Stylable CGM

Keywords: CGM, CGM Open, DOM, CSS

Dave Cruikshank
The Boeing Company
Seattle
Washington
USA
david.w.cruikshank@boeing.com

Biography

> Dave Cruikshank is a Technical Fellow with The Boeing Company in the area of graphics and digital data interchange. He is the Chief Technical Officer of the CGM Open consortium. He is the chair of the ATA Graphics Working Group and the author of the ATA graphics specifications. He is a member of the ATA TICC Executive Committee. He is also the technical architect of the ATA intelligent graphics model and contributed to the development of the Application Structuring amendment of the CGM standard. He has several years of experience with CGM, SGML, and XML interchange.

Lofton Henderson
Lofton Henderson Consulting
Boulder
Colorado
USA
lofton@rockynet.com

Biography

> Lofton Henderson is an independent contractor with long experience in computer graphics, and especially in Web graphics technologies - WebCGM and SVG. He co-authored the "The CGM Handbook" Henderson & Mumford, Academic Press, 1993, 450 pp). He has long experience on the ANSI and ISO graphics standards committees, and has had a role with several industry consortia defining CGM profiles: ATA, PIP, J2008, and RIF. Currently, he is co-chair of the W3C QA Working Group, Program Director and past chair of the CGM Open Consortium, a member of the W3C SVG Working Group, and a member of two OASIS technical committees, Conformance, and XSLT conformance.

Abstract

The CGM Open Consortium is made up of CGM technology vendors, users, and standards participants. Working with W3C graphics resources, the CGM Open Consortium developed the WebCGM Profile as an approved W3C Recommendation in 1999. Since that time several CGM products have appeared making use of WebCGM to deploy interactive graphics in the Web environment. The interoperability of WebCGM has been demonstrated several times by the vendors involved in CGM Open. The producers and consumers of technical graphics are migrating significant resources to the Web, and WebCGM is a solution designed for and tailored to the efficient and effective management of that resource. Recently, the vendors associated with CGM Open have led the way in prototyping proprietary DOM-like functionality. Now the CGM Open consortium is encapsulating that experience with a standard DOM definition for WebCGM.

While working on an API to WebCGM Viewers (WebCGM DOM), the need arose for communicating sometimes temporary or transient changes to the graphical attributes (style) in the WebCGM instance. CGM inherently contains presentation information through the attributes bound to the graphical primitives. This situation resembles the problem faced by CSS1 with legacy HTML versions (HTML 3 and 4) where presentation attributes were integrated into the instance. Although CGM is a binary format, a WebCGM instance has a well-defined tree structure of named elements, and the CSS approach therefore should be applicable. The CSS solution for HTML is being emulated to cover the requirements for the external styling of WebCGM. Some of the CSS selectors are adopted verbatim, and are combined with styling attributes appropriate to the WebCGM graphics model. Generally, as much as possible of the existing CSS (and other standard) machinery is re-used in the authors' design effort.

Table of Contents

NOTE: This paper concerns the application of CSS (Cascading Style Sheets) styling solutions to the CGM (Computer Graphics Metafile) graphics format. Familiarity with CSS is assumed. Familiarity with basic concepts of graphics formats (i.e. graphical primitive elements, graphical attributes, non-graphical metadata for interaction, etc.) is assumed. A brief summary of particulars of the CGM format will be given.

The authors do not take the design in this document to the level of detail where it could be directly implemented. Rather, we explain and illustrate all of the key principles of the styling solution. Binding the solution into a specific implementable syntax would be a relatively easy next level of refinement of details.

Introduction & motivation

WebCGM is a W3C (World Wide Web Consortium) recommendation that was developed as a collaborative effort between members of the CGM Open Consortium and graphical experts from the W3C. It was developed in response to a W3C document describing requirements for scalable vector graphics in the web environment (see http://www.w3.org/Graphics/ScalableReq). Vendor interoperability of WebCGM has been demonstrated several times since it was approved.

Recently, a requirement was identified in CGM Open to control the behavior of the interaction with dynamic WebCGM graphics. CGM Open initiated a project to develop a standardized way of doing this using a DOM (Document Object Model) . In the process of designing a DOM for WebCGM, the problem of communicating element attribute changes to an implementation of the WebCGM DOM led to the idea of a CSS-like styling solution for WebCGM.

