standards including GKS-3D, PHIGS and work on a Computer Graphics Reference Model initiated in July 1985 [cgr]. Particular areas of attention at that workshop were segments and storage, primitives, and input.

An overview of GKS-94 at the end of the review process was published by Brodlie, Damnjanovic, Duce and Hopgood [BDDH95]. This highlighted a number of areas including: normalized device coordinate (NDC) picture, primitive classes, output attributes, viewing, windowing environments, and input.

The overall architecture of GKS-94 is shown in figure 1.

The central part of GKS-94 was the normalized device coordinate (NDC) picture. Primitives defined in world coordinates by the application were transformed to NDC by a normalization transformation (a window to viewport transformation). Primitives could be stored at this level in a picture part store and then assembled to form a scene in the NDC picture. A key property of the NDC picture was that it's contents were well-defined and hence the NDC picture could be exported to an NDC metafile (Part 2 of the standard) and an NDC metafile could be inserted into the NDC picture. Picture parts could be archived from a picture part archive (Part 4 of the standard) and retrieved from an archive.

Workstations could then view the NDC picture through selection criteria (determining which primitives were to be displayed on which workstation and a coordinate transformation, the workstation transformation. This clear separation of levels, world then NDC picture, leading to the realized picture, plus the concept of well-defined picture content and storage, were consequences of the Computer Graphics Reference Model and addressed shortcomings that had been identified in GKS-85.

GKS-94 also provided a much richer set of output primitives than GKS-85, arranged in classes, including curve (set of polyline, set of NURB, set of conic sections), marker, area (set of fill area, set of closed NURB, set of elliptic sector, set of elliptic segment, set of elliptic disc), character (text), image (cell array) and the design primitive. The design primitive drew inspiration from PostScript and some formal modelling of the GKS primitives [DF86] and was based on the idea of extruding a tiling through a stencil, but in a recursive way so that tilings could be constructed recursively from design primitives.

The attributes of primitives in GKS-94 were organized in four classes: identification, NDC, source, logical. Another key idea in GKS-94 was the idea that the attributes of primitives could be determined by a binding process operating over the different levels in the model. For example, the geometry of a primitive would be bound at the NDC picture level as would various text attributes such as TEXT HEIGHT and TEXT ALIGNMENT. The colour of a primitive might be bound at the NDC level (indicating that the primitive should be rendered in the same colour on different devices, or in other words that colour had some specific meaning for the primitive), or alternatively binding might be deferred to the workstation (or logical picture level) in which case a bundle index bound at the NDC level would determine attribute values at the logical picture level by lookup in a workstation-specific bundle table. Taking the curve primitive as an example, the source attributes consisted of the bundle index (CURVE INDEX) and a set of attribute source flags (ASFs) which specified whether the logical attributes of the primitive (CURVE TYPE, CURVEWIDTH SCALE FACTOR, CURVE COLOUR SPECIFIER) were to be determined by a logical attribute bound to the primitive at NDC level or by the CURVE INDEX bound at NDC level. This approach provided a flexible mechanism for defining, on a primitive by primitive basis, which attributes were to be workstation dependent (through the bundle index mechanism) or workstation independent (through logical attributes bound at the NDC level). In other words this provided control over attributes that were used to differentiate one set of primitives of the same class from another, in a workstation dependent way, perhaps using colour on a workstation capable of colour output, or curve type (solid, dashed, etc.) on a workstation with only a monochrome capability.

Another significant enhancement to GKS-85 was the functionality to support clipping. In GKS-85 a clipping indicator was associated with each normalization transformation and the application could choose whether or not to clip primitives to the rectangular boundary defined by the transformation's viewport. This was generalised considerably in GKS-94, perhaps as a consequence of advances in both clipping algorithms and hardware performance and the functionality in systems such as PostScript (which permitted clipping to a clip path constructed from line and curve segments [Ado88]. In GKS-94 all primitives had an associated SCISSOR SET attribute, consisting of a set of named scissors. A scissor consisted of a clipping indicator, clipping rectangle set, shielding indicator and shielding rectangle set. Clipping rectangles and shielding rectangles were defined in NDC space. Functionality was also included to add and remove sets of scissors from the NDC picture. A scissor mode determined whether scissoring took place after the global and local transformations had been applied but before the primitive was dispatched to workstations, or was deferred until after the workstation transformation had been applied. In some respects this was more general than the PostScript model (shielding), but in others less general (description of boundaries).