The description of the WebCGM Profile on the Graphics Activity pages at W3C has this to say about WebCGM (see http://www.w3.org/graphics/WebCGM):

'While WebCGM is a binary file format and is not "stylable", nevertheless WebCGM follows published W3C requirements for a scalable graphics format where such are applicable.'

"Not stylable" does not necessarily mean that CSS-like styling could not be applied. The applicability of CSS is confirmed by the following statement in CSS2 (see http://www.w3.org/TR/REC-CSS2/intro.html):

"CSS can be used with any structured document format,.."

WebCGM is a structured document format.

In exploring the application of CSS to WebCGM, early thinking focused on CSS1. CSS1 is the format that was originally applied to legacy HTML (analogous to our approach to

WebCGM). We will use CSS2 as our reference for this specification. The CSS2-CSS1 changes are described beginning with (see http://www.w3.org/TR/REC-CSS2/changes.html):

"CSS2 builds on CSS1 and all valid CSS1 style sheets are valid CSS2 style sheets. The changes between the CSS1 specification (see htt://ww.w3.org/TR/REC-CSS1) and this specification fall into three groups: new functionality, updated descriptions of CSS1 functionality, and changes to CSS1."

Amongst the new functionality, the subsequent bullet list mentions: "An extended selector mechanism, including child selectors, adjacent selectors, and attribute selectors." These features may be handy (especially. attribute selectors) for specifying a styling scheme to apply to WebCGM.

Overview of WebCGM functionality

WebCGM is a profile of the CGM standard (ISO/IEC 8632:1999). It was designed to support both static and dynamic functionalities of technical illustrations in the web environment. A WebCGM file is a streamable binary file that is compact and is well suited for the display of technical illustrations in support of technical documentation.

The profile specifies a comprehensive set of graphical primitives, including polylines, polygons, circles, ellipses, graphical text, compound lines, closed figures, high order curves, and raster capabilities. It also contains a comprehensive set of attribute elements to control the graphical appearance of the primitives, including line and edge colors, widths, and types, text fonts, colors, and sizes.

To support interactivity, WebCGM was developed to support the functionalities of navigation, query, and data extraction. The navigation functionality supports the ability to navigate between graphical objects within an illustration, from a graphical object in one illustration to an object in another illustration, and from a graphical object in on illustration to a text object. The query functionality provides the ability to search for a graphical object based on a property associated with it. The data extraction functionality allows the retrieval of non-graphical metadata associated with a graphical object.

Creating an APS (Application Structure) element is the method of creating graphical objects in CGM. Attributes associated with APS are defined using the APS Attribute element. WebCGM defines a small set of APS types and associated APS attribute types. APS types defined by WebCGM are layer, grobject, para, and subpara. Layer APS have attributes of descriptions and names associated with them. Grobject types have attributes of region, viewcontext, linkuri, screentip, and name associated with them. Para and subpara have attributes of region, linkuri, screentip, and content associated with them. The combination of these APS and attributes provide the basis for supporting the requirements of dynamic CGM files in technical documentation.

Specific problem definition & requirements

Scope

The proposed WebCGM styling solution has a limited scope -- not all aspects of a CGM file need to be controllable with styling, and access to individual graphical primitive elements (and attributes) is out of scope, unless a single graphical element comprises the entire content of an APS. The APS the atomic unit of referenceable structure in WebCGM.

Only a subset of the graphical items enumerated in the following sub-sections need to be controllable. For the functional requirements of this design, the non-graphical metadata does not need to be referenceable by external styling.

Highlight functionality

It was decided that the method of highlighting graphical objects in support of DOM functionality should be viewer dependent (i.e., there should be a 'highlight' attribute, which causes an object to be highlighted in a way which is otherwise unspecified. The method is at the discretion of the viewer).

Control of attributes

The styling solution being pursued is a consequence of a particular requirement on the WebCGM DOM/ API (Application Programming Interface) . The DOM needs to support the capability of setting values and unsetting them as well. That is to say, changing line color from black to red requires the application to "remember" that the original line color was black so that it can be restored. Unsetting values is tricky because it potentially requires knowledge of the initial graphical state of an element. In a true full DOM model, this would be achievable by 'get' functions, which are complementary to 'set' function. However, the requirements of this graphical API have a scope and granularity that by design falls short of a proper, full DOM model, as explained in the following.

The control of attribute values is divided into two levels of granularity: those that apply to the whole picture -- 'pictureScope' -- and those that apply to an entire APS -- 'apsScope'. Individual CGM graphical primitive elements within a picture or an APS cannot be addressed and styled. Whenever an attribute is set, all of the graphical elements of that type (line, edge, text, or fill) that the attribute applies to will be affected. For example, if there are, within an APS, several different line elements with different line colors, all of them will be uniformly changed to a new value. The problem with reversing the change per a classical DOM model is twofold:

- a 'getColor' inquiry at the APS granularity in principle does not have a well defined answer. The APS doesn't have a color. It is the individual line elements within the APS which have the color.

- if the 'get' and 'set' granularity were lowered to individual primitives, it would be very inefficient for accomplishing the limited goals of the WebCGM DOM (for which this styling solution is a supporting component).

It addition to directly setting values of graphical attributes, two other methods of attribute control are required. They are intensity and scaling. Intensity may apply to color attributes or a raster image, while scaling can apply to text height or line and edge weight.

Control of color attributes

This section and the following sections identify the requirements for those graphical aspects of WebCGM that need to be controllable through an external styling solution. The requirements derive from the use case of WebCGM access through a DOM-like API.

The following functions were identified to support control of color attributes:

- PictureScopeLineColor
- PictureScopeEdgeColor
- PictureScopeFillColor
- PictureScopeTextColor
- PictureScopeMarkerColor
- APSscopeLineColor
- APSscopeEdgeColor
- APSscopeFillColor
- APSscopeTextColor
- APSscopeMarkerColor
- PictureScopeRasterIintensity
- APSscopeRasterIntensity

Color attributes may be set absolutely by RGB values or by a relative intensity in the range of zero to one. The relative intensity is a scale factor that is to be applied to the existing color.

Control of line/edge weight

The following functions were identified to support control of line and edge weight attributes:

- PictureScopeLineWeight
- PictureScopeEdgeWeight
- APSscopeLineWeight
- APSscopeEdgeWeight

Line and edge weight may be set absolutely in mm or by a relative scale factor greater than zero. The relative scale factor is applied to the existing line or edge weight.

Control of character height

The following functions were identified to support control of the character height attribute:

- PictureScopeCharacterHeight
- APSscopeCharacterHeight

Character height may be set absolutely in mm or by a relative scale factor greater than zero. The relative scale factor is applied to the existing character height.

NOTE: The width of the restricted text box must be adjusted when the character height is changed.

About the above-enumerated attributes

The above-listed attributes deviate from the actual CGM attribute model. They either don't exist in the CGM model (e.g., character height scale factor), or they behave differently (e.g., the scale factor for line weight is applied to the current modal line weight, not to the implementation-dependent nominal line width as is the case with CGM's 'line width scale factor').

It is also worth observing that the required ways of setting weights, heights, etc, will be independent of and supersede the various CGM modes for those attributes (e.g., the line width specification mode). The same is true of the color attributes (the current color selection mode is over-ridden.)

Key issues & resolutions

The following is a summary list of key issues that had to be addressed in the design of the styling solution. Details are found in the following sub-sections.

- binary or clear text format
  - proposal: clear text
- WebCGM tree model
- Alignment: degree of alignment with CSS, SVG (Scalable Vector Graphics) , etc
  - proposal: not too hard -- joint/shared code is not a likely use case
  - proposal: borrow principles, customize for WebCGM
- what subset of CSS selectors model
- declaration values in style rules
- what names to use to match standard WebCGM elements
  - common names with SVG? [proposal: no]
  - CGM Clear Text names? [proposal: yes -- many CGM implementations have parsers]
- applicability of CSS inheritance model
- cascade, and specificity, precedence

        - user style sheets: applied from outside of WebCGM instance [proposal: yes]
        - author style sheets: embedded or referenced from inside [proposal: not now, maybe later]
        - user agent style sheets: basically, User-agent-dependent defaults [proposal: no]
- interaction of WebCGM (virtual) stylesheet with WebCGM DOM? What are some user scenarios or use cases?
    - Proposal: defer this issue until further WebCGM DOM design.
- what is the processing model, relating WebCGM instance, DOM, viewier, external style sheet, etc?
    - Proposal: defer this issue until further WebCGM DOM design.
- should WebCGM 2.0 introduce a "processing instruction" that allows reference to external style sheet?
    - Proposal: defer this issue until further WebCGM DOM design.