A set of names (rather than a single name) could be associated with a primitive at NDC level and this set of names could be used, together with application-specified selection criteria, to determine which primitives should be displayed on which workstations, which primitives were to be highlighted or pickable by a given input device, etc. Selection criteria were constructed from boolean operators (not, and, or) and set-theoretic operators (contains(NameSet), isin(NameSet), equals(NameSet), Selectall and Rejectall). Thus application-defined names played a central role in controlling GKS-94.

The input functionality was based on the notion of logical input devices, devices that could return values of particular types (LOCATOR, VALUATOR, CHOICE, PICK, STRING and COMPOSITE) and could operate in one of three modes (REQUEST, SAMPLE and EVENT). However, since the input model in SVG is very different to this model, this aspect of the GKS-94 functionality will not be considered further in this paper.

4. An Overview of SVG

4.1. Markup language

This description is based on the current W3C Recommendation, SVG version 1.1 (Second edition) [SVG01b]. Work is in progress to define SVG version 2, but this is still very much a work in progress at the time of writing and is not a stable basis for comparison.

The SVG document describes five basic concepts: graphical objects, symbols, raster effects, fonts and animation. This section is structured slightly differently, starting with the underlying model and coordinate systems and focuses on those aspects of SVG that have a natural counterparts in GKS-94.

SVG is expressed as a markup language conforming to the rules of XML. Nowadays SVG can either be used as a stand-alone markup language, perhaps linked from documents in other markup languages or as a set of element tags within an HTML5 document for embedded graphical content. Being based on XML, an SVG document is naturally tree-structured. Full advantage is taken of this in SVG through the group element (g) which can contain graphical content and be nested to arbitrary depth. Rendering in SVG is based on the painter's model and the tree-structure of an SVG document defines a rendering order (though groups are first rendered to an offscreen canvas to which group-level effects are applied).

In the general case, SVG output is rendered by an SVG user agent into a viewport which is determined by negotation with a parent user agent. In a simple case, width and height attributes of the root svg element determine the width and height of this window and an initial user coordinate system in which one unit in the user coordinate system is one pixel in the viewport. More complex usage allows a viewBox to be defined (as min x, min y and width and height attributes) which establishes a coordinate system that is mapped to the viewport and also allows for nested SVG elements and control over the mapping when the aspect ratio of viewBox and viewport differ. A feature of SVG is that the origin of the coordinate system is the top left hand corner of the viewport, x increases left-to-right and y increases top-to-bottom.

2D coordinate transformations can be applied to groups of objects as well as individual objects and include the usual set of translation, scaling, rotation and skew as well as specification by a 3×2 matrix. Nested g elements are one way to specify a sequence of transformations.

Graphical objects in SVG fall into three classes, shapes, text and images. The shapes class provides elements to define basic shapes such as rectangles, circles, ellipses and general shapes using a path element. The text primitive provides a rich set of functionality for rendering text content, but the details will not be elaborated further. The image primitive indicates that the contents of a complete file are to be rendered into a given rectangle within the current user coordinate system. Conforming SVG viewers are required to support at least JPEG, PNG and SVG format files. Thus SVG files can be nested through the image element.