Constraints

Aside from above-enumerated requirements, there is one additional basic constraint:

- don't alter any standard WebCGM syntax or semantics.

Alignment of WebCGM styling with SVG & CSS

It has been observed that there might be the possibility for significant code sharing if the WebCGM styling solution attempted maximal alignment with SVG (e.g., element names) and CSS (e.g., strictly adhering to the cascade order). Presently, we have not seen any use cases to suggest that CGM/SVG code sharing is a scenario of high importance. After preliminary design, it seems that some CSS stylesheet parsing might be reusable, but not much else.

The appropriate solution, given the relatively disjoint application communities, seems to be: borrow the styling principles from CSS, including matching some specific details such as kinds of Selectors, but otherwise customize the solution for the WebCGM DOM requirements.

Binary or clear text

CSS is a clear text format, which makes it really easy to put together stylesheets manually (is "manual" a use case? or are the use cases all automatic machine processing?). The rule to set all H1 elements to blue is: `H1 {color: blue}`

CGM is a binary format. Nevertheless, these transactions between (external) user stylesheets and binary CGM will be through some interface or API (the DOM?), and therefore the translation can be handled by the implementation.

The CGM stylesheets should be text (e.g.,`grobject {color: red})` to turn all 'grobject' APSs to red. (See next sections about modelling the tree and the selectors scheme).

A related issue has to do with some undefined aspects of the CGM DOM/API. How will values of some things like 'linkuri' data be communicated through the DOM? In CGM, they are user-unfriendly SDR (Structured Data Record) In WebCGM, those are binary.

Alternatives

1. each API function would specialize it's parameter list for the information content. e.g.,
    - setScreenTip (apsid, "a simple ApsAttr")
    - setLinkUri (apsid, "first substring", "second substring", "third substring")
2. pass a binary SDR
    - setScreenTip (apsid, ptr-to-binary-coded-SDR)
3. pass a clear text SDR
    - setScreenTip (apsid, "14 1 'a simple ApsAttr'")
    - setLinkURI (apsid, "14 3 'first substring' 'second substring' 'third substring'")

NOTE: This is presented in terms of a simple API in some classical programming language like C. We don't know what it would look like if the WebCGM API were for example built on top of XML (eXtensible Markup Language) XML DOM core, in some O-O programming language, per "An Approach to a CGM DOM" (see http://www.cgmopen.org/technical/approach_to_cgm_dom.html).

Proposal: None yet. If DOM binding were like above in classical programming language, then option #1 is most appealing (but most inflexible). If it is built on generic XML DOM, then #3 looks most appealing.

WebCGM tree model

Modeling APS nodes

To apply a CSS-like styling solution, we need to be clear about a tree model for CGM graphics. This is mainly for the purpose of defining selectors (CSS selectors act on a tree structure). It also affects design issues of inheritance and cascading.

WebCGM tree structure is simple:

--> pictures

----> APS elements (nested)

------> graphical prims/attrs (leaf nodes)

NOTE: Multiple pictures are deprecated in WebCGM 1.0 (2nd Edition).

What are the element nodes at the APS level? This is an issue. If we draw analogy to a markup grammar, there are two choices:

1. `APS type="grobject | para | .." <==> <aps type="grobject | para | .."></aps>`
2. `APS type="grobject | para | .." <==> <grobject ..> | <para ..> | ..`

In #2, the APS 'type' parameter is combined with the APS element itself and modelled as (XML) elements of distinct kinds. In #1, the tree is modelled with one kind of element/node -- APS -- which carries a 'type' attribute (attribute in the XML sense). Although they are equivalent, #2 is most appealing and cleaner when selectors are chosen and applied.

Solution. #2. Model WebCGM as a tree with element nodes of 4 kinds: grobject, para, subpara, layer.

Modeling APS attributes

NOTE: The styling solution applies only to the graphical aspects, not the APS attribute (non-graphical metadata) aspects. Because the latter are which are in the WebCGM DOM, the following analysis is included for completeness and context. From the perspective of styling alone, it can be ignored.