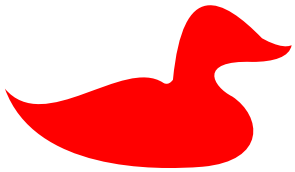

Figure 2: *Output from path element.*

The path primitive is the most general way of specifying geometry. An attribute of the primitive, the 'd' attribute, is a string defining the outline of the shape. The string can contain moveto, lineto, curve, elliptical arcs and closepath specifiers. Each specifier is denoted by a single letter (M, L, etc.) followed by coordinate parameters. There are shorthands for horizontal and vertical lines and corresponding upper and lower case letters denote absolute and relative coordinate positions. Quadratic and cubic Bezier curves may be specified (Q, C) and there is a shorthand for piecewise continuous curves (T, S). The interior of a shape may be filled and the boundary stroked, in a variety of ways. Some functionality in SVG requires distance-along-a-path to be computed. User-agents are advised to use approximation algorithms for this in complex cases (e.g. elliptical arcs), though an attribute is provided to enable the author to specify a precise path length. An example path is shown below. The corresponding scene is shown in figure 2. This shape was used as a running example in [HDGS83], with an outline represented by a polyline.

```
<path fill="red"
  d="M 0 312c40 48 120-32 160-6
  c0 0 5 4 10-3c10-103 50-83 90-42
  c0 0 20 12 30 7c-2 12-18 17-40 17
  c-55-2-40 25-20 35c30 20 35 65-30 71
  c-50 4-170 4-200-79 z"/>
```

Collections of shapes may be reused through the defs, symbol, use mechanism. Elements within a defs element or a symbol element are treated as definitions and are not rendered. Group (g) elements may also be used within a defs element to create a similar effect to symbol. The use element instantiates a symbol at a given position and with an optional transformation.

Clip paths can be specified by the clipPath element and may include shape as well as text elements and use elements. The example below, figure 3, shows a red duck, clipped to the outline of the text 'GKS' displayed on top of the same duck path element filled in green. The path definitions are abbreviated to '...'.

```
<clipPath id="myClip">
<text x="0" y="400" style="font-size:220px;
  font-family:Calibri;
```

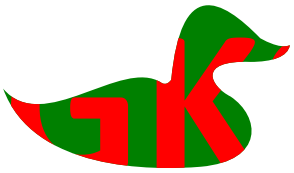

Figure 3: *Clipping to a text element.*

```
  font-weight:bold">GKS</text></clipPath>
<path fill="green" d="..."/>
<path style="clip-path:url(#myClip)" fill="red"
  d="..."/>
```

All graphical output elements may have an id attribute which assigns a unique name to the element (within the scope of the document) and a class attribute which assigns a name or set of names to an element. The appearance of SVG primitives is determined by a rich set of properties that determine the appearance of interior filling (e.g. by solid colour, gradient colour, patterns) and boundary strokes (e.g. stroke width, colour, shape to be used at end of open subpaths). Determining the values of these properties for a given primitive uses Cascading Style Sheets (CSS) as well as a set of presentation attributes (XML attributes, such as color, of the elements that represent graphical objects such as path, circle and text).

CSS styling of SVG documents is very similar to CSS styling of HTML documents. Appearance properties are specified by declarations (name: value pairs). A CSS rule takes the form selector {declarations} where the selector may operate over document structure (e.g. g>path meaning all path elements with a g element parent), class values (path.cpu meaning all path elements with class attributes containing the class name cpu), attribute values (circle[r=5] all circles with radius 5) and identification attribute (path#workstation, the path element with an id attribute whose value is workstation). CSS rules may be grouped into stylesheets and style may cascade from stylesheet to stylesheet starting with browser defaults, then internal and external stylesheets and finally inline style (specified by a style attribute on an SVG element).

A simple example of CSS styling is shown below. The values of the class attributes on the two path primitives determine which style is to be applied. Again path definitions have been omitted. This example also illustrates the g element and the transformation attribute. The output is shown in figure 4

```
<style>
path.pi1 {stroke:black; stroke-width:5;
  fill:green}
path.pi2 {stroke:blue; stroke-width:5;
  fill:orange; opacity:0.5}
</style>
<path class="pi1" d="..."/>
```

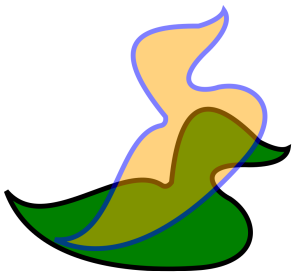

Figure 4: *Simple CSS Styling.*

Figure 5: *Simple CSS Media Queries.*

```
<g transform="translate(50,362) rotate(-45)
  translate(0,-312)">
<path class="pi2" d="..."/>
</g>
```

CSS supports so-called media queries [CSS12]. For example one can write selectors such as media screen and (max-width: 300px) meaning that the rule will only be applied if the media type is screen and the maximum width feature has value 300 pixels. A wide range of media types and media features are recognised. To illustrate this with a simple example, if the style element in the example above is changed to that shown below, if the screen width is initially less than 400px, the two ducks will be shown in grey as in figure 5. If the screen width is more than 400px the appearance is the same as for the previous example (figure 4). If the scene is displayed in a web browser, expanding and shrinking the browser window results in a transition from one to the other.

```
@media screen and (min-width:401px) {
path.pi1 {stroke:black; stroke-width:5;
  fill:green}
path.pi2 {stroke:blue; stroke-width:5;
  fill:orange; opacity:0.5}
}
@media screen and (max-width:400px) {
path.pi1 {stroke:none; fill:gray}
path.pi2 {stroke:none; fill:black;
  opacity:0.2}
}
```

To have a working implementation quickly, a major mis-

take was the way in which presentation attributes and CSS styling interact. Styling always takes precedence over presentation attributes. Hence mixing the two forms of property control can produce unexpected results and the W3C Recommendation warns against doing this. This has caused significant problems ever since.

An important feature of SVG is the declarative animation functionality. Relatively simple XML markup can be used to animate a wide range of properties including attributes such as transformations, object sizes and start positions, appearance properties such as colour, opacity, and geometry (the d attribute of the path primitive). Animations can be chained so that one starts when another ends, and can be initiated by user generated events such as clicking on a graphical object. The animation functionality being represented by XML markup, it can be readily generated by programme or XSLT stylesheet.

Another consequence of XML markup is that SVG content can be dynamically created and modified by scripting (e.g. in JavaScript) using the Document Object Model (DOM) interface, which provides the ability to modify element content, element attributes, styling properties and the structure of the scene tree (e.g. adding, removing elements, moving content from place to place) together with an SVG specific DOM.

4.2. DOM API and SVG applications

The Document Object Model (DOM) provides a set of interfaces to enable programming languages such as JavaScript and Java to create and modify the content of an SVG document. The DOM can be thought of as presenting a view of the document as a tree-structured collection of objects each presenting an interface dependent on the object type, for example be it an object representing an element or an attribute. There are also methods to traverse a tree and to locate content within a tree (e.g. selecting objects by the type of element they represent (e.g. path element), or by the value of an ID attribute which is unique with a document. Some DOM interfaces are shared across all XML documents (e.g. to obtain the type of a node, or a node's parent or child nodes), others are specific to SVG (e.g. the interface to path data). The SVG specific interfaces are defined in the SVG 1.1 W3C Recommendation, the generic XML DOM interfaces are defined in a different W3C Recommendation.

Interestingly, the SVG DOM in SVG1.1 is defined in a scripting/programming language neutral way, using the Interface Definition Language (IDL) from the Object Management Group (OMG). The SVG1.1. Recommendation also contains bindings to Java and ECMAScript (JavaScript). The language specific bindings follow the language-independent interface definition in IDL, making best use of the features and styles of the programming language concered. This mirrors the separation of the language-independent definition of the API in GKS and the separate language binding standards that provided bindings to specific programming languages.

There are a variety of tools and toolkits that depend on the DOM API. Inkscape is a professional quality vector graphics editor which provides a user interface for creating and modifying SVG content. Illustration tools such as Adobe Illustrator can import and export content in SVG documents. Raphael.js is a small JavaScript library to facilitate the creation and modification of SVG content. There is also an SVG plugin for the popular jQuery JavaScript library that serves a similar purpose. The Apache Batik toolkit is a Java toolkit for applications to use SVG in applications or applets.