Related to this is how to model the CGM APS attribute elements within an APS. These might be, for example, screen tips or links, (i.e., respectively CGM APS attribute elements whose 'type' parameter is 'screentip' or 'linkuri'). They could be modelled as:

1. child elements of their containing APS
2. XML attributes on their containing element

#2 is most appealing, but there is a problem. Some ApsAttr types can occur multiple times on one element. One way to handle this would be to define an attribute multiplicity and pass a more complex argument that encodes the multiplicity (i.e., if there is a single linkuri, its parameter is of the form of an SDR with three string members: `"14 3 'first-substring' 'second-substring' 'third-substring'"`).

But if there were two linkuri on one APS, then it could be parameterized in the stylesheet (and DOM) in a couple of ways:

1. an SDR with two SDR members
    - "1 2 'first-linkuri-3-member-SDR' 'second-linkuri-3-member-SDR'"
2. a list-of-SDR approach
    - "'first-linkuri-3-member-SDR' 'second-linkuri-3-member-SDR'"

The difference is subtle. The first encodes the two distinct linkuri parameters as a CGM-formatted nested SDR. The second uses the CSS/HTML "list of" approach to parameters.

Neither one is appealing -- by the time you have considered whatever quoting might be needed for the CSS syntax, and the nesting of parameters in the CGM Clear Text SDR coding, you could be looking at several levels deep in nested quotes.

No specific recommendation yet. However, it does seem promising to handle the un-XML-like multiple-attribute problem as a packaging issue in the API/DOM.

Selectors & declarations

As described in the CSS2 Introduction (see http://www.w3.org/TR/REC-CSS2/intro.html), there are two key parts to a CSS rule: "A CSS rule consists of two main parts: selector ('H1') and declaration ('color: blue')".

What subset of CSS selectors for WebCGM styling

CSS Selectors are how you address the elements within the document, to which your styling should be applied. CSS2 has a table of selectors capabilities (see http://www.w3.org/TR/REC-CSS2/selector.html#q1).

Looking at this table, these seem like the absolute minimal selector pattern matching set:

```
* -- Matches any element [universal selector]

E -- Matches any E element (i.e., an element of type E) [type selector]

E#myid -- Matches any E element ID equal to "myid" [ID selector]
```

Also very useful would be the attribute selectors (so that you could match, or example, on 'name' attribute):

```
E[foo] -- Matches any E element with the "foo" attribute set (whatever
the value).

E[foo="warning"] -- Matches any E element whose "foo" attribute value
is exactly equal to "warning".
```

We could accomplish a lot with these. In particular, note a capability that would be available if we were to make the convention that a picture is a type of element. All of the pictureScope and apsScope variants could be handled by the selectors mechanisms, instead of some other special syntax. For example, `picture {lineColor: 50%}grobject[name="foo"] {lineColor: 100%}`would dim all lines in the picture to half-intensity, except for those in the grobject whose name attribute had the value "foo". (Note. This relies on some specificity rules that we haven't explained yet.) Even though multiple pictures are deprecated, you could use the picture element type in combination with ID selector in a multi-picture metafile.

There are other interesting selectors in the CSS selectors table as well, for example the contextual selectors which allow you to match depending on child, descendent, and sibling relationships in the tree. Also, the link pseudo-classes (link, visited) and the dynamic pseudo-class selectors (hover, active, focus) might have some interesting application.

WebCGM element names to use in selectors

Because of the scope of this styling solution, most of the issue of how to name WebCGM elements in styling selectors goes away. If the granularity of the styling solution descended to the individual graphical elements, then it would be tempting to use the names of the CGM Clear Text encoding, since many CGM viewer implementations support that encoding.

However, the only names that are needed are:

- for element selectors: picture, grobject, para, subpara, layer
- for attribute selectors: linkuri, name, screentip, viewcontext, region, layerdescription, layername, content

The question of whether to use CGM Clear Text names becomes moot, since our design scope is limited to this handful of APS and APS attribute types.

We could attempt to force alignment of the names for element selectors, for example, with SVG. But two reasons argue against it: the concepts do not align clearly (e.g., grobject is sort of like SVG g, but layer does not correlate well with any SVG concept; and, there are no use cases (yet) indicating the need).

Declaration of attribute values

As previously noted, there is something of a conceptual disconnect between the graphical attributes that exist in WebCGM, and those that we desire to impose by external stylesheets.