Finally, the JavaScript visualization toolkit, D3 [BOH11], uses SVG as a rendering engine and depends on the ability to dynamically create and modify SVG content through the DOM API.

5. HTML 5 Canvas

Although beyond the scope of this paper, brief mention will be made of the HTML5 canvas element and the 2D API which is a popular way to create 2D web graphics for amongst some users. HTML states that'The canvas element provides scripts with a resolution-dependent bitmap canvas, which can be used for rendering graphs, game graphics, art, or other visual images on the fly'. W3C is in the process of standardizing an API providing objects, methods and properties to draw and manipulate graphics on a canvas drawing surface, called HTML Canvas 2D Context. At the time of writing this was at Candidate Recommendation stage though the main browsers have supported earlier versions for some time [can15]. Canvas can draw rectangles, text, paths and images. Canvas path specification bears a resemblance to the SVG element, though the attribute set in canvas is much less rich. Canvas can be thought of as a pixel level API rather than a vector graphics API such as the SVG DOM provides.

6. Reflections

6.1. Picture structure

Taken overall, there are similiarities between the architectures of GKS-94 and SVG. The terminology is different, SVG doesn't use the terminology of a workstation, but it does allow appearance to be tailored to device capabilities and, unlike GKS-94, to user preferences. SVG through the CSS styling mechanism supports the idea that appearance may need to be tailored to device capabilities, through the use of CSS Media Queries, though the range of media features in SVG isn't and would not be expected to be, the same as the set of workstation capabilities recognised in the GKS-94 workstation description table. Also in SVG, scripting can be used to control the content of stylesheets. CSS also provides, through the cascade, mechanisms for users to tailor appearance to their preferences (for example for accessibility purposes).

The SVG designers were not consciously trying to relate SVG to the Computer Graphics Reference Model as the GKS-94 designers were. However, SVG does have a well-defined scene structure, being a tree of primitives with a well-defined ordering (the SVG painter's model). In GKS-94 the scene structure was the NDC picture which was a sequence of primitives. The SVG scene structure can be modified through scripting using the DOM interface in very flexible ways, but the scene structure cannot be modified by declarative animation. In GKS-94 the API provided a REORDER NDC PICTURE function to move a subsequence of primitives from one position in the sequence to another. In the authors' experience moving a sequence of primitives from one part of the tree to another is a frequent requirement in the SVG animations he has developed. Being able to change the order on the fly in a declarative way would be a valuable addition to SVG.

Another important feature of GKS-94 was the picture part store. In SVG the symbol element and the defs element coupled with g elements provide a very flexible storage capability with similar functionality and intent. Symbols and g elements can be named using id attributes (which gives a unique name within a document) and can be referenced from other documents using hyperlinking. Animations that the authors have created make extensive use of stores, though these may be held in an XSLT transformation used to generate an SVG document rather than in the document itself.

GKS-94 distinguished a local transformation and a global transformation. This distinction is not present in SVG, though the tree-structured nature of the SVG scene does enable hierarchical scene modelling with different transformations applied and composed at each level. In this respect, SVG is closer to PHIGS.

More will be said about attribute binding in a moment but it is interesting to note here that GKS-94 allowed delayed binding of appearance attributes to the workstation level, but did not allow delayed binding of geometry. There has been work on Constraint SVG by Meyer et al. [MMM04] which allowed parametrised geometry and constraints on the parameters to be defined and then used a constraint solver to instantiate the parameter values. In a sense though late binding is possible with SVG scripting, a primitive can be defined with dummy values and then a script can provide the actual values. This cannot, however, be done in a declarative way. One might wish for a stronger form of script/declarative animation duality. SVG animation does allow a form of late binding through the value *indefinite*, meaning not yet bound. A value could, for example, be bound using a DOM method.

6.2. Primitives

SVG has a much richer text primitive than GKS-94. A particularly nice feature in SVG1.1 was the ability to provide user-defined fonts in which glyphs were just SVG fragments. Hopgood constructed an animation of Australian road trains which were essentially represented by strings of characters, each character denoting a particular kind of truck with the associated glyph being a graphical representation (which could be animated). Using the SVG animate text along path capability to move the trains around gave a very compact representation of the animation.

The primitive set in SVG has less structure than the GKS-94 set and in some ways is less rich. There are however similarities, given the origins of both the GKS-94 design primitive and the SVG1.1 path primitive in PostScript and PDF. GKS-94 went further than SVG in the types of curves allowed, including NURB curves.

The GKS-94 design primitive had associated stencil and tiling stores. SVG does not have a sub-path store, equivalent to the stencil store. Such a feature would be a useful addition to SVG to permit the re-use of complex geometry.

6.3. Attributes

In GKS-94 appearance attributes could be specified at NDC picture level or could be resolved at workstation level through the use of an attribute index and a workstation dependent bundle table. Which mechanism to use could be specified for each attribute of a particular primitive through attribute source flags. SVG although it has CSS style control and individual presentation attributes, does not use them in this way. That said, CSS can be used to mimic NDC and workstation control, by regarding a media specific style sheet as workstation control and a general stylesheet (applying to all media) as NDC control. In this sense class attributes in CSS encompass both GKS namesets and table bundle indices.

There is no control of the kind present in GKS-94 and fundamentally there is no recognition that properties controlled by attributes might be an intrinsic property of the primitive itself or a property that can be bound later, in a variety of ways, to differentiate one group of primitives from another.

There is an interesting parallel between namesets as used in GKS-94 and class attributes as used in SVG. Both enable applications to associate a set of names with a primitive. Developments in the semantic web have identified one use of class attributes to capture concepts that are meaningful at an application level - one example of this is the use of microformats as class names. Damnjanovic et al. [DDR93] did some proof-of-concept work with namesets and filters in GKS-94 which used namesets in this kind of way.

Filters in GKS-94 were constructed from set-theoretic and boolean operators over sets of names. CSS selectors allow selectors of the form path.class1.class2 (selecting a path element whose class attribute contains both class1 and class2) and path.class1, path.class2 (selecting a path element whose class attribute contains either class1 or class2 (or both)). CSS version 3 also supports a :not(selector) operator.

6.4. Final thoughts

The needs of graphics standards change dramatically over time due to the changing environment. In the early days of graphics standardization, the environment was more standalone and the graphics system could assume more control of the environment. A graphics system might well have had exclusive use of a graphical output device. The capabilities of the most common devices were limited and it was important to minimize factors such as the number of times a screen should be rewritten. Nowadays devices have much richer capabilities but at the same time the environment is now constrained by other agents such as the operating system, window manager, capabilities of graphics cards etc. SVG is most commonly displayed inside a web browser and in that environment it has to co-exist with other standards, which may well have overlapping concerns and competing demands. As a consequence standards nowadays seem to have shorter lifetimes and are subject to many constraints, the interface to the user and accessibility concerns being but two at the current time.

Separating style and content is a contentious issue. It appears relatively clear for textual content (e.g. HTML/CSS), but the separation works less well for graphical content, especially in a mixed environment including text, multimedia and graphics aimed at styling content in general (incuding animation effects). A new road map is needed for this area. In the case of graphics, ISO/IEC standards did endeavour to define a clear separation between cases where the same appearance property is used for differentiation (and the choice of value is to an extent arbitrary) and where it is an intrinsic part of the primitive (and the choice of value should be immutable). It could be argued that authors of web graphics need to distinguish similar cases through separate markup constructs. In SVG if presentation attributes always had higher priority than CSS defined values this would have been achievable, but that was not the approach incorporated in the W3C Recommendation. Another area that remains conceptually problematical, though not one that is explored here, is the markup for the mathematical content. The MathML W3C Recommendation is now Version 3.0 Second Edition, yet it seems that the content/style issue applies here too, especially when mathematical content is mixed with textual and graphical content.