- the attribute types themselves deviate, e.g.,
  - LineColour is the CGM attribute. It applies to individual CGM graphical line primitives.
  - PictureScopeLineColor is the proposed external style rule, and it would apply to all lines in a picture.
- the attribute values themselves, e.g.,
  - LineColour is specified within CGM by either an index (into a color table), or a n-tuple direct color specification in the applicable color space (RGB or sRGB only, in WebCGM, although the ISO CGM standard also allows CIE*LUV, CMYK, etc.)

- PictureScopeLineColor is proposed to have values of RGB or relative percentage (e.g., "50%") to apply to the existing WebCGM color of affected lines.

Given the scope and requirements that are being satisfied by this styling design exercise, this is not a problem. The communication of style information is unidirectional, from outside application to viewer display engine. This only requires that all mappings of the style rule declaration values be well enough defined that the viewer can make the visual effect happen.

This disconnect in the models appears to only become problematic if the scope of the WebCGM DOM design were extended in the future, specifically to including inquiry functions that provided 'get' access to the graphical aspects and attributes of individual elements in the WebCGM instance, as opposed to the current limited scope of metadata aspects. There would be no reason that such an extension could not be made, but it would require a careful definition of the precise effect on each CGM elements virtual state, of each of these styling declarations.

The values assumed by the declarations should align with CSS data types, specifically it looks like the needs are: RGB color, percents (for relative intensity), and lengths (mm appear sufficient).

Inheritance

It seems clear that the graphical styling attributes ought to inherit. (i.e., if you are going to make the "automobile" grobject turn red, then it ought include the wheels). That is, all other potential nested components (APS) ought to be affected.

Cascade, precedence, specificity

The CSS cascade (see http://www.w3.org/TR/REC-CSS2/cascade.html#cascade) defines a number of possible sources of styling: author, user, user-agent default. That is also the basic order of importance: author styles override user styles, user styles override user-agent default styles. There are numerous sources of styles in CSS (some of which are moot for a WebCGM styling solution):

1. intra-element style attributes;
2. a single style element in the document header;
3. @import rules;
4. link elements (HTML) or stylesheet PIs (XML) to external stylesheets
5. (unspecified) user-agent mechanism that allows user to supply stylesheet
6. user-agent default stylesheet

For CGM, all graphical attributes are well-defined, and therefore "user-agent default styles" makes no sense. The CGM basically comes with complete author style

information (including CGM defaults). For this CGM styling application, apparently only #5 is also relevant -- external application of user stylesheet.

It seems clear that we want "user" styles (#5) to have precedence over author styles (the completely defined CGM styles). This could be done by forcing `!important` into every CGM style declaration, but that seems inefficient.

Recommendation. For CGM styling, define cascade precedence the opposite of CSS -- user overrides author.

For specificity, basically the CSS specificity rules (see http://www.w3.org/TR/REC-CSS2/cascade.html#specificity) are okay. When there are clashing styles, the most specific wins. (e.g., `picture {lineColor: 50%}grobject#foo {lineColor: 100%}`dims all lines except those in the grobject whose ID value is foo, because that rule's selector is more specific than the 50% rule).

Processing model

So far, there are no concrete design proposals for the overall architecture of the system.

- How are styling rules communicated to the viewer?
- Is there some sort of rule-by-rule interface?
- Or, do you have to pass an entire stylesheet at a time?
- How do you turn off a stylesheet after you have applied it and want to go back to default? (i.e., unapply the styles)
- How does the styling interface to the viewer relate to DOM? (DOM presumably will 'get' and 'set' all the APSattr parameters)

This is the next stage of design. It is the goal of the current design paper to specify the essential content of a WebCGM external stylesheet, and how its application (by whatever means) affects WebCGM graphics being displayed by a viewer.

References

Minutes of CGM Open meeting March 2001 (see http://cgmopen.org/meetings/cgmo_tech_cleveland_01.html)

Minutes of CGM Open meeting December 2001 (see http://cgmopen.org/meetings/cgmo_tech_orlando_02.html#review)

Minutes of CGM Open meeting June 2002 (see http://cgmopen.org/meetings/cgmo_tech_frankfurt_02.html)

"An approach to CGM DOM" (see http://cgmopen.org/technical/approach_to_cgm_dom.html)

WebCGM 1.0 Second Release (see http://www.w3.org/TR/REC-WebCGM)

CGM technology (see http://www.cgmopen.org'/)