In many respects the level at which SVG operates is much closer to the device than was the case for some of the earlier standards. Constraint SVG [MMM04] has been mentioned earlier in this paper. A higher-level constraint standard (2D and 3D) with a clear separation between the CGRM layers, might help to clarity how SVG and CSS should develop.

References

[AD90] ARNOLD D., DUCE D.: *ISO Standards for Computer Graphics*. Butterworths, 1990. ISBN 0408040173. 1

[Ado88] ADOBE SYSTEMS: *PostScript Language Program Design*. Addison-Wesley, 1988. 3

[BDDH95] BRODLIE K. W., DAMNJANOVIC L. B., DUCE D. A., HOPGOOD F. R. A.: Gks-94: An overview. *IEEE Computer Graphics and Applications 15* (1995), 64–71. 2

[BOH11] BOSTOCK M., OGIEVETSKY V., HEER J.: D3: Data-driven documents. *IEEE Transactions on Visualization and Computer Graphics (Proc. InfoVis) 17*, 12 (2011), 2301–2309. 6

[can15] W3C: *HTML Canvas 2d Context (W3C Candidate Recommendation)*, July 2015. 6

[cgr] Introduction to the computer graphics reference model. http://www.gscassociates.com/pubs/cgrmintr.html. Contains dates of significant CGRM papers. 2

[CSS12] WORLD WIDE WEB CONSORTIUM (W3C): *Media Queries*, June 2012. 5

[DDR93] DAMNJANOVIC L., DUCE D., ROBINSON S.: GKS-9x: Some Implementation Considerations. *Computer Graphics Forum 12*, 3 (1993), C–295 – C–313. 7

[DF86] DUCE D., FIELDING E.: Towards a Formal Specification of the GKS Output Primitives. In *Eurographics '86* (1986), Requicha A., (Ed.), North-Holland, pp. 307–323. 3

[DHH02] DUCE D., HERMAN I., HOPGOOD B.: Web 2D Graphics File Formats. *Computer Graphics Forum 21*, 1 (2002), 43–64. 1

[GKS85] ISO/IEC 7942-1:1985(EN): *Information technology - Computer graphics and image processing - Graphical Kernel System (GKS) - Part 1: Functional description*, 1985. 1

[gks87] GKS Review Workshop. *Computer Graphics Forum 6* (1987), 367–369. 2

[GKS94] ISO/IEC 7942-1:1994(EN): *Information technology - Computer graphics and image processing - Graphical Kernel System (GKS) - Part 1: Functional description*, 1994. 1

[HDGS83] HOPGOOD F., DUCE D., GALLOP J., SUTCLIFFE D.: *Introduction to the Graphical Kernel System, GKS*. Academic Press, 1983. Second Edition 1986. Translated into French, Japanese and Italian. 4

[MMM04] MCCORMACK C. L., MARRIOTT K., MEYER B.: Constraint svg. In *Proceedings of the 13th International World Wide Web Conference* (New York, NY, USA, 2004), WWW Alt. '04, ACM, pp. 310–311. 7, 8

[svga] The secret origin of svg. http://www.w3.org/Graphics/SVG/WG/wiki/Secret_Origin_of_SVG. Accessed 8 June 2015. 1

[svgb] The world wide web consortium releases first working draft of scalable vector graphics (svg) specification (press release). http://www.w3.org/Press/1999/SVG-WD.html. Accessed 8 June 2015. 1

[svg01a] W3C: *Scalable Vector Graphics (SVG) 1.0 Specification*, September 2001. 1

[SVG01b] W3C: *Scalable Vector Graphics (SVG) 1.1 Specification*, September 2001. 4

[VRM97] ISO/IEC: *Information technology - Computer graphics and image processing - The Virtual Reality Modeling Language - Part 1: Functional specification and UTF-8 encoding*, 1997. 1

[web10] W3C/OASIS: *W3C Recommendation WebCGM 2.1*, March 2010. 